Hybrid I/O Automata ⋆

Nancy Lynch [1]

MIT Laboratory for Computer Science, Cambridge, MA 02139, USA

Roberto Segala [2]

Dipartimento di Informatica, Università di Verona, Strada Le Grazie 15, 37134 Verona, Italy

Frits Vaandrager [3]

Nijmeegs Instituut voor Informatica en Informatiekunde, University of Nijmegen, P.O. Box 9010, 6500 GL Nijmegen, The Netherlands

Abstract

Hybrid systems are systems that exhibit a combination of discrete and continuous behavior. Typical hybrid systems include computer components, which operate in discrete program steps, and real-world components, whose behavior over time intervals evolves according to physical constraints. Important examples of hybrid systems include automated transportation systems, robotics systems, process control systems, systems of embedded devices, and mobile computing systems. Such systems can be very complex, and very difficult to describe and analyze.

This paper presents the *Hybrid Input/Output Automaton (HIOA)* modeling framework, a basic mathematical framework to support description and analysis of hybrid systems. An important feature of this model is its support for decomposing hybrid system descriptions. In particular, the framework includes a notion of *external behavior* for a hybrid I/O automaton, which captures its discrete and continuous interactions with its environment. The framework also defines what it means for one HIOA to *implement* another, based on an inclusion relationship between their external behavior sets, and defines a notion of *simulation*, which provides a sufficient condition for demonstrating implementation relationships. The framework also includes a *composition* operation for HIOAs, which respects the implementation relation and a notion of *receptiveness*, which implies that an HIOA does not block the passage of time. The framework is intended to support analysis methods from both computer science and control theory.

This work is a simplification of our earlier HIOA model. The main simplification in the new model is a clearer separation between the mechanisms used to model discrete and continuous interaction between components. In particular, the new model removes the dual use of external variables for discrete and continuous interaction.

1 Introduction

1.1 Overview

Recent years have seen a rapid growth of interest in *hybrid systems*—systems that intermix discrete and continuous behavior [28,70,12,9,62,10,34,73,80,51,20]. Typical hybrid systems include computer components, which operate in discrete program steps, and real-world components, whose behavior over time intervals evolves according to physical constraints. Such systems are used in many application domains, including automated transportation, avionics, automotive control, robotics, process control, embedded devices, consumer electronics, and mobile computing.

Hybrid systems can be very complex, and therefore very difficult to describe and reason about. At the same time, because they involve real-world activity, they often have stringent safety requirements. This combination of factors leads to a need for rigorous mathematical models for describing hybrid systems and their properties, and for practical analysis methods based on these models.

In this paper, we present a basic mathematical framework to support description and analysis of hybrid systems: the *Hybrid Input/Output Automaton* modeling framework. A *Hybrid I/O Automaton (HIOA)* is a kind of nondeterministic, possibly infinite-state, state machine. The state of an HIOA is divided into *state variables*, and it may also have additional *input variables* and *output variables*. The state can change in two ways: instantaneously by the occurrence of a *discrete transition*, or according to some *trajectory* when time passes. Formally, a discrete transition is a triple consisting of a source

---

⋆ An extended abstract of this paper appeared as [52].

Email addresses: `lynch@theory.lcs.mit.edu` (Nancy Lynch), `segala@sci.univr.it` (Roberto Segala), `fvaan@cs.kun.nl` (Frits Vaandrager).

[1] Supported by PATH 1784-18454LD; AFOSR F49620-00-1-0097, F49620-97-1-0337, and SA2796PO 1-0000243658; NTT MIT9904-12; NSF ACI-9876931, CCR-9909114, and CCR-9804665; multi-sponsored consortium project Oxygen; DARPA F33615-01-C-1850.

[2] Supported by MURST project TOSCA.

[3] Supported by Esprit Project 26270, Verification of Hybrid Systems (VHS), GBE/SION project 612-14-004, Stepwise Refinement of Hybrid Systems, and PROGRESS project TES4199, Verification of Hard and Softly Timed Systems (HaaST).

state, an *action* (for synchronization with other automata), and a target state. Trajectories are functions that describe the evolution of the state variables, along with the input and output variables, over intervals of time. Trajectories may be continuous or discontinuous functions.

HIOAs are intended to be used to model all components of hybrid systems, including physical components, controllers, sensors, actuators, computer software, communication services, and humans that interact with the rest of the system. The framework is very general: for example, we do not require that trajectories be expressible using systems of equations of a particular form, and we do not require that discrete transitions be expressible using a particular logical language. Particular kinds of systems of equations and particular logical languages can be used to define special cases of the general model.

The most important feature of the hybrid I/O automaton framework is its support for decomposing hybrid system description and analysis; this is important because many hybrid systems are too complex to understand all at once. A key to this decomposition is that the framework includes a rigorously-defined notion of *external behavior* for hybrid I/O automata, which captures their discrete and continuous interactions with their environment. The external behavior of each HIOA is defined by a simple mathematical object called a *trace*. The framework also includes notions of *abstraction* and *parallel composition*.

For abstraction, the framework includes notions of *implementation* and *simulation*, which can be used to view hybrid systems at multiple levels of abstraction, starting from a high-level version that describes required properties, and ending with a low-level version that describes a detailed design or implementation. In particular, the HIOA framework defines what it means for one HIOA, $\mathcal{A}$, to *implement* another HIOA, $\mathcal{B}$, namely, any trace that can be exhibited by $\mathcal{A}$ is also allowed by $\mathcal{B}$. In this case, $\mathcal{A}$ might be more deterministic than $\mathcal{B}$, in terms of either discrete transitions or trajectories. For instance, $\mathcal{B}$ might be allowed to perform an output action at an arbitrary time before noon, whereas $\mathcal{A}$ produces the same output sometime between 10 and 11AM. Or $\mathcal{B}$ might allow an output variable y to evolve with $\dot{y} \in [0, 2]$, whereas $\mathcal{A}$ might ensure that $\dot{y} = 1$.

The notion of a *simulation relation* from $\mathcal{A}$ to $\mathcal{B}$ provides a sufficient condition for demonstrating that $\mathcal{A}$ implements $\mathcal{B}$. A simulation relation is defined to satisfy three conditions, one relating start states, one relating discrete transitions, and one relating trajectories of $\mathcal{A}$ and $\mathcal{B}$.

For parallel composition, the framework provides a *composition operation*, by which HIOAs modeling individual hybrid system components can be combined to produce a model for a larger hybrid system. The model for the composed system can describe interactions among the components, including joint participation in discrete transitions and trajectories. Composition requires certain "compatibility" conditions, namely, that each output variable and output action be controlled by at most one automaton, and that internal variables and actions of one automaton cannot be shared by any other automaton. The composition operation respects the implementation relation, for example, if $\mathcal{A}_1$ implements $\mathcal{A}_2$ then the composition of $\mathcal{A}_1$ and $\mathcal{B}$ implements the composition of $\mathcal{A}_2$ and $\mathcal{B}$. Composition also satisfies *projection* results saying that a trace of a composition of HIOAs projects to give traces of the individual HIOAs, and *pasting* results saying that compatible behaviors of components are "pastable" to give behaviors of the composition. Such results are essential if the models are to be used for compositional design and verification of systems. In addition, the framework includes *hiding* operations for output actions and variables, which respect the implementation relationship.

An interesting complication that arises in the hybrid setting is the possibility that a state machine could "prevent time from passing", for example, by blocking it entirely, or by scheduling infinitely many discrete actions to happen in a finite amount of time—so-called *Zeno behavior*. The HIOA framework includes a notion of *receptiveness*, which says that an HIOA does not contribute to producing Zeno behavior, and which (under suitable compatibility conditions) is preserved by composition. We also give simple sufficient conditions for these compatibility conditions to hold.

The generality of the HIOA framework means that a large collection of analysis methods, derived from both discrete and continuous analysis methods, can be applied to systems modeled as HIOAs. For example, inductive methods for proving invariant assertions and simulation relationships (see, e.g, [58,72]), which are commonly used in computer science for reasoning about discrete systems, can be extended to the hybrid setting and expressed by theorems about HIOAs. Other discrete analysis methods that should be extendible include proving progress using well-founded sets (see, e.g., [26]), assume-guarantee compositional reasoning (e.g., [36,16]), and deducing properties within temporal logic and other logical formalisms. All of these methods could be supported by interactive theorem proving software. Automatic methods based on state-space searching and based on decision procedures for automata on infinite paths (see, e.g., [16]), should also be extendible; however, these methods will apply only to special cases of the general model.

Likewise, key methods used in control theory for reasoning about continuous systems, such as stability analysis using Lyapunov functions (e.g., [79]) and robust control techniques (e.g., [23]), should be extendible to hybrid systems using HIOAs.

1.2 Evolution of the HIOA Framework:

The HIOA framework has evolved from two earlier input/output automaton models: the basic I/O automaton model of Lynch and Tuttle [55,56] and the timed I/O automaton model of Lynch, Vaandrager et al. [60,74]. Basic I/O automata consist essentially of states, start states, and discrete transitions. They have been used fairly extensively to describe and analyze asynchronous distributed algorithms—see, for example, [48].

Timed I/O automata add explicit *time-passage steps*, which allow time to pass in discrete jumps. In the simplest cases, time-passage steps involve just the passage of time, with no other changes to the state. However, in general, they are allowed to change the state in more elaborate ways, including changing variables that represent physical quantities. Timed I/O automata have been used mainly to describe timing-based distributed algorithms and communication protocols (e.g., [78,45,75,76,19,77,25]). Timed I/O automata have also been used in a few cases to model simple hybrid system "challenge problems", including the Generalized Railroad Crossing problem [30,31]. In these examples, the time-passage steps include changes to physical quantities such as train position and water level.

An early version of the HIOA modeling framework appeared in [53,54]. It augmented timed I/O automata by adding input and output variables and explicit *trajectories*; the trajectories describe the evolution of the state and external variables over intervals of time, rather than just their cumulative changes. This version of the HIOA framework was used to describe and analyze many hybrid systems examples, including automated transportation systems [61,49,83,81,82,50,42,44], intelligent vehicle highway systems [22,47], aircraft control systems [46,43], automotive control systems [24], and consumer electronics systems [11].

We summarize the results of these modeling efforts briefly. In these examples, HIOAs were used to model system components of many different kinds, including real-world components, computer programs, communication channels, sensors, actuators, and humans (for example, pilots interacting with aircraft control systems). Individual component automata were generally highly nondeterministic, and often allowed for bounded uncertainty in the values of quantities represented in the state. Component states often included timing information, for example, the current time and deadlines for the performance of certain actions. Composition was used to combine the component HIOAs into models of the complete systems. Levels of abstraction were used to describe several kinds of relationships between HIOAs, for example: the relationship between a detailed view of a system and a more abstract view; the relationship between a description of a system in terms of higher derivatives (e.g., acceleration) and a description in terms of lower derivatives (e.g., velocity or position); and the relationship between a version of a system that includes periodic sampling and correction and a version in which adjustment is continuous, but within an envelope of uncertainty.

The examples were analyzed using a variety of methods including invariant assertions, simulation relations, compositional reasoning, differential equations and integration. Many of the invariants and simulation relations involved timing data and data representing real-world quantities. Invariants and simulation relations were proved using inductive arguments on the length of executions, as is usual in the purely discrete setting. However, unlike in the discrete setting, the proofs in the hybrid setting included two different kinds of inductive steps: for discrete steps and trajectories. Arguments about discrete steps involved the sort of algebraic deduction that is typical in the discrete setting, whereas arguments about trajectories involved manipulation of differential equations and integrals. For example, a technique involving "positive invariant sets", derived from control theory, was used in [15] for showing that certain properties of the state are preserved during trajectories.

In general, the formal HIOA framework proved to be adequate for these examples. However, it was not ideal, because it introduced some complications that proved to be distracting. The main source of complication seemed to be the fact that the model has two mechanisms for modeling discrete communication: *shared actions* and *shared variables*. Also, it uses the same mechanism—shared variables—to model both discrete and continuous interaction between components. This intertwining of mechanisms led to some technicalities, for example, each automaton had to include a special *environment action* e, which is associated with discrete changes to input variables. To simplify matters, we were led to develop the new version of the HIOA model presented in this paper. The new version has a clearer separation between the mechanisms used to model discrete and continuous activity, and has only one mechanism for discrete communication: shared actions.

In the literature on discrete state machine models, both shared actions and shared variables are popular mechanisms for modeling interactions between system components. The shared action approach is used, for example, in the extensive research literature on process algebras (e.g., [35,66,67]), and in the work on I/O automata (e.g., [55,49]). The shared variable approach is used, for example, in the temporal logic and model-checking communities (e.g., [64,40,7]). The expressive power of shared action and shared variable communication is similar, and translations between special cases of these two types of models have been developed [39,18]. Choosing between these two forms of communication seems to be generally a matter of custom and convenience. One advantage of the shared-action approach is that it leads to simple mathematical notions of external behavior of state machines, based on sequences of actions (which are usually called "traces").

The new HIOA framework presented in this paper uses (only) shared actions for discrete communication, and uses shared variables for continuous communication. Discrete events are not allowed to make changes to shared variables, and the special environment action e is eliminated. Because the new model maintains a clearer separation between mechanisms for describing discrete and continuous activity, it is simpler overall—in its definitions, result statements, and proofs—than the earlier HIOA model of [53,54].

Another simplification in the new framework appears in the definitions and results involving receptiveness. In the original HIOA model of [53,54], and in other work that dealt with receptiveness [21,1,74] for discrete systems, receptiveness was defined in terms of two-player games between the system and its environment. In such a game, the goal of the system is to construct an infinite, non-Zeno execution, and the goal of the environment is to prevent this from happening. The simplification in this material in the new model is a result of our modeling of the game itself as an HIOA.

1.3 Other Related Work

Besides the models already discussed above, other precursors to the new HIOA model include the phase transition system models of [63,3,38] and Branicky's hybrid control systems [13,14]. Phase transition systems are similar to HIOAs in their combined treatment of discrete and continuous activity, for example, they have notions similar to our trajectories and hybrid sequences. However, work on phase transition system models does not address system decomposition issues such as external behavior, implementation relationships, and composition, which are emphasized in our paper. Branicky's hybrid control systems are also similar to ours in their modeling of discrete and continuous activity. This work has a control theory flavor, focusing on standard configurations including plant, controller, sensor and actuator, and focusing on stability results. Again, system decomposition issues are not addressed.

System decomposition issues, including levels of abstraction, compositionality, and receptiveness have been addressed by Alur and Henzinger [8] in their work on hybrid reactive modules. A major difference between this work and ours is that reactive modules communicate via shared variables and not via shared actions. Another difference is that hybrid reactive modules include an additional layer of structure tailored to modeling synchronous systems—structure that is not present in the HIOA model. In [8], a definition of receptiveness based on two-player games, similar to the definition in [53,54], is proposed, and is shown to be preserved by parallel composition. However, in [8], no circular dependencies ("feedback loops") are allowed among the continuous variables of different components, a restriction that greatly simplifies the analysis.

In [6,33], compositional trace-based semantics are presented for Statecharts-like languages that support hierarchical design of hybrid systems. These languages, called Charon and Masaccio, respectively, allow one to describe hierarchical state machines that communicate with their environment using shared variables. Communication via shared actions is not supported. Besides parallel composition and variable hiding, the languages also contain other operations required for the construction of hierarchical state machines, such as variable renaming and serial composition. The trace semantics presented in [6,33] for Charon and Masaccio is more concrete than the one that we present here: discrete events that do not change the observable part of the state are not eliminated from traces. As a consequence, a system that just lets time pass and performs a discrete "tick" step once every time unit is not an implementation of the same system without any discrete steps. The two systems are equivalent according to the trace semantics of this paper. We believe that our semantics are more intuitively appealing; the price we pay is that the proofs of our compositionality results are more complicated. [33] also contains some interesting proof rules for assume-guarantee reasoning. In [6,33], Zeno behavior and the issue of receptiveness are not considered.

1.4 Paper Organization

The rest of this paper is organized as follows. Section 2 contains mathematical preliminaries. Next, Section 3 defines notions that are useful for describing the behavior of hybrid systems, most importantly, trajectories and hybrid sequences. Section 4 defines *Hybrid Automata (HAs)*, which contain all of the structure of HIOAs except for the classification of external actions and variables as inputs or outputs. It also defines external behavior for HAs and implementation and simulation relationships between HAs. Section 5 presents composition and hiding operations for HAs. Section 6 defines *Hybrid I/O Automata (HIOAs)* by adding an input/output classification to HAs, and extends the theory of HAs to HIOAs. It also introduces a "strong compatibility" condition that ensures that HIOAs are composable, and describes situations in which strong compatibility is guaranteed to hold. Section 7 presents the theory of receptiveness, including a main theorem stating that receptiveness is preserved by composition (assuming strong compatibility). Finally, Section 8 presents some conclusions. Examples derived from earlier work on hybrid system modeling are included throughout. Appendix A lists some notational conventions used in the paper.

2 Mathematical Preliminaries

In this section, we give basic mathematical definitions that will be used as a foundation for our definitions of hybrid automata and hybrid I/O automata. These definitions involve functions, sequences, partial orders, and time. The automata definitions appear later, in Sections 4 and 6. Since most of the definitions here are reasonably standard, we encourage the reader to skip ahead to Section 3 and return to this section as needed.

2.1 Functions

If f is a function, then we denote the domain and range of f by $dom(f)$ and $range(f)$, respectively. If also S is a set, then we write $f \lceil S$ for the restriction of f to S, that is, the function g with $dom(g) = dom(f) \cap S$ such that $g(c) = f(c)$ 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 unique function h with $dom(h) = dom(f) \cup dom(g)$ satisfying the condition: for each $c \in dom(h)$, if $c \in dom(f)$ then $h(c) = f(c)$ and if $c \in dom(g)$ then $h(c) = g(c)$. More generally, if F is a set of pairwise compatible functions then we write $\bigcup F$ for the unique function h with $dom(h) = \bigcup\{dom(f) \mid f \in F\}$ satisfying the condition: for each $f \in F$ and $c \in dom(f)$, $h(c) = f(c)$.

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 function g with $dom(g) = dom(f)$ such that $g(c) = f(c) \lceil S$ for each $c \in dom(g)$. The restriction operation $\downarrow$ is extended to sets of functions by pointwise extension. Also, if f is a function whose range is a set of functions, all of which have a particular element d in their domain, then we write $f \downarrow d$ for the function g with $dom(g) = dom(f)$ such that $g(c) = f(c)(d)$ for each $c \in dom(g)$.

We say that two functions f and g whose ranges are sets of functions are *pointwise compatible* if for each $c \in dom(f) \cap dom(g)$, $f(c)$ and $g(c)$ are compatible. If f and g have the same domain and are pointwise compatible, then we denote by $f \dot{\cup} g$ the function h with $dom(h) = dom(f)$ such that $h(c) = f(c) \cup g(c)$ for each $c \in dom(h)$.

2.2 Sequences

Let S be any set. A *sequence* over S is a function from a downward closed subset of the natural numbers to S. Thus, the domain of a sequence is either the set of all natural numbers, or is of the form $\{0, \ldots, k\}$, for some natural number k. In the first case we say that the sequence is infinite, and in the second case finite. The sets of finite and infinite sequences over S are denoted by S^* and S^ω, respectively. Concatenation of a finite sequence with a finite or infinite sequence is denoted by juxtaposition. We use λ to denote the empty sequence, that is, the sequence with the empty domain. The sequence containing one element $c \in S$ is abbreviated as c. We say that a sequence σ is a *prefix* of a sequence ρ, denoted by $\sigma \leq \rho$, if $\sigma = \rho \lceil dom(\sigma)$. Thus, $\sigma \leq \rho$ if either $\sigma = \rho$, or σ is finite and $\rho = \sigma\sigma'$ for some sequence σ'. If σ is a nonempty sequence then $head(\sigma)$ denotes the first element of σ and $tail(\sigma)$ denotes σ with its first element removed. Moreover, if σ is finite, then $last(\sigma)$ denotes the last element of σ and $init(\sigma)$ denotes σ with its last element removed.

2.3 Partial Orders

We recall some basic definitions and results regarding partial orders (posets), and in particular, complete partial orders (cpos) from [29,32]. A *partial order (poset)* is a set S together with a binary relation $\sqsubseteq$ that is reflexive, antisymmetric, and transitive. In the sequel, we usually denote posets by the set S without explicit mention to the binary relation $\sqsubseteq$.

A subset $P \subseteq S$ is *bounded (above)* if there is a $c \in S$ such that $d \sqsubseteq c$ for each $d \in P$; in this case, c is an *upper bound* for P. A *least upper bound (lub)* for a subset $P \subseteq S$ is an upper bound c for P such that $c \sqsubseteq e$ for every upper bound e for P. If P has a lub, then it is necessarily unique, and we denote it by $\bigsqcup P$. A subset $P \subseteq S$ is *directed* if every finite subset Q of P has an upper bound in P. A poset S is *complete*, and hence is a *complete partial order (cpo)* if every directed subset P of S has a lub in S.

We say that $P' \subseteq S$ *dominates* $P \subseteq S$, denoted by $P \sqsubseteq P'$, if for every $c \in P$ there is some $c' \in P'$ such that $c \sqsubseteq c'$. We use the following two simple lemmas, adapted from [32] [Lemmas 3.1.1 and 3.1.2].

Lemma 2.1 *If P, P' are directed subsets of a cpo S and $P \sqsubseteq P'$ then $\bigsqcup P \sqsubseteq \bigsqcup P'$.*

Lemma 2.2 *Let $P = \{c_{ij} \mid i \in I, j \in J\}$ be a doubly indexed subset of a cpo S. Let P_i denote the set $\{c_{ij} \mid j \in J\}$ for each $i \in I$. Suppose*

(1) P is directed,
(2) each P_i is directed with lub c_i, and
(3) the set $\{c_i \mid i \in I\}$ is directed.

Then $\sqcup P = \sqcup\{c_i \mid i \in I\}$.

A finite or infinite sequence of elements, $c_0, c_1, c_2, \ldots$, of a poset S is called a *chain* if $c_i \sqsubseteq c_{i+1}$ for each non-final index i. We define the *limit* of the chain, $\lim_{i\to\infty} c_i$, to be the lub of the set $\{c_0, c_1, c_2, \ldots\}$ if S contains such a bound; otherwise, the limit is undefined. Since a chain is a special case of a directed set, each chain of a cpo has a limit.

A function $f : S \to S'$ between posets S and S' is *monotone* if $f(c) \sqsubseteq f(d)$ whenever $c \sqsubseteq d$. If f is monotone and P is a directed set, then the set $f(P) = \{f(c) \mid c \in P\}$ is directed as well. If f is monotone and $f(\bigsqcup P) = \bigsqcup f(P)$ for every directed set P, then f is said to be *continuous*.

An element c of a cpo S is *compact* if, for every directed set P such that $c \sqsubseteq \bigsqcup P$, there is some $d \in P$ such that $c \sqsubseteq d$. We define $\mathsf{K}(S)$ to be the set of compact elements of S. A cpo S is *algebraic* if every $c \in S$ is the lub of the set $\{d \in \mathsf{K}(S) \mid d \sqsubseteq c\}$. A simple example of an algebraic cpo is the set of finite or infinite sequences over some given domain, equipped with the prefix ordering. Here the compact elements are the finite sequences.

2.4 Time

Throughout this paper, we fix a *time axis* T, which is a subgroup of $(\mathsf{R}, +)$, the real numbers with addition. We assume that every infinite, monotone, bounded sequence of elements of T has a limit in T. The reader may find it convenient to think of T as the set R of real numbers, but the set Z of integers and the singleton set $\{0\}$ are also examples of allowed time axes. We define $\mathsf{T}^{\geq 0} \triangleq \{t \in \mathsf{T} \mid t \geq 0\}$.

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 is *left-closed* (*right-closed*) if it has a minimum (resp., maximum) element, and *left-open* (*right-open*) otherwise. An interval is *closed* if it is both left-closed and right-closed, and *open* if it is both left-open and right-open. We write $\min(J)$ and $\max(J)$ for the minimum and maximum 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 Describing Hybrid Behavior

In this section, we give basic definitions that are useful for describing discrete and continuous behavior of a system or system component, including discrete and continuous changes to the system's state, and discrete and continuous flow of information into and out of the system. The key notions are *static* and *dynamic types* for variables, *trajectories*, and *hybrid sequences*.

3.1 Static and Dynamic Types

We assume a universal set V of *variables*. A variable represents either a location within the state of a system or a location where information flows from one system component to another. For each variable v, we assume both a *(static) type*, which gives the set of values it may take on, and a *dynamic type*, which gives the set of trajectories it may follow. Formally, for each variable v we assume the following:

- $type(v)$, the *(static) type* of v. This is a nonempty set of values.
- $dtype(v)$, the *dynamic type* of v. This is a set of functions from left-closed intervals of T to $type(v)$ that satisfies the following properties:

(1) *(Closure under time shift)*
For each $f \in dtype(v)$ and $t \in \mathsf{T}$, $f + t \in dtype(v)$.

(2) *(Closure under subinterval)*
For each $f \in dtype(v)$ and each left-closed interval $J \subseteq dom(f)$, $f \lceil J \in dtype(v)$.

(3) *(Closure under pasting)*
Let $f_0, f_1, f_2, \ldots$ be a sequence of functions in $dtype(v)$ such that, for each index i such that f_i is not the final function in the sequence, $dom(f_i)$ is right-closed and $\max(dom(f_i)) = \min(dom(f_{i+1}))$. Then the function f defined by $f(t) \triangleq f_i(t)$, where i is the smallest index such that $t \in dom(f_i)$, is in $dtype(v)$.

The pasting-closure property is useful for modeling "discontinuities" in the evolution of variables caused by discrete transitions. Dynamic types provide a convenient way of describing restrictions on system behavior over time intervals, for example, restrictions on the behavior of system input variables.

Example 3.1 (Discrete variables) Let v be any variable and let C be the set of constant functions from a left-closed interval to $type(v)$. Then C is closed under time shift and subinterval. If the dynamic type of v is obtained by closing C under the pasting operation, then v is called a *discrete* variable. This is essentially the same as the definition of a discrete variable in [63]. ■

Example 3.2 (Standard real-valued function classes) If we take $\mathsf{T} = \mathsf{R}$ and $type(v) = \mathsf{R}$, then other examples of dynamic types can be obtained by taking the pasting closure of standard function classes from real analysis, such as the set of continuous functions, the set of differentiable functions, the set of functions that are differentiable k times (for any k), the set of smooth functions, the set of integrable functions, the set of L^p functions (for any p), the set of measurable locally essentially bounded functions [79], or the set of all functions. ■

Standard function classes are closed under time shift and subinterval, but not under pasting. A natural way of defining a dynamic type is as the pasting closure of a class of functions that is closed under time shift and subinterval. In such a case, it follows that the new class is closed under all three operations.

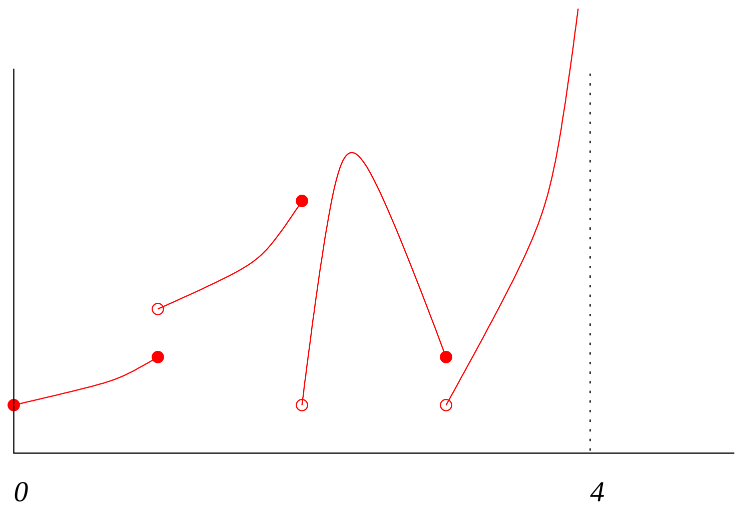

Fig. 1. Example of a function in a dynamic type based on continuous functions.

Example 3.3 (Pasting closure of the continuous functions) Figure 1 shows an example of an element f in a dynamic type based on (more precisely, equal to the pasting closure of) a subclass of the continuous functions. Function f is defined on the interval $[0, 4)$ and is obtained by pasting together four pieces. At the boundary points between these pieces, f takes the value specified by the leftmost piece, which makes f continuous from the left. Note that f is undefined at time 4. ■

In practice, most interesting dynamic types are pasting closures of subclasses of the continuous functions. Note that functions in such dynamic types are continuous from the left. Elsewhere in the literature on hybrid systems (e.g., [37]), functions that are continuous from the right are considered. To some extent, the choice of how to define function values at discontinuities is arbitrary. An advantage of our choice is a nice correspondence between concatenation and prefix ordering of trajectories and hybrid sequences (see Lemmas 3.5 and

3.7).

In this paper, we will occasionally be slightly sloppy and say that the dynamic type of a variable v is the function class F, even though F in not closed under the three required operations. In such a case, we mean that the dynamic type of v is the function class that results from closing F under the three operations.

3.2 Trajectories

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

3.2.1 Basic Definitions

Let $V \subseteq \mathsf{V}$ be a set of variables. A *valuation* $\mathbf{v}$ for V is a function that associates with each variable $v \in V$ a value in *type*(v). We write *val*(V) for the set of valuations for V. Let J be a left-closed interval of T with left endpoint equal to 0. Then a *J-trajectory* for V is a function $\tau : J \rightarrow val(V)$, such that for each $v \in V$, $\tau \downarrow v \in dtype(v)$. A *trajectory* for V is a J-trajectory for V, for any J. We write *trajs*(V) for the set of all trajectories for V.

A trajectory for V with domain $[0, 0]$ is called a *point* trajectory for V. If $\mathbf{v}$ is a valuation for V then $\wp(\mathbf{v})$ denotes the point trajectory for V that maps 0 to $\mathbf{v}$. We say that a J-trajectory is *finite* if J is a finite interval, *closed* if J is a (finite) closed interval, *open* if J is a right-open interval, and *full* if $J = \mathsf{T}^{\geq 0}$.

If τ is a trajectory then $\tau.ltime$, the *limit time* of τ, is the supremum of $dom(\tau)$. Also, we define $\tau.fval$, the *first valuation* of τ, to be $\tau(0)$, and if τ is closed, we define $\tau.lval$, the *last valuation* of τ, to be $\tau(\tau.ltime)$. For τ a trajectory and $t \in \mathsf{T}^{\geq 0}$, we define

$$
\begin{aligned}
&\tau \trianglelefteq t \triangleq \tau \lceil [0, t], \\
&\tau \triangleleft t \triangleq \tau \lceil [0, t), \\
&\tau \trianglerighteq t \triangleq (\tau \lceil [t, \infty)) - t.
\end{aligned}
$$

Note that, since dynamic types are closed under time shift and subintervals, the result of applying the above operations is always a trajectory, except when the result is a function with an empty domain. By convention, we also write $\tau \trianglelefteq \infty \triangleq \tau$ and $\tau \triangleleft \infty \triangleq \tau$.

3.2.2 Prefix Ordering

Trajectory τ is a *prefix* of trajectory τ', denoted by $\tau \leq \tau'$, if τ can be obtained by restricting τ' to a subset of its domain. Formally, if τ and τ' are trajectories for V, then $\tau \leq \tau'$ iff $\tau = \tau' \lceil dom(\tau)$. Alternatively, $\tau \leq \tau'$ iff there exists a $t \in \mathsf{T}^{\geq 0} \cup \{\infty\}$ such that $\tau = \tau' \trianglelefteq t$ or $\tau = \tau' \triangleleft t$. If $\tau \leq \tau'$ then clearly $dom(\tau) \subseteq dom(\tau')$. If T is a set of trajectories for V, then $pref(T)$ denotes the *prefix closure* of T, defined by:

$$pref(T) \triangleq \{\tau \in trajs(V) \mid \exists \tau' \in T : \tau \leq \tau'\}.$$

We say that T is *prefix closed* if $T = pref(T)$.

The following lemma gives a simple domain-theoretic characterization of the set of trajectories over a given set V of variables:

Lemma 3.4 *Let V be a set of variables. The set $trajs(V)$ of trajectories for V, together with the prefix ordering $\leq$, is an algebraic cpo. Its compact elements are the closed trajectories.*

Proof: It is trivial to check that $(trajs(V), \leq)$ is a partial order. In order to prove that it is a cpo, assume that T is a directed subset of $trajs(V)$. We prove that T has a least upper bound. It is routine to check that a set of trajectories is directed iff it is totally ordered by prefix. So T is totally ordered. Using this, it follows that the trajectories in T are pairwise compatible functions. Therefore, function $\bigcup T$ is defined.

We now prove that $\bigcup T$ is a trajectory for V. If $\bigcup T \in T$ then this is immediate. Otherwise, let $t \in \mathsf{T} \cup \{\infty\}$ be the supremum of the limit times of all trajectories in T. There exists an infinite ascending chain $t_0, t_1, t_2, \ldots$ of limit times of trajectories in T such that $t = \lim_{i \to \infty} t_i$ and all the t_i's are different. For each i, let τ_i be a trajectory in T with $t_i = \tau_i.ltime$. Next define, for each i, $\tau'_i = \tau_{i+1} \trianglelefteq t_i$. Then, by construction, the trajectories $\tau'_0, \tau'_1, \tau'_2, \ldots$ are closed and pairwise compatible, and $\bigcup_i \tau'_i = \bigcup T$. Let $\tau''_0, \tau''_1, \tau''_2, \ldots$ be the sequence of functions defined by

$$\begin{aligned}
&\tau''_0 \triangleq \tau'_0, \\
&\tau''_i \triangleq \tau'_i \lceil [\tau'_{i-1}.ltime, \infty) \quad \text{if } i > 0.
\end{aligned}$$

By construction, the τ''_i's are closed, pairwise compatible, and $\bigcup_i \tau''_i = \bigcup_i \tau'_i$. Using the assumption that dynamic types are closed under pasting, it follows that $\bigcup_i \tau''_i$ (and hence $\bigcup T$) is a trajectory.

Now we show that $\bigcup T$ is a lub for T. It follows immediately from the construction of $\bigcup T$ that $\bigcup T$ is an upper bound for T. Suppose that τ' is also an upper bound for T. We prove that $\bigcup T \leq \tau'$. Since each $\tau \in T$ satisfies $dom(\tau) \subseteq dom(\tau')$, also $\bigcup_{\tau \in T} dom(\tau) \subseteq dom(\tau')$. By definition of $\bigcup T$, $dom(\bigcup T) = \bigcup_{\tau \in T} dom(\tau)$. Hence $dom(\bigcup T) \subseteq dom(\tau')$. Let t be an element of $dom(\bigcup T)$. Then t is in the domain of some $\tau \in T$. Since τ is a prefix of both $\bigcup T$ and τ', $(\bigcup T)(t) = \tau'(t)$. Thus, $\tau' \lceil dom(\bigcup T) = \bigcup T$, that is, $\bigcup T \leq \tau'$. It follows that $trajs(V)$ is a cpo.

We leave it to the reader to check that the closed trajectories are the compact elements in this cpo, and that the cpo is algebraic. ■

3.2.3 Concatenation

The concatenation of two trajectories is obtained by taking the union of the first trajectory and the function obtained by shifting the domain of the second trajectory until the start time agrees with the limit time of the first trajectory; the last valuation of the first trajectory, which may not be the same as the first valuation of the second trajectory, is the one that appears in the concatenation. Formally, suppose τ and τ' are trajectories for V, with τ closed. Then the *concatenation* $\tau \frown \tau'$ is the function given by

$$\tau \frown \tau' \triangleq \tau \cup (\tau' \lceil (0, \infty) + \tau.ltime).$$

Because dynamic types are closed under time shift and pasting, it follows that $\tau \frown \tau'$ is a trajectory for V. Observe that $\tau \frown \tau'$ is finite (resp., closed, full) if and only if τ' is finite (resp., closed, full). Observe also that concatenation is associative.

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

Lemma 3.5 *Let τ and υ be trajectories for V with τ closed. Then*

$$\tau \leq \upsilon \Leftrightarrow \exists \tau' : \upsilon = \tau \frown \tau'.$$

Note that if $\tau \leq \upsilon$, then the trajectory τ' such that $\upsilon = \tau \frown \tau'$ is unique except that it has an arbitrary value for $\tau'.fval$. Note also that the "$\Leftarrow$" implication in Lemma 3.5 would not hold if the first valuation of the second argument, rather than the last valuation of the first argument, were used in the concatenation.

We extend the definition of concatenation to any (finite or countably infinite) number of arguments. Let $\tau_0, \tau_1, \tau_2, \ldots$ be a (finite or infinite) sequence of trajectories such that τ_i is closed for each nonfinal index i. Define trajectories $\tau_0', \tau_1', \tau_2', \ldots$ inductively by

$$\tau'_0 \triangleq \tau_0,$$
$$\tau'_{i+1} \triangleq \tau'_i \frown \tau_{i+1} \text{ for nonfinal } i.$$

Lemma 3.5 implies that for each nonfinal i, $\tau'_i \leq \tau'_{i+1}$. We define the *concatenation* $\tau_0 \frown \tau_1 \frown \tau_2 \cdots$ to be the limit of the chain $\tau'_0, \tau'_1, \tau'_2, \ldots$; existence of this limit follows from Lemma 3.4.

3.3 Hybrid Sequences

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

3.3.1 Basic Definitions

Fix a set A of actions and a set V of variables. An (A, V)-*sequence* is a finite or infinite alternating sequence $\alpha = \tau_0\, a_1\, \tau_1\, a_2\, \tau_2 \ldots$, where

(1) each τ_i is a trajectory in $trajs(V)$,
(2) each a_i is an action in A,
(3) if α is a finite sequence then it ends with a trajectory, and
(4) if τ_i is not the last trajectory in α then $dom(\tau_i)$ is closed.

A *hybrid sequence* is an (A, V)-sequence for some A and V.

Since the trajectories in a hybrid sequence can be point trajectories, our notion of hybrid sequence allows a sequence of discrete actions to occur at the same real time, with corresponding changes of variable values. An alternative approach is described in [69], where state changes at a single real time are modeled using a notion of "superdense time". Specifically, hybrid behavior is modeled in [69] using functions from an extended time domain, which includes countably many elements for each real time, to states.

If α is a hybrid sequence, with notation as above, then we define the *limit time* of α, $\alpha.ltime$, to be $\sum_i \tau_i.ltime$. A hybrid sequence α is defined to be:

- *time-bounded* if $\alpha.ltime$ is finite.
- *admissible* if $\alpha.ltime = \infty$.

- *closed* if α is a finite sequence and the domain of its final trajectory is a closed interval.
- *Zeno* if α is neither closed nor admissible, that is, if α is time-bounded and is either an infinite sequence, or else a finite sequence ending with a trajectory whose domain is right-open.

A more standard definition of "Zeno" would be simply "a time-bounded infinite sequence". We add the second option to the definition in order to guarantee a simple property of the hiding/restriction operator, see Lemma 4.9(2). Except for Lemma 4.9(2), all results of this paper hold also for the more standard definition. We say that a hybrid sequence is "non-Zeno" if it is not Zeno, that is, if it is closed or admissible."

For any hybrid sequence α, we define the *first valuation* of α, $\alpha.\mathit{fval}$, to be $\tau_0.\mathit{fval}$. Also, if α is closed, we define the *last valuation* of α, $\alpha.\mathit{lval}$, to be $\mathit{last}(\alpha).\mathit{lval}$, that is, the last valuation in the final trajectory of α.

3.3.2 Prefix Ordering

We say that (A, V)-sequence $\alpha = \tau_0\, a_1\, \tau_1 \ldots$ is a *prefix* of (A, V)-sequence $\beta = \upsilon_0\, b_1\, \upsilon_1 \ldots$, denoted by $\alpha \leq \beta$, provided that (at least) one of the following holds:

(1) $\alpha = \beta$.
(2) α is a finite sequence ending in some τ_k; $\tau_i = \upsilon_i$ and $a_{i+1} = b_{i+1}$ for every i, $0 \leq i < k$; and $\tau_k \leq \upsilon_k$.

Like the set of trajectories over V, the set of (A, V)-sequences is a cpo:

Lemma 3.6 *Let V be a set of variables and A a set of actions. The set of (A, V)-sequences, together with the prefix ordering $\leq$, is an algebraic cpo. Its compact elements are the closed (A, V)-sequences.*

Proof: We leave to the reader the routine check that $\leq$ is a partial order. Note that this uses the fact that $\leq$ is a partial order on trajectories (Lemma 3.4).

In order to prove that we have a cpo, let S be a directed subset of (A, V)-sequences. We prove that S has a least upper bound. It is easy to check that S is totally ordered by the prefix ordering $\leq$. We distinguish two cases.

(1) There is no finite upper bound on the number of trajectories that occur in the sequences in S. In this case, we can construct an infinite sequence $\alpha_0, \alpha_1, \alpha_2 \ldots$ of elements of S such that, for each i, α_i contains at least i actions and $i+1$ trajectories, and $\alpha_i \leq \alpha_{i+1}$. For each $i \in \mathsf{N}$, let τ_i be the $i+1$-st trajectory (the one indexed by i) in α_{i+1}, and for $i \geq 1$, let a_i be

the i-th action in α_i. Let $\alpha = \tau_0\, a_1\, \tau_1\, a_2\, \tau_2 \ldots$. It is easy to verify that α is an upper bound of the set $\{\alpha_i \mid i \in \mathsf{N}\}$ and in fact, is the only upper bound of this set. It follows that α is the lub of S, as needed.

(2) There is a finite upper bound k on the number of trajectories that occur in the (A, V)-sequences in S. In this case, let S' be the set obtained by removing all sequences with fewer than k trajectories from S. Since S' is totally ordered, $\mathit{init}(\alpha) = \mathit{init}(\alpha')$ for any $\alpha, \alpha' \in S'$. (Recall that *init* is an ordinary sequence operation—it yields all but the last element of the sequence.) Choose any $\alpha \in S'$ and let $\sigma = \mathit{init}(\alpha)$. Let T be the set of final trajectories of sequences in S'. Again using the fact that S' is totally ordered, we obtain that T is totally ordered by the prefix ordering on trajectories. Let τ be the least upper bound of T (this upper bound exists by Lemma 3.4). It is routine to check that $\sigma\, \tau$ is a least upper bound of S', and thus of S.

We leave it to the reader to check that the closed (A, V)-sequences are the compact elements in this cpo, and that the cpo is algebraic. ■

3.3.3 Concatenation

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

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

(Here, *init*, *last*, *head* and *tail* are ordinary sequence operations.)

Lemma 3.7 *Let α and β be (A, V)-sequences with α closed. Then*

$$\alpha \leq \beta \Leftrightarrow \exists \alpha' : \beta = \alpha \frown \alpha'.$$

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

As we did for trajectories, we extend the concatenation definition for (A, V)-sequences to any finite or infinite number of arguments. Let $\alpha_0, \alpha_1, \ldots$ be a finite or infinite sequence of (A, V)-sequences such that α_i is closed for each nonfinal index i. Define (A, V)-sequences $\alpha'_0, \alpha'_1, \ldots$ inductively by

$$\alpha'_0 \triangleq \alpha_0,$$
$$\alpha'_{i+1} \triangleq \alpha'_i \frown \alpha_{i+1} \text{ for nonfinal } i.$$

Lemma 3.7 implies that for each nonfinal i, $\alpha'_i \leq \alpha'_{i+1}$. We define the *concatenation* $\alpha_0 \frown \alpha_1 \cdots$ to be the limit of the chain $\alpha'_0, \alpha'_1, \ldots$; existence of this limit is ensured by Lemma 3.6.

3.3.4 Restriction

Let A and A' be sets of actions and let V and V' be sets of variables. The (A', V')-*restriction* of an (A, V)-sequence α, denoted by $\alpha \lceil (A', V')$, is obtained by first projecting all trajectories of α on the variables in V', then removing the actions not in A', and finally concatenating all adjacent trajectories. Formally, we define the (A', V')-restriction first for closed (A, V)-sequences and then extend the definition to arbitrary (A, V)-sequences using a limit construction. The definition for closed (A, V)-sequences is by induction on the length of those sequences:

$$\tau \lceil (A', V') = \tau \downarrow V' \text{ if } \tau \text{ is a single trajectory,}$$

$$\alpha\, a\, \tau \lceil (A', V') = \begin{cases} (\alpha \lceil (A', V'))\; a\; (\tau \downarrow V') & \text{if } a \in A', \\ (\alpha \lceil (A', V')) \frown (\tau \downarrow V') & \text{otherwise.} \end{cases}$$

Note that in the case where, due to removal of some action, we concatenate two adjacent trajectories, we lose the first state of the second trajectory (by letting the last state of the first trajectory dominate). It is easy to see that the restriction operator is monotone on the set of closed (A, V)-sequences. Hence, if we apply this operation to a directed set, the result is again a directed set. Together with Lemma 3.6, this allows us to extend the definition of restriction to arbitrary (A, V)-sequences by:

$$\alpha \lceil (A', V') = \sqcup \{\beta \lceil (A', V') \mid \beta \text{ is a closed prefix of } \alpha\}.$$

Lemma 3.8 *(A', V')-restriction is a continuous operation.*

Proof: This follows by general domain-theoretic arguments. For convenience, in this proof we write $f(\alpha)$ as an abbreviation for $\alpha \lceil (A', V')$.

First we establish that (A', V')-restriction is monotone for arbitrary (A, V)-sequences. Let α, α' be (A, V)-sequences with $\alpha \leq \alpha'$; we show that $f(\alpha) \leq f(\alpha')$. Let P and P' denote the set of closed prefixes of α and α', respectively. By transitivity of the prefix ordering, it follows that P' dominates P, that is, $P \sqsubseteq P'$. Since the restriction operation is monotone on closed (A, V)-sequences, it follows that $f(P) \sqsubseteq f(P')$. Then Lemma 2.1 implies that $\sqcup f(P) \leq \sqcup f(P')$. By the definition of the restriction operation, this implies that $f(\alpha) \leq f(\alpha')$, which shows monotonicity.

Now we complete the proof that (A, V)-restriction is continuous by assuming that P is any directed set of (A, V)-sequences and showing that $f(\sqcup P) = \sqcup f(P)$. By the definition of the restriction operation, $f(\sqcup P) = \sqcup\{f(\beta) \mid \beta \text{ is a closed prefix of } \sqcup P\}$. By Lemma 3.6 and the definition of compact elements, any closed prefix β of $\sqcup P$ is also a prefix of some $\alpha \in P$. Therefore, $f(\sqcup P) = \sqcup\{f(\beta) \mid \beta \text{ is closed and } \exists\alpha \in P : \beta \text{ is a prefix of } \alpha\}$.

Now we apply Lemma 2.2 to the right hand side of this last equation. To do this, we must show:

(1) $Q \triangleq \{f(\beta) \mid \beta \text{ is closed and } \exists\alpha \in P : \beta \text{ is a prefix of } \alpha\}$ is a directed set. To see this, consider any nonempty finite subset $R \subseteq Q$. Each element of R is a prefix of some $\alpha \in P$. Therefore, since P is a directed set, there is some single $\alpha' \in P$ such that each element of R is a prefix of α'. Therefore, R is a directed set; since R is finite, it has a lub in R, and hence in Q, as needed.
(2) For each $\alpha \in P$, $\{f(\beta) \mid \beta \text{ is closed and } \beta \text{ is a prefix of } \alpha\}$ is a directed set with lub $f(\alpha)$. The first part follows because the set of closed prefixes of α is a directed set and f is monotone. The second part follows from the definition of restriction.
(3) The set $f(P)$ is directed. This follows because P is a directed set and f is monotone.

Then Lemma 2.2 implies that

$$\sqcup\{f(\beta) \mid \beta \text{ is closed and } \exists\alpha \in P : \beta \text{ is a prefix of } \alpha\} =$$

$$= \sqcup\{f(\alpha) \mid \alpha \in P\} = \sqcup f(P).$$

Thus, $f(\sqcup P) = \sqcup f(P)$, as needed. ■

The proofs of the following three lemmas are left to the reader.

Lemma 3.9 $(\alpha_0 \frown \alpha_1 \frown \cdots) \lceil (A, V) = \alpha_0 \lceil (A, V) \frown \alpha_1 \lceil (A, V) \frown \ldots$.

Lemma 3.10 $(\alpha \lceil (A, V)) \lceil (A', V') = \alpha \lceil (A \cap A', V \cap V')$.

Lemma 3.11

(1) α is time-bounded if and only if $\alpha \lceil (A, V)$ is time-bounded.
(2) α is admissible if and only if $\alpha \lceil (A, V)$ is admissible.
(3) If α is closed then $\alpha \lceil (A, V)$ is closed.
(4) If α is non-Zeno then $\alpha \lceil (A, V)$ is non-Zeno.

4 Hybrid Automata

In this section, as a preliminary step toward defining hybrid I/O automata, we define a slightly more general *hybrid automaton* model. In hybrid automata, actions and variables are classified as external or internal. External actions and variables are not further classified as input or output; the input/output distinction is added later, in Section 6. We define how hybrid automata execute and define implementation and simulation relations between hybrid automata.

4.1 Definition of Hybrid Automata

A hybrid automaton is a state machine whose states are valuations of *variables*, and that uses other variables for communication with its environment. It also has a set of *actions*, some of which may be internal and some external. The state of a hybrid automaton may change in two ways: by *discrete transitions*, which change the state atomically and instantaneously, and by *trajectories*, which describe the evolution of the state over intervals of time. The discrete transitions are labeled with actions; this will allow us to synchronize the transitions of different hybrid automata when we compose them in parallel. The evolution described by a trajectory may be described by continuous or discontinuous functions.

Definition 4.1 *A* hybrid automaton (HA) $\mathcal{A} = (W, X, Q, \Theta, E, H, D, \mathcal{T})$ *consists of:*

- *A set W of* external variables *and a set X of* internal variables*, disjoint from each other. We write $V \triangleq W \cup X$.*
- *A set $Q \subseteq val(X)$ of* states*.*
- *A nonempty set $\Theta \subseteq Q$ of* start states*.*
- *A set E of* external actions *and a set H of* internal actions*, disjoint from each other. We write $A \triangleq E \cup H$.*
- *A set $D \subseteq Q \times A \times Q$ of* discrete transitions*. We use $\mathbf{x} \xrightarrow{a}_{\mathcal{A}} \mathbf{x}'$ as shorthand for $(\mathbf{x}, a, \mathbf{x}') \in D$. We sometimes drop the subscript and write $\mathbf{x} \xrightarrow{a} \mathbf{x}'$, when we think $\mathcal{A}$ should be clear from the context. We say that a is* enabled *in $\mathbf{x}$ if there exists an $\mathbf{x}'$ such that $\mathbf{x} \xrightarrow{a} \mathbf{x}'$.*
- *A set $\mathcal{T}$ of trajectories for V such that $\tau(t) \lceil X \in Q$ for every $\tau \in \mathcal{T}$ and $t \in dom(\tau)$. Given a trajectory $\tau \in \mathcal{T}$ we denote $\tau.fval \lceil X$ by $\tau.fstate$ and, if τ is closed, we denote $\tau.lval \lceil X$ by $\tau.lstate$. We require that the following axioms hold:*
  T1 (Prefix closure)
  For every $\tau \in \mathcal{T}$ and every $\tau' \leq \tau$, $\tau' \in \mathcal{T}$.
  T2 (Suffix closure)

For every $\tau \in \mathcal{T}$ and every $t \in dom(\tau)$, $\tau \trianglerighteq t \in \mathcal{T}$.

T3 (Concatenation closure)

Let $\tau_0, \tau_1, \tau_2, \ldots$ be a sequence of trajectories in $\mathcal{T}$ such that, for each nonfinal index i, τ_i is closed and $\tau_i.lstate = \tau_{i+1}.fstate$. Then $\tau_0 \frown \tau_1 \frown \tau_2 \cdots \in \mathcal{T}$.

Axioms **T1-3** express some natural conditions on the set of trajectories that we need to construct our theory. A key part of this theory is a parallel composition operation for hybrid automata. In a composed system, any trajectory of any component automaton may be interrupted at any time by a discrete transition of another (possibly independent) component automaton. Axiom **T1** ensures that the part of the trajectory up to the discrete transition is a trajectory, and axiom **T2** ensures that the remainder is a trajectory. Axiom **T3** is required because the environment of a hybrid automaton, as a result of its own internal discrete transitions, may change its continuous dynamics repeatedly, and the automaton must be able to follow this behavior.

The earlier definition of hybrid automata in [53,54] used a special stuttering action e instead of axiom **T3**. Another key difference between the new definition of hybrid automaton and the earlier one is that in [53,54], the external variables were considered to be part of the state. This meant, for example, that discrete transitions could depend on the values of these variables, a situation that introduced technical complications. A local transition of one automaton could change an output variable, which could cause a discrete change in a second automaton, which in turn could change an input variable in the first automaton. To avoid cyclic constraints during the interaction of systems, we had to add several axioms, which complicated the use of our automaton definitions in applications.

In the new definition, we explicitly identify the set Q of states as a subset of $val(X)$. In the earlier definition of [53,54] any valuation in $val(X)$ was called a state. The reason for introducing Q is that in Section 6, we will require that in each state each input trajectory is accepted. In actual system descriptions, we often encounter valuations which are not reachable from the initial state, which in fact we do not want to view as states, and from which no behavior is enabled.[4] By excluding these "ghost" valuations from Q, we save ourselves the trouble of having to think about them.

Hybrid automata that have no external variables are very similar to the timed automata defined in [60,74]. The main difference is that hybrid automata have trajectories as a primitive rather than a derived notion. Also, the state of a timed automaton need not be organized using variables with particular types and dynamic types.

[4] Typical examples are the valuations that do not satisfy the "location invariants" of Alur-Dill style timed automata [2].

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

Notation: In examples we typically specify sets of trajectories using differential and algebraic equations and inclusions. Below we explain a few notational conventions that help us in doing this. Suppose the time domain T is R, τ is a (fixed) trajectory over some set of variables V, and $v \in V$. With some abuse of notation, we use the variable name v to denote the function $\tau \downarrow v$ in $dom(\tau) \rightarrow type(v)$, which gives the value of v at all times during trajectory τ. Similarly, we view any expression e containing variables from V as a function with domain $dom(\tau)$. Using these conventions we can say, for example, that τ satisfies the algebraic equation

$$v = e,$$

which means that, for every $t \in dom(\tau)$, $v(t) = e(t)$, that is, the constraint on the variables expressed by equation $v = e$ holds for each state on trajectory τ. Suppose that v is a variable and e is a real-valued expression containing variables from V. Suppose also that e, when viewed as a function, is integrable. Then we say that τ satisfies

$$\dot{v} = e$$

if, for every $t \in dom(\tau)$, $v(t) = v(0) + \int_0^t e(t')dt'$. Note that this interpretation of the differential equation makes sense even at points where v is not differentiable. A similar interpretation of differential equations is used by Polderman and Willems [71], who call these "weak solutions".

In the remainder of this subsection, we give two simple examples of hybrid automata.

Example 4.2 (Vehicle HA) We describe an HA *Vehicle*, displayed[5] in Figure 2, which models a vehicle that follows a suggested acceleration approximately, to within an error of $\epsilon \geq 0$. The time domain T is R. The state of the *Vehicle* automaton includes two real-valued internal variables *vel* and *acc*, which represent the actual velocity and acceleration of the vehicle, respectively. In addition, the automaton has two real-valued external variables, *vel-out* and *acc-in*, representing reported velocity and suggested acceleration.

[5] We use an arrow notation because later on in this paper, in Section 6, we will view *acc-in* as an input variable and *vel-out* as an output variable. Within the context of the present chapter the arrow notation has no meaning.

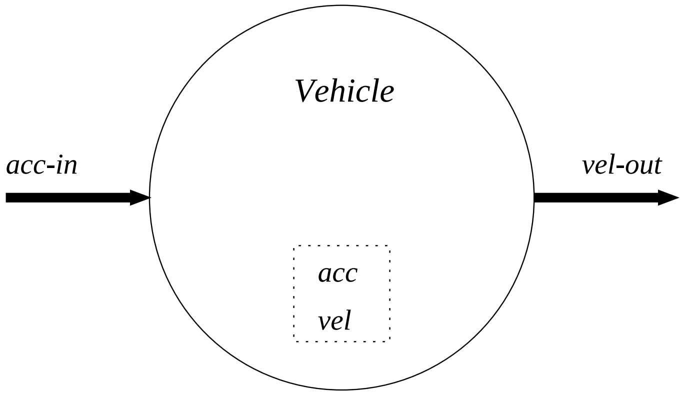

Fig. 2. The hybrid automaton *Vehicle*.

The dynamic type of the variables vel, $vel\text{-}out$, and $acc\text{-}in$ is the (pasting closure of the) set of continuous functions. The dynamic type of acc is the set of integrable functions.

Vehicle is defined to be the HA such that $W = \{acc\text{-}in, vel\text{-}out\}$, $X = \{vel, acc\}$, Q is the set of all valuations of the variables vel and acc, and Θ consists of the single valuation that assigns 0 to both state variables. The set of actions is empty, and (therefore) D, the set of discrete transitions, is empty. Set $\mathcal{T}$ consists of all trajectories that satisfy:

$$\dot{vel} = acc \tag{1}$$
$$acc(t) \in [acc\text{-}in(t) - \epsilon, acc\text{-}in(t) + \epsilon] \quad \text{for } t > 0 \tag{2}$$
$$vel\text{-}out = vel \tag{3}$$

Equation (1) says that the velocity is obtained by integrating the acceleration. Inclusion (2) asserts that, except possibly for the left endpoint, the actual acceleration is within ϵ of the suggested acceleration. Equation (3) says that the velocity is reported accurately. We leave the reader to show that the trajectory axioms **T1**–**T3** are satisfied; the form of the equations and inclusions used to define the trajectories should make this clear. We restrict to the case $t > 0$ in equation (2) because we do not want to constrain either the input or the starting state of trajectories. The reason for this restriction is technical (it ensures that *Vehicle* can be viewed as a proper HIOA that satisfies the input trajectory enabling property) and should become clearer in Section 6. ■

Example 4.3 (Controller HA) Now we describe an HA *Controller*, displayed in Figure 3, which models a controller that suggests accelerations for a vehicle, with the intention of ensuring that the vehicle's velocity does not exceed a pre-specified velocity vmax. The controller monitors the vehicle's velocity, and every time d, for some fixed $\mathsf{d} > 0$, it produces a new suggested acceleration to be followed for the next time d. The acceleration is chosen in

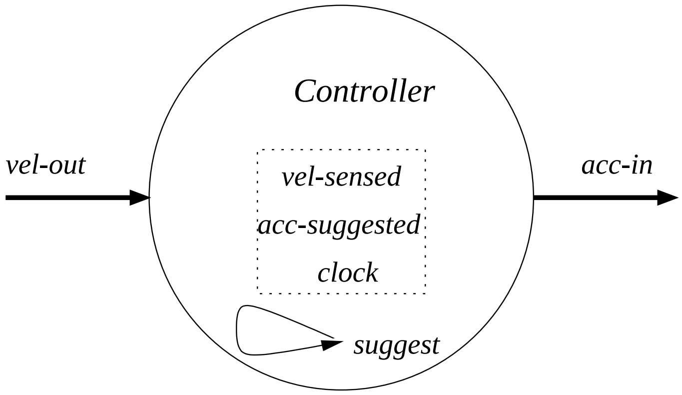

Fig. 3. The hybrid automaton *Controller*.

such a way that, if it is followed to within an error of ϵ, the velocity will remain below vmax (provided the vehicle is not going too fast in the first place). We assume that $\mathsf{vmax} \geq \epsilon\, \mathsf{d}$.

The components of the *Controller* HA are as follows: $W = \{\mathit{vel\text{-}out}, \mathit{acc\text{-}in}\}$ and $X = \{\mathit{vel\text{-}sensed}, \mathit{acc\text{-}suggested}, \mathit{clock}\}$. All variables are of type R. The dynamic types of *vel-out*, *vel-sensed*, *acc-in*, and *clock* are the (pasting closure of the) set of continuous functions, and *acc-suggested* is a discrete variable. Q is the set of valuations of X in which $\mathit{clock} \leq \mathsf{d}$. Θ consists of one valuation, which assigns 0 to all state variables. $E = \emptyset$ and H contains the single action *suggest*. Set D consists of the *suggest* steps specified by[6] :

$$\mathit{clock} = \mathsf{d} \tag{4}$$
$$\mathit{vel\text{-}sensed} + (\mathit{acc\text{-}suggested}' + \epsilon)\mathsf{d} \leq \mathsf{vmax} \tag{5}$$
$$\mathit{clock}' = 0 \tag{6}$$
$$\mathit{vel\text{-}sensed}' = \mathit{vel\text{-}sensed} \tag{7}$$

Equation (4) says that the clock indicates that it is time for the suggested acceleration to be computed. Inequality (5) says that the new suggested acceleration is chosen so that, if the vehicle follows it for the next time d, even with an error of ϵ, the velocity will still remain at most vmax. Equation (6) says that the clock is reset after the discrete transition. Equation (7) says that the transition does not change the value of *vel-sensed*. Set $\mathcal{T}$ consists of all trajectories that satisfy:

$$\dot{\mathit{acc\text{-}suggested}} = 0 \tag{8}$$
$$\dot{\mathit{clock}} = 1 \tag{9}$$
$$\mathit{vel\text{-}sensed}(t) = \mathit{vel\text{-}out}(t) \qquad \text{for } t > 0 \tag{10}$$

[6] Here we use the standard convention that v denotes the value of a variable in the start state of a discrete transition, and v' denotes the value in the end state.

$$acc\text{-}in = acc\text{-}suggested \tag{11}$$

Since *acc-suggested* is a discrete variable, the reader might think that adding constraint (8) makes no difference. However, if we expand this constraint using our definition of solutions for differential equations, we obtain

$$acc\text{-}suggested(t) = acc\text{-}suggested(0) + \int_0^t 0 \, dt' = acc\text{-}suggested(0),$$

which means that *acc-suggested* remains constant throughout the full trajectory. So the effect of adding differential equation (8) is that it rules out the jumps that are allowed by the dynamic type of *acc-suggested*. Equation (9) states that *clock* has rate 1, and is therefore a clock variable in the sense of the timed automaton model of [5].

Equation (10) says that the velocity sensed by the controller is the same as the velocity reported to the controller by its environment. Equation (11) asserts that the acceleration that the controller provides to its environment is the same as the acceleration that it has most recently computed. Again, we leave the reader to show that the trajectory axioms **T1**–**T3** are satisfied. ■

4.2 Executions and Traces

We now define execution fragments, executions, trace fragments, and traces, which are used to describe automaton behavior. An *execution fragment* of a hybrid automaton $\mathcal{A}$ is an (A, V)-sequence $\alpha = \tau_0 \, a_1 \, \tau_1 \, a_2 \, \tau_2 \ldots$, where (1) each τ_i is a trajectory in $\mathcal{T}$, and (2) if τ_i is not the last trajectory in α then $\tau_i.lstate \stackrel{a_{i+1}}{\to} \tau_{i+1}.fstate$. An execution fragment records what happens during a particular run of a system, including all the instantaneous, discrete state changes and all the changes to the state and external variables that occur while time advances. We write $frags_{\mathcal{A}}$ for the set of all execution fragments of $\mathcal{A}$.

If α is an execution fragment, with notation as above, then we define the *first state* of α, $\alpha.fstate$, to be $\tau_0.fstate$. We say that α is an execution fragment *from* a state $\mathbf{x}$ if $\alpha.fstate = \mathbf{x}$. An execution fragment α is defined to be an *execution* if $\alpha.fstate$ is a start state, that is, $\alpha.fstate \in \Theta$. We write $execs_{\mathcal{A}}$ for the set of all executions of $\mathcal{A}$. If α is a closed (A, V)-sequence then we define the *last state* of α, $\alpha.lstate$, to be $last(\alpha).lstate$. A state of $\mathcal{A}$ is *reachable* if it is the last state of some closed execution of $\mathcal{A}$.

Example 4.4 (Vehicle execution) Since the *Vehicle* HA of Example 4.2 has no discrete steps, each of its executions is a one-element sequence consisting of a single trajectory over all the variables of *Vehicle*. An example of such

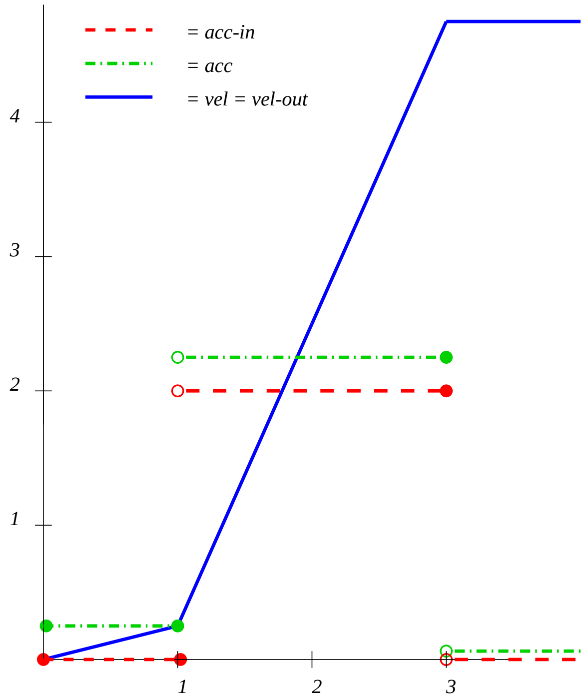

Fig. 4. An execution of the *Vehicle* (lower two lines after 3 are supposed to coincide).

an execution, depicted graphically in Figure 4, is the one consisting of the trajectory τ with $\tau.ltime = \infty$, and such that:

$$acc\text{-}in(t) = \begin{array}{cl} 0 & \text{if } t \leq 1, \\ 2 & \text{if } 1 < t \leq 3, \\ 0 & \text{if } t > 3. \end{array}$$

$$acc(t) = \begin{array}{cl} \epsilon & \text{if } t \leq 1, \\ 2+\epsilon & \text{if } 1 < t \leq 3, \\ 0 & \text{if } t > 3. \end{array}$$

$$vel(t) = vel\text{-}out(t) = \begin{array}{cl} \epsilon t & \text{if } t \leq 1, \\ (2+\epsilon)t - 2 & \text{if } 1 < t \leq 3, \\ 4+3\epsilon & \text{if } t > 3. \end{array}$$

Any finite prefix of τ would also yield an execution of *Vehicle*. The trace of τ is the one-element sequence obtained by projecting τ on $\{acc\text{-}in, vel\text{-}out\}$. ■

Example 4.5 (Controller execution) In the *Controller* HA of Example 4.3, suppose $\mathsf{d} = 1$, so the suggested acceleration is recalculated at times

1, 2, etc. Also suppose that $\mathsf{vmax} \geq 4 + 4\epsilon$. Then an example execution of *Controller* is the infinite sequence $\alpha = \tau_0\ \mathit{suggest}\ \tau_1\ \mathit{suggest}\ \tau_2 \ldots$, where, for every i and for every $t \in \mathit{dom}(\tau_i)$

(1) $\tau_i.\mathit{ltime} = 1$.
(2) $\tau_i(t)(\mathit{clock}) = t$.
(3) If $i = 0$ then $\tau_i(t)(v)$ is equal to 0 for $v \in \{\mathit{acc\text{-}suggested}, \mathit{acc\text{-}in}\}$ and ϵt for $v \in \{\mathit{vel\text{-}out}, \mathit{vel\text{-}sensed}\}$.
(4) If $1 \leq i \leq 2$ then $\tau_i(t)(v)$ is equal to 2 for $v \in \{\mathit{acc\text{-}suggested}, \mathit{acc\text{-}in}\}$ and $(2+\epsilon)(i+t) - 2$ for $v \in \{\mathit{vel\text{-}out}, \mathit{vel\text{-}sensed}\}$.
(5) If $i \geq 3$ then $\tau_i(t)(v)$ is equal to 0 for $v \in \{\mathit{acc\text{-}suggested}, \mathit{acc\text{-}in}\}$ and $4 + 3\epsilon$ for $v \in \{\mathit{vel\text{-}out}, \mathit{vel\text{-}sensed}\}$.

The assumed bound on vmax implies that the suggested accelerations in this execution are actually possible suggestions according to the rule given in the *Controller* automaton definition. The trace of execution α consists of a single trajectory because *Controller* has no external actions. This trajectory is defined by:

$$\begin{array}{rcl} \mathit{acc\text{-}in}(t) = & 0 & \text{if } t \leq 1, \\ & 2 & \text{if } 1 < t \leq 3, \\ & 0 & \text{if } t > 3. \end{array}$$

$$\begin{array}{rcl} \mathit{vel\text{-}out}(t) = & \epsilon t & \text{if } t \leq 1, \\ & (2+\epsilon)t - 2 & \text{if } 1 < t \leq 3, \\ & 4 + 3\epsilon & \text{if } t > 3. \end{array}$$

■

Like trajectories also execution fragments are closed under countable concatenation.

Lemma 4.6 *Let $\alpha_0, \alpha_1, \ldots$ be a finite or infinite sequence of execution fragments of $\mathcal{A}$ such that, for each nonfinal index i, α_i is closed and $\alpha_i.\mathit{lstate} = \alpha_{i+1}.\mathit{fstate}$. Then $\alpha_0 \frown \alpha_1 \frown \cdots$ is an execution fragment of $\mathcal{A}$.*

Proof: Follows easily from the definitions, using axiom **T3**. ■

Lemma 4.7 *Let α and β be execution fragments of $\mathcal{A}$ with α closed. Then*

$$\alpha \leq \beta \quad \Leftrightarrow \quad \exists \alpha' \in \mathit{frags}_{\mathcal{A}} : \beta = \alpha \frown \alpha'.$$

Proof: Implication "$\Leftarrow$" follows directly from the corresponding implication in Lemma 3.7. Implication "$\Rightarrow$" follows from the definitions and **T2**. ■

The external behavior of a hybrid automaton is captured by the set of "traces" of its execution fragments, which record external actions and the trajectories that describe the evolution of external variables. Formally, if α is an execution fragment, then the *trace* of α, denoted by $\mathit{trace}(\alpha)$, is the (E, W)-restriction of α. (Recall that E denotes the external actions and W the external variables.) A *trace fragment* of a hybrid automaton $\mathcal{A}$ *from* a state $\mathbf{x}$ of $\mathcal{A}$ is the trace of an execution fragment of $\mathcal{A}$ from $\mathbf{x}$. We write $\mathit{tracefrags}_{\mathcal{A}}(\mathbf{x})$ for the set of trace fragments of $\mathcal{A}$ from $\mathbf{x}$. Also, we define a *trace* of $\mathcal{A}$ to be a trace fragment from a start state, that is, the trace of an execution of $\mathcal{A}$, and write $\mathit{traces}_{\mathcal{A}}$ for the set of traces of $\mathcal{A}$.

The following lemma follows trivially from Lemma 3.11:

Lemma 4.8 *If α is an execution fragment of $\mathcal{A}$ then*

(1) α is time-bounded if and only if $\mathit{trace}(\alpha)$ is time-bounded.
(2) α is admissible if and only if $\mathit{trace}(\alpha)$ is admissible.
(3) If α is closed then $\mathit{trace}(\alpha)$ is closed.
(4) If α is non-Zeno then $\mathit{trace}(\alpha)$ is non-Zeno.

In parts (3) and (4) of the above lemma, the converse implications do not hold. Counterexamples can be obtained by taking an execution fragment α that ends with an infinite sequence of internal actions without any delay in between. However, a slight weakening of the converse implications does hold:

Lemma 4.9 *If β is a trace fragment of $\mathcal{A}$ from state $\mathbf{x}$ then*

(1) If β is closed then there exists an execution fragment α of $\mathcal{A}$ from $\mathbf{x}$ such that $\mathit{trace}(\alpha) = \beta$ and α is closed.
(2) If β is non-Zeno then there exists an execution fragment α of $\mathcal{A}$ from $\mathbf{x}$ such that $\mathit{trace}(\alpha) = \beta$ and α is non-Zeno.

If the definition of non-Zeno were broadened to include the case of a right-open final trajectory, then part 2 of the above lemma can fail. It might be that the only execution that leads to such a trace is a Zeno execution, one with infinitely many internal events, and delays which get smaller and smaller.

The next definition defines an implementation relation between hybrid automata in terms of inclusion of traces: a low-level specification $\mathcal{A}$ *implements* a high-level specification $\mathcal{B}$ if any behavior (trace) of $\mathcal{A}$ is also an allowed behavior of $\mathcal{B}$. Without additional assumptions, our implementation relation only preserves safety properties. However, in Section 7 we will see that if the low-level specification automaton is required to be *receptive*, our implementation relation also preserves bounded liveness properties.

Definition 4.10 *Hybrid automata* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *are* comparable *if they have the same external interface, that is, if* $W_1 = W_2$ *and* $E_1 = E_2$*. If* $\mathcal{A}_1$ *and* $\mathcal{A}_2$ *are comparable then we say that* $\mathcal{A}_1$ implements A_2*, denoted by* $\mathcal{A}_1 \leq \mathcal{A}_2$*, if the traces of* $\mathcal{A}_1$ *are included among those of* $\mathcal{A}_2$*, that is, if* $traces_{\mathcal{A}_1} \subseteq traces_{\mathcal{A}_2}$*.*[7]

4.3 Simulation Relations

In this subsection, we define simulation relations between hybrid automata. Simulation relations may be used to show that one HA implements another, in the sense of inclusion of sets of traces.

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

(1) If $\mathbf{x}_{\mathcal{A}} \in \Theta_{\mathcal{A}}$ then there exists a state $\mathbf{x}_{\mathcal{B}} \in \Theta_{\mathcal{B}}$ such that $\mathbf{x}_{\mathcal{A}}\ R\ \mathbf{x}_{\mathcal{B}}$.
(2) If $\mathbf{x}_{\mathcal{A}}\ R\ \mathbf{x}_{\mathcal{B}}$ and α is an execution fragment of $\mathcal{A}$ consisting of one action surrounded by two point trajectories, with $\alpha.\mathit{fstate} = \mathbf{x}_{\mathcal{A}}$, then $\mathcal{B}$ has a closed execution fragment β with $\beta.\mathit{fstate} = \mathbf{x}_{\mathcal{B}}$, $\mathit{trace}(\beta) = \mathit{trace}(\alpha)$, and $\alpha.\mathit{lstate}\ R\ \beta.\mathit{lstate}$.
(3) If $\mathbf{x}_{\mathcal{A}}\ R\ \mathbf{x}_{\mathcal{B}}$ and α is an execution fragment of $\mathcal{A}$ consisting of a single closed trajectory, with $\alpha.\mathit{fstate} = \mathbf{x}_{\mathcal{A}}$, then $\mathcal{B}$ has a closed execution fragment β with $\beta.\mathit{fstate} = \mathbf{x}_{\mathcal{B}}$, $\mathit{trace}(\beta) = \mathit{trace}(\alpha)$, and $\alpha.\mathit{lstate}\ R\ \beta.\mathit{lstate}$.

The definition of a simulation from $\mathcal{A}$ to $\mathcal{B}$ yields a correspondence for open trajectories:

Lemma 4.11 *Let* $\mathcal{A}$ *and* $\mathcal{B}$ *be comparable HAs and let* R *be a simulation from* $\mathcal{A}$ *to* $\mathcal{B}$*. Let* $\mathbf{x}_{\mathcal{A}}$ *and* $\mathbf{x}_{\mathcal{B}}$ *be states of* $\mathcal{A}$ *and* $\mathcal{B}$*, respectively, such that* $\mathbf{x}_{\mathcal{A}}\ R\ \mathbf{x}_{\mathcal{B}}$*. Let* α *be an execution fragment of* $\mathcal{A}$ *from state* $\mathbf{x}_{\mathcal{A}}$ *consisting of a single open trajectory. Then* $\mathcal{B}$ *has an execution fragment* β *with* $\beta.\mathit{fstate} = \mathbf{x}_{\mathcal{B}}$ *and* $\mathit{trace}(\beta) = \mathit{trace}(\alpha)$*.*

Proof: Let τ be the single open trajectory in α. Using axioms **T1** and **T2**, we construct an infinite sequence $\tau_0, \tau_1, \ldots$ of closed trajectories of $\mathcal{A}$ such that

[7] In [60,27,53,54], definitions of the set of traces of an automaton and of one automaton implementing another are based on closed and admissible executions only. The results we obtain in this paper using the newer, more inclusive definition imply corresponding results for the earlier definition. For example, we have the following property: If $\mathcal{A}_1 \leq \mathcal{A}_2$ then the set of traces that arise from closed or admissible executions of $\mathcal{A}_1$ is a subset of the set of traces that arise from closed or admissible executions of $\mathcal{A}_2$.

$\tau = \tau_0 \frown \tau_1 \frown \cdots$. Then, working inductively, we construct a sequence $\beta_0, \beta_1, \ldots$ of closed execution fragments of $\mathcal{B}$ such that $\beta_0.\mathit{fstate} = \mathbf{x}_\mathcal{B}$ and, for each i, $\tau_i.\mathit{lstate}\ R\ \beta_i.\mathit{lstate}$, $\beta_i.\mathit{lstate} = \beta_{i+1}.\mathit{fstate}$, and $\mathit{trace}(\tau_i) = \mathit{trace}(\beta_i)$. This construction uses induction on i, using Property 3 of the definition of a simulation relation in the induction step. Now let $\beta = \beta_0 \frown \beta_1 \frown \cdots$. By Lemma 4.6, β is an execution fragment of $\mathcal{B}$. Clearly, $\beta.\mathit{fstate} = \mathbf{x}_\mathcal{B}$. By Lemma 3.9 applied to both α and β, $\mathit{trace}(\beta) = \mathit{trace}(\alpha)$. Thus β has the required properties. ■

Theorem 4.12 *Let $\mathcal{A}$ and $\mathcal{B}$ be comparable HAs and let R be a simulation from $\mathcal{A}$ to $\mathcal{B}$. Let $\mathbf{x}_\mathcal{A}$ and $\mathbf{x}_\mathcal{B}$ be states of $\mathcal{A}$ and $\mathcal{B}$, respectively, such that $\mathbf{x}_\mathcal{A}\ R\ \mathbf{x}_\mathcal{B}$. Then $\mathit{tracefrags}_\mathcal{A}(\mathbf{x}_\mathcal{A}) \subseteq \mathit{tracefrags}_\mathcal{B}(\mathbf{x}_\mathcal{B})$.*

Proof: Suppose that δ is the trace of an execution fragment of $\mathcal{A}$ that starts from $\mathbf{x}_\mathcal{A}$; we prove that δ is also a trace of an execution fragment of $\mathcal{B}$ that starts from $\mathbf{x}_\mathcal{B}$. Let $\alpha = \tau_0\, a_1\, \tau_1\, a_2\, \tau_2 \ldots$ be an execution fragment of $\mathcal{A}$ such that $\alpha.\mathit{fstate} = \mathbf{x}_\mathcal{A}$ and $\delta = \mathit{trace}(\alpha)$. We consider cases:

(1) α is an infinite sequence.
Using axioms **T1** and **T2**, we can write α as an infinite concatenation $\alpha_0 \frown \alpha_1 \frown \alpha_2 \cdots$, in which the execution fragments α_i with i even consist of a trajectory only, and the execution fragments α_i with i odd consist of a single discrete step surrounded by two point trajectories.
We define inductively a sequence $\beta_0, \beta_1, \ldots$ of closed execution fragments of $\mathcal{B}$, such that $\beta_0.\mathit{fstate} = \mathbf{x}_\mathcal{B}$ and, for all i, $\beta_i.\mathit{lstate} = \beta_{i+1}.\mathit{fstate}$, $\alpha_i.\mathit{lstate}\ R\ \beta_i.\mathit{lstate}$, and $\mathit{trace}(\beta_i) = \mathit{trace}(\alpha_i)$. We use Property 3 of the definition of a simulation relation for the construction of the β_i's with i even, and Property 2 for the construction of the β_i's with i odd. Let $\beta = \beta_0 \frown \beta_1 \frown \beta_2 \cdots$. By Lemma 4.6, β is an execution fragment of $\mathcal{B}$. Clearly, $\beta.\mathit{fstate} = \mathbf{x}_\mathcal{B}$. By Lemma 3.9, $\mathit{trace}(\beta) = \mathit{trace}(\alpha)$. Thus β has the required properties.

(2) α is a finite sequence ending with a closed trajectory.
Similar to the first case.

(3) α is a finite sequence ending with an open trajectory.
Similar to the first case, using Lemma 4.11. ■

Corollary 4.13 *Let $\mathcal{A}$ and $\mathcal{B}$ be comparable HAs and let R be a simulation from $\mathcal{A}$ to $\mathcal{B}$. Then $\mathit{traces}_\mathcal{A} \subseteq \mathit{traces}_\mathcal{B}$.*

Proof: Suppose $\beta \in \mathit{traces}_\mathcal{A}$. Then $\beta \in \mathit{tracefrags}_\mathcal{A}(\mathbf{x}_\mathcal{A})$ for some start state $\mathbf{x}_\mathcal{A}$ of $\mathcal{A}$. Property 1 of the definition of simulation relation implies the existence of a start state $\mathbf{x}_\mathcal{B}$ of $\mathcal{B}$ such that $\mathbf{x}_\mathcal{A}\ R\ \mathbf{x}_\mathcal{B}$. Then Theorem 4.12 implies that $\beta \in \mathit{tracefrags}_\mathcal{B}(\mathbf{x}_\mathcal{B})$. Since $\mathbf{x}_\mathcal{B}$ is a start state of $\mathcal{B}$, this implies that $\beta \in \mathit{traces}_\mathcal{B}$, as needed. ■

Example 4.14 (Vehicle implementation) Now denote the *Vehicle* HA of Example 4.2 by $\mathit{Vehicle}(\epsilon)$, making the uncertainty parameter explicit. Assume that $0 \leq \epsilon_1 \leq \epsilon_2$. Let $\mathcal{A} = \mathit{Vehicle}(\epsilon_1)$ and $\mathcal{B} = \mathit{Vehicle}(\epsilon_2)$. We claim that $\mathcal{A} \leq \mathcal{B}$. We can show this by demonstrating that the identity mapping is a simulation relation from $\mathcal{A}$ to $\mathcal{B}$. Since these HAs have no discrete steps, we need only show Properties 1 and 3 of the definition of simulation relation. Property 1 is obvious because the two HAs have the same (unique) start state, which assigns 0 to both state variables. For Property 3, assume that $\mathbf{x}_{\mathcal{A}}\ R\ \mathbf{x}_{\mathcal{B}}$ and α consists of a closed trajectory τ of $\mathcal{A}$ with $\alpha.\mathit{fstate} = \mathbf{x}_{\mathcal{A}}$. Let $\beta = \alpha$. Clearly, β is a closed hybrid sequence, $\beta.\mathit{fstate} = \mathbf{x}_{\mathcal{B}}$, $\mathit{trace}(\beta) = \mathit{trace}(\alpha)$, and $\alpha.\mathit{lstate}\ R\ \beta.\mathit{lstate}$. It remains to show that β is an execution fragment of $\mathcal{B}$, that is, that τ is a trajectory of $\mathcal{B}$. This follows immediately from the definition of trajectories for $\mathit{Vehicle}(\epsilon_1)$ and $\mathit{Vehicle}(\epsilon_2)$; the only interesting point is that, for every $t \in \mathit{dom}(\tau)$, $t > 0$, we have: $[\mathit{acc\text{-}in}(t) - \epsilon_1, \mathit{acc\text{-}in}(t) + \epsilon_1] \subseteq [\mathit{acc\text{-}in}(t) - \epsilon_2, \mathit{acc\text{-}in}(t) + \epsilon_2]$. ■

Example 4.15 (Controller implementation) Denote the *Controller* HA of Example 4.3 by $\mathit{Controller}(\mathsf{vmax})$, making the maximum velocity parameter explicit. Assume that $0 \leq \mathsf{vmax}_1 \leq \mathsf{vmax}_2$. We claim that $\mathit{Controller}(\mathsf{vmax}_1) \leq \mathit{Controller}(\mathsf{vmax}_2)$; again, we show this by demonstrating that the identity mapping is a simulation relation. This requires showing all three properties of the definition of simulation relation. Properties 1 and 3 are immediate, because vmax does not appear in the definitions of the start states and the trajectories. For Property 2, the key is that, if $\mathit{vel\text{-}sensed} + (\mathit{acc\text{-}suggested}' + \epsilon)\mathsf{d} \leq \mathsf{vmax}_1$, then also $\mathit{vel\text{-}sensed} + (\mathit{acc\text{-}suggested})' + \epsilon)\mathsf{d} \leq \mathsf{vmax}_2$. ■

5 Operations on Hybrid Automata

In this section, we present two kinds of operations on hybrid automata: parallel composition and hiding.

5.1 Composition

We now introduce the operation of parallel composition for hybrid automata, which allows an automaton representing a complex system to be constructed by composing automata representing individual system components. Our composition operation identifies external actions with the same name in different component automata, and likewise for external variables. When any component automaton performs a discrete step involving an action a, so do all component automata that have a in their signatures. Likewise, when any component automaton performs a trajectory involving a particular evolution of values for an external variable v, then so do all component automata that have v in their signatures. We prove several results that say that the composition operation respects our notions of external behavior and implementation.

We define composition as a partial, binary operation on hybrid automata. Since internal actions of an automaton $\mathcal{A}_1$ are intended to be unobservable by any other automaton $\mathcal{A}_2$, we allow $\mathcal{A}_1$ to be composed with $\mathcal{A}_2$ only if the internal actions of $\mathcal{A}_1$ are disjoint from the actions of $\mathcal{A}_2$. Similarly, we require disjointness of the internal variables of $\mathcal{A}_1$ and the variables of $\mathcal{A}_2$.

Definition 5.1 *We say that hybrid automata $\mathcal{A}_1$ and $\mathcal{A}_2$ are* compatible *if $H_1 \cap A_2 = H_2 \cap A_1 = \emptyset$ and $X_1 \cap V_2 = X_2 \cap V_1 = \emptyset$. 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 $\mathcal{A} = (W, X, Q, \Theta, E, H, D, \mathcal{T})$ where*

- $W = W_1 \cup W_2$ *and* $X = X_1 \cup X_2$.
- $Q = \{\mathbf{x} \in val(X) \mid \mathbf{x} \lceil X_1 \in Q_1 \wedge \mathbf{x} \lceil X_2 \in Q_2\}$.
- $\Theta = \{\mathbf{x} \in Q \mid \mathbf{x} \lceil X_1 \in \Theta_1 \wedge \mathbf{x} \lceil X_2 \in \Theta_2\}$.
- $E = E_1 \cup E_2$ *and* $H = H_1 \cup H_2$.
- *For each* $\mathbf{x}, \mathbf{x}' \in Q$ *and each* $a \in A$, $\mathbf{x} \xrightarrow{a}_{\mathcal{A}} \mathbf{x}'$ *iff for* $i = 1, 2$, *either (1)* $a \in A_i$ *and* $\mathbf{x} \lceil X_i \xrightarrow{a}_i \mathbf{x}' \lceil X_i$, *or (2)* $a \notin A_i$ *and* $\mathbf{x} \lceil X_i = \mathbf{x}' \lceil X_i$.
- $\mathcal{T} \subseteq trajs(V)$ *is given by* $\tau \in \mathcal{T} \Leftrightarrow \tau \downarrow V_1 \in \mathcal{T}_1 \;\wedge\; \tau \downarrow V_2 \in \mathcal{T}_2$.

Whenever we write $\mathcal{A}_1 \| \mathcal{A}_2$, we implicitly assume that $\mathcal{A}_1$ and $\mathcal{A}_2$ are compatible.

Theorem 5.2 *If $\mathcal{A}_1$ and $\mathcal{A}_2$ are hybrid automata then $\mathcal{A}_1 \| \mathcal{A}_2$ is a hybrid automaton.*

Proof: Let $\mathcal{A}$ denote $\mathcal{A}_1 \| \mathcal{A}_2$ as above. We show that $\mathcal{A}$ satisfies the properties of a hybrid automaton (cf. Section 4.1). Disjointness of W and X follows from disjointness of W_1 and X_1, disjointness of W_2 and X_2, and compatibility. Similarly, disjointness of E and H follows from disjointness of E_1 and H_1, disjointness of E_2 and H_2, and compatibility. Nonemptiness of Θ follows from nonemptiness of Θ_1 and Θ_2 and disjointness of X_1 and X_2. We verify the **T** properties:

T1 Let $\tau \in \mathcal{T}$, let τ' be a trajectory such that $\tau' \leq \tau$, and let $i \in \{1, 2\}$. By the definition of composition, $\tau \downarrow V_i \in \mathcal{T}_i$. By the definition of prefix, $\tau' \downarrow V_i \leq \tau \downarrow V_i$. By **T1** applied to $\mathcal{A}_i$, $\tau' \downarrow V_i \in \mathcal{T}_i$. Then by definition of composition, $\tau' \in \mathcal{T}$, as needed.

T2 Let $\tau \in \mathcal{T}$, $t \in dom(\tau)$, $\tau' = \tau \trianglerighteq t$, and $i \in \{1, 2\}$. By the definition of composition, $\tau \downarrow V_i \in \mathcal{T}_i$. Then by **T2** applied to $\mathcal{A}_i$, $(\tau \downarrow V_i) \trianglerighteq t \in \mathcal{T}_i$. Observe that $(\tau \downarrow V_i) \trianglerighteq t = \tau' \downarrow V_i$; therefore, $\tau' \downarrow V_i \in \mathcal{T}_i$. Then by the definition of composition, $\tau' \in \mathcal{T}$, as needed.

T3 Let $\tau_0, \tau_1, \tau_2, \ldots$ be a sequence of trajectories in $\mathcal{T}$ such that, for each nonfinal index j, τ_j is closed and $\tau_j.lstate = \tau_{j+1}.fstate$. Let τ denote $\tau_0 \frown \tau_1 \frown \tau_2 \cdots$, and let $i \in \{1, 2\}$. By the definition of composition, operation, for each index j, $\tau_j \downarrow V_i \in \mathcal{T}_i$, and for each nonfinal index j, $\tau_j \downarrow V_i$ is closed and $(\tau_j \downarrow V_i).lstate = (\tau_{j+1} \downarrow V_i).fstate$. By **T3** applied to $\mathcal{A}_i$, $\tau_0 \downarrow V_i \frown \tau_1 \downarrow V_i \frown \tau_2 \downarrow V_i \cdots \in \mathcal{T}_i$. Observe that $\tau \downarrow V_i = \tau_0 \downarrow V_i \frown \tau_1 \downarrow V_i \frown \tau_2 \downarrow V_i \cdots$; therefore, $\tau \downarrow V_i \in \mathcal{T}_i$. Then by the definition of composition, $\tau \in \mathcal{T}$, as needed. ■

The following "projection lemma" says that executions of a composition of HAs project to give executions of the component automata. Moreover, certain properties of the executions of the composition imply, or are implied by, similar properties for the component executions.

Lemma 5.3 *Let $\mathcal{A} = \mathcal{A}_1 \| \mathcal{A}_2$ and let α be an execution fragment of $\mathcal{A}$. Then $\alpha \lceil (A_1, V_1)$ and $\alpha \lceil (A_2, V_2)$ are execution fragments of $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively. Furthermore,*

(1) α is time-bounded iff both $\alpha \lceil (A_1, V_1)$ and $\alpha \lceil (A_2, V_2)$ are time-bounded.
(2) α is admissible iff both $\alpha \lceil (A_1, V_1)$ and $\alpha \lceil (A_2, V_2)$ are admissible.
(3) α is closed iff both $\alpha \lceil (A_1, V_1)$ and $\alpha \lceil (A_2, V_2)$ are closed.
(4) α is Zeno iff at least one of $\alpha \lceil (A_1, V_1)$ and $\alpha \lceil (A_2, V_2)$ is Zeno.
(5) α is an execution iff both $\alpha \lceil (A_1, V_1)$ and $\alpha \lceil (A_2, V_2)$ are executions.

Proof: Simple application of the definitions. ■

Example 5.4 (Composition and Zeno executions) Consider a composition $\mathcal{A} = \mathcal{A}_1 \| \mathcal{A}_2$ in which the two components have no actions or variables in common. We describe a Zeno execution fragment α of $\mathcal{A}$ in which only one of the projected execution fragments is Zeno. Namely, let $\alpha = \tau_0\, a_1\, \tau_1\, a_2\, \tau_2 \ldots$, where $\tau_0.ltime = 1$ and for all $i \geq 1$, τ_i is a point trajectory. Also, all the a_i's are actions of $\mathcal{A}_1$ but not of $\mathcal{A}_2$. Then $\alpha \lceil (A_1, V_1)$, which includes all the a_i's, is a Zeno execution fragment, whereas $\alpha \lceil (A_2, V_2)$, which consists of the single right-closed trajectory $\tau_0 \downarrow V_2$, is a closed execution fragment. ■

Example 5.5 (Execution of vehicle and controller) Consider the *Vehicle* and *Controller* automata of Examples 4.2 and 4.3 (for the same ϵ). These two HAs are compatible. Their composition is displayed in Figure 5. An example execution of the composition is the infinite sequence $\alpha = \tau_0 suggest \tau_1 suggest \tau_2 \ldots$, where, for every i and for every $t \in dom(\tau_i)$:

(1) $\tau_i.ltime = 1$.

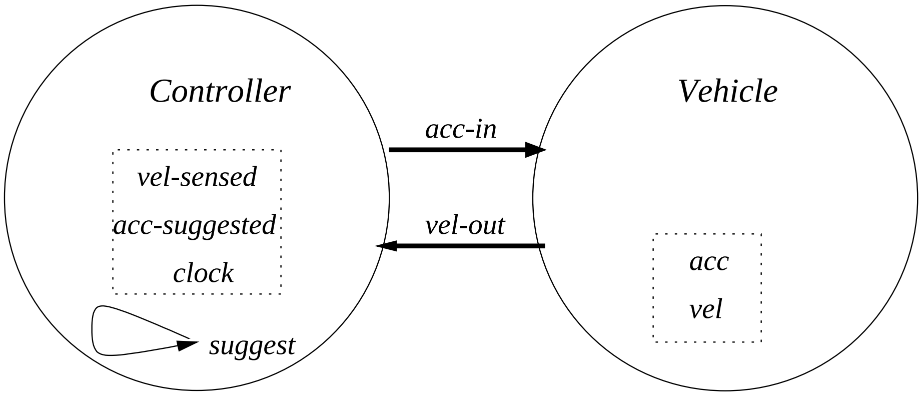

Fig. 5. Composition of hybrid automata *Vehicle* and *Controller*.

(2) $\tau_i(t)(clock) = t$.
(3) If $i = 0$ then $\tau_i(t)(v)$ is equal to 0 for $v \in \{acc\text{-}suggested, acc\text{-}in\}$, ϵ for $v = acc$, and ϵt for $v \in \{vel, vel\text{-}out, vel\text{-}sensed\}$.
(4) If $1 \leq i \leq 2$ then $\tau_i(t)(v)$ is equal to 2 for $v \in \{acc\text{-}suggested, acc\text{-}in\}$, $2 + \epsilon$ for $v = acc$, and $(2 + \epsilon)(i + t) - 2$ for $v \in \{vel, vel\text{-}out, vel\text{-}sensed\}$.
(5) If $i \geq 3$ then $\tau_i(t)(v)$ is equal to 0 for $v \in \{acc\text{-}suggested, acc\text{-}in, acc\}$ and $4 + 3\epsilon$ for $v \in \{vel, vel\text{-}out, vel\text{-}sensed\}$.

This execution is admissible. Its projections on the *Vehicle* and *Controller* automata are given by the admissible executions in Examples 4.4 and 4.5, respectively. ■

The following lemma says that we obtain the same result for an execution fragment α of a composition if we first extract the trace and then restrict to one of the components, or if we first restrict to the component and then take the trace.

Lemma 5.6 *Let $\mathcal{A} = \mathcal{A}_1 \| \mathcal{A}_2$, and let α be an execution fragment of $\mathcal{A}$. Then, for $i = 1, 2$, $trace(\alpha) \lceil (E_i, W_i) = trace(\alpha \lceil (A_i, V_i))$.*

Proof: Recall that $trace(\alpha) = \alpha \lceil (E, W)$. The result follows straightforwardly by Lemma 3.10 and the observation that $W \cap W_i = W_i = V_i \cap W_i$ and $E \cap E_i = E_i = A_i \cap E_i$. ■

The following fundamental theorem relates the set of traces of a composed automaton to the sets of traces of the component automata. It is expressed in terms of equality between two sets of traces. Set inclusion in one direction expresses the idea that a trace of a composition "projects" to yield traces of the components. Set inclusion in the other direction expresses the idea that traces of components can be "pasted together" to yield a trace of the composition.

Theorem 5.7 *Let $\mathcal{A} = \mathcal{A}_1 \| \mathcal{A}_2$. Then $traces_{\mathcal{A}}$ is exactly the set of (E, W)-*

sequences whose restrictions to $\mathcal{A}_1$ and $\mathcal{A}_2$ are traces of $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively. That is,

$$traces_{\mathcal{A}} = \{\beta \mid \beta \text{ is } (E,W)\text{-sequence and } \beta \lceil (E_i, W_i) \in traces_{\mathcal{A}_i}, i = 1, 2\}.$$

Proof: For one direction, suppose that β is a trace of $\mathcal{A}$. Then by definition, β is an (E,W)-sequence. Let α be an execution of $\mathcal{A}$ such that $\beta = trace(\alpha)$. Let $i \in \{1,2\}$. Then Lemma 5.6 implies that $\beta \lceil (E_i, W_i) = trace(\alpha \lceil (A_i, V_i))$. Since, by Lemma 5.3, $\alpha \lceil (A_i, V_i)$ is an execution of $\mathcal{A}_i$, $\beta \lceil (E_i, W_i)$ is a trace of $\mathcal{A}_i$.

Conversely, let β be an (E,W)-sequence such that $\beta \lceil (E_i, W_i)$ is a trace of $\mathcal{A}_i$, $i = 1, 2$. Then there are executions α_1 and α_2 of $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively, such that, for $i = 1, 2$, $trace(\alpha_i) = \beta \lceil (E_i, W_i)$. Decompose α_1 into $\alpha_1^0 \frown \alpha_1^1 \frown \alpha_1^2 \frown \cdots$, decompose α_2 into $\alpha_2^0 \frown \alpha_2^1 \frown \alpha_2^2 \frown \cdots$, and decompose β into $\beta^0 \frown \beta^1 \frown \beta^2 \frown \cdots$ in such a way that for each j, (1) $trace(\alpha_i^j) = \beta^j \lceil (E_i, W_i)$ for $i \in \{1,2\}$, (2) α_i^j is either a trajectory or an action surrounded by point trajectories, $i \in \{1,2\}$, and (3) if both α_1^j and α_2^j consist of actions surrounded by point trajectories then these actions are identical. Axioms **T1** and **T2** imply that such decompositions exist.[8]

Now we define a sequence of execution fragments of $\mathcal{A}$, $\alpha^0, \alpha^1, \ldots$, such that:

(1) $\alpha^0.fstate \in \Theta_{\mathcal{A}}$,
(2) For every nonfinal j, $\alpha^j.lstate = \alpha^{j+1}.fstate$, and
(3) For every j, $trace(\alpha^j) = \beta^j$.

By Lemma 4.6, the concatenation $\alpha^0 \frown \alpha^1 \frown \cdots$ is an execution of $\mathcal{A}$. Moreover, by Lemma 3.9, the trace of this execution is β. To define each α^j, we distinguish the following cases:

(1) Each of α_1^j and α_2^j is a trajectory.
Then suppose that $\alpha_1^j = \tau_1$ and $\alpha_2^j = \tau_2$. Define α^j to be the function τ with domain $dom(\tau_1)$ such that $\tau(t) = \tau_1(t) \cup \tau_2(t)$ for every t. (Compatibility of τ_1 and τ_2 follows here, and in the remaining three cases, from the facts that $\alpha_1^j = \beta^j \lceil (E_1, W_1)$ and $\alpha_2^j = \beta^j \lceil (E_2, W_2)$.)
(2) α_1^j is a trajectory and α_2^j is an action surrounded by point trajectories.
Then α_1^j must be a point trajectory as well. Let $\alpha_1^j = \wp(\mathbf{v}_1)$ and $\alpha_2^j = \wp(\mathbf{v}_2) a \wp(\mathbf{v}_2')$. Then define α^j to be $\wp(\mathbf{v}_1 \cup \mathbf{v}_2)\, a\, \wp(\mathbf{v}_1 \cup \mathbf{v}_2')$.
(3) α_1^j is an action surrounded by point trajectories and α_2^j is a trajectory.
This is symmetric with the previous case.
(4) Each of α_1^j and α_2^j is an action (the same in both cases) surrounded by point trajectories.

[8] See [59] for a detailed existence proof for similar decompositions.

Let $\alpha_1^j = \wp(\mathbf{v}_1)a\wp(\mathbf{v}_1')$ and $\alpha_2^j = \wp(\mathbf{v}_2)a\wp(\mathbf{v}_2')$. Define α^j to be $\wp(\mathbf{v}_1 \cup \mathbf{v}_2)\ a\ \wp(\mathbf{v}_1' \cup \mathbf{v}_2')$.

It is straightforward to verify that the α^j fragments satisfy the required properties. ■

The following theorem describes a basic substitutivity property:

Theorem 5.8 *Suppose $\mathcal{A}_1$ and $\mathcal{A}_2$ are comparable HAs with $\mathcal{A}_1 \leq \mathcal{A}_2$. Suppose $\mathcal{B}$ is an HA that is compatible with each of $\mathcal{A}_1$ and $\mathcal{A}_2$. Then $\mathcal{A}_1 \| B$ and $\mathcal{A}_2 \| B$ are comparable and $\mathcal{A}_1 \| \mathcal{B} \leq \mathcal{A}_2 \| \mathcal{B}$.*

Proof: The fact that $\mathcal{A}_1 \| B$ and $\mathcal{A}_2 \| B$ are comparable follows from the fact that $\mathcal{A}_1$ and $\mathcal{A}_2$ are comparable and the definition of composition.

Let $\beta \in traces_{\mathcal{A}_1 \| \mathcal{B}}$. By Theorem 5.7, $\beta \lceil (E_1, W_1) \in traces_{\mathcal{A}_1}$ and $\beta \lceil (E_{\mathcal{B}}, W_{\mathcal{B}}) \in traces_{\mathcal{B}}$. Since $\mathcal{A}_1 \leq \mathcal{A}_2$, $\beta \lceil (E_1, W_1) \in traces_{\mathcal{A}_2}$. Since $\mathcal{A}_1$ and $\mathcal{A}_2$ have the same external interface, $(E_1, W_1) = (E_2, W_2)$. Thus, $\beta \lceil (E_2, W_2) \in traces_{\mathcal{A}_2}$. It follows from Theorem 5.7 that $\beta \in traces_{\mathcal{A}_2 \| \mathcal{B}}$. ■

Example 5.9 (Invariant for combined vehicle and controller) Consider again the composition of the *Vehicle* and *Controller* automata of Examples 4.2 and 4.3 (for the same ϵ). In the composed automaton, it turns out that the velocity is always less than or equal to vmax, that is, in all reachable states,

$$vel \leq \mathsf{vmax} \tag{12}$$

This statement may be proved by induction on the length of closed execution fragments. In the proof, we use the fact that $clock \leq \mathsf{d}$, which follows from the definition of Q. We also use assertions (3) and (11). In addition, we require the following auxiliary invariants:

$$vel + (acc\text{-}suggested + \epsilon)(\mathsf{d} - clock) \leq \mathsf{vmax} \tag{13}$$

$$clock > 0 \Rightarrow \quad acc \leq acc\text{-}suggested + \epsilon \tag{14}$$

$$vel\text{-}sensed = vel \tag{15}$$

$$0 \leq clock \tag{16}$$

Here the interesting assertion is (13), which says, essentially, that the velocity will stay less than or equal to vmax if the vehicle accelerates at the currently suggested acceleration plus ϵ until the next recalculation. The main invariant (12) and the auxiliary invariants (13)-(16) can all be proved together. All are easily seen to be true in the initial state. There are two kinds of inductive steps, for discrete *suggest* transitions and for trajectories. Discrete transitions are easily seen to preserve all the assertions; the most interesting property to show is invariant (13), which holds because of the constraints on the new suggested acceleration, the fact that $\mathit{vel\text{-}sensed} = \mathit{vel}$, and the fact that, in the new state, $\mathit{clock} = 0$.

Trajectories also preserve all the assertions; now the interesting thing to show is the conjunction of (12) and (13). Depending on whether or not $\mathit{acc\text{-}suggested} + \epsilon \geq 0$, it suffices to show only (12) or only (13). For example, suppose $\mathit{acc\text{-}suggested} + \epsilon \geq 0$; we show the auxiliary invariant (13). The trajectory guarantees that $\mathit{vel}' \leq \mathit{vel} + (\mathit{acc\text{-}suggested} + \epsilon)t$ and $\mathit{clock}' = \mathit{clock} + t$, where t is the limit time of the trajectory and unprimed and primed instances of the variables are used (as usual) to indicate their values at the beginning and end of the trajectory, respectively. The inequality is based on the integral definition of vel in terms of acc and the relationship between acc and $\mathit{acc\text{-}suggested}$. Then

$$
\begin{aligned}
&\mathit{vel}' + (\mathit{acc\text{-}suggested}' + \epsilon)(\mathsf{d} - \mathit{clock}') \\
&= \mathit{vel}' + (\mathit{acc\text{-}suggested} + \epsilon)(\mathsf{d} - \mathit{clock} - t) \\
&= \mathit{vel}' - (\mathit{acc\text{-}suggested} + \epsilon)t + (\mathit{acc\text{-}suggested} + \epsilon)(\mathsf{d} - \mathit{clock}) \\
&\leq \mathit{vel} + (\mathit{acc\text{-}suggested} + \epsilon)(\mathsf{d} - \mathit{clock}) \\
&\leq \mathsf{vmax} \quad \text{(by inductive hypothesis)}
\end{aligned}
$$

Note that, because of the two kinds of inductive steps, the inductive proof divides cleanly into separate parts that involve discrete and continuous reasoning. ■

5.2 Hiding

We define two hiding operations for hybrid automata, which hide external actions and external variables, respectively, and we prove that these operations respect the implementation relationship. The hiding operations reclassify external actions or external variables as internal actions or variables.

- If $E \subseteq E_{\mathcal{A}}$, 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 $H_{\mathcal{B}} = H_{\mathcal{A}} \cup E$.
- 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 $\mathcal{T}_{\mathcal{B}} = \mathcal{T}_{\mathcal{A}} \downarrow (V_{\mathcal{A}} - W)$.

Lemma 5.10 *Let $E \subseteq E_{\mathcal{A}}$ and $W \subseteq W_{\mathcal{A}}$. Then $\mathsf{ActHide}(E, \mathcal{A})$ and $\mathsf{VarHide}(W, \mathcal{A})$ are HAs.*

Proof: This is a straightforward application of the definitions. ■

The following lemma characterizes the traces of the automata that result from applying the hiding operations:

Lemma 5.11 *Let $\mathcal{A}$ be an HA.*

(1) If $E \subseteq E_{\mathcal{A}}$ then $traces_{\mathsf{ActHide}(E,\mathcal{A})} = \{\beta \lceil (E_{\mathcal{A}} - E, V_{\mathcal{A}}) \mid \beta \in traces_{\mathcal{A}}\}$.
(2) If $W \subseteq W_{\mathcal{A}}$ then $traces_{\mathsf{VarHide}(W,\mathcal{A})} = \{\beta \lceil (A_{\mathcal{A}}, W_{\mathcal{A}} - W) \mid \beta \in traces_{\mathcal{A}}\}$.

Proof: For (1), first observe that $\mathsf{ActHide}(E, \mathcal{A})$ has the same set of executions as $\mathcal{A}$. Then apply Lemma 3.10. The proof of (2) is straightforward. ∎

Theorem 5.12 *Suppose $\mathcal{A}$ and $\mathcal{B}$ are HAs with $\mathcal{A} \leq \mathcal{B}$, and suppose $E \subseteq E_{\mathcal{A}}$ 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: Straightforward, using Lemma 5.11. ∎

Example 5.13 (Implementing a velocity specification) In the composition of the *Vehicle* and *Controller* automata defined in Example 5.5, we may hide the *acc-in* variable used for communication between the two components. Thus, we define

$$\mathcal{A} = \mathsf{VarHide}(\{acc\text{-}in\}, Vehicle \| Controller).$$

In the resulting automaton $\mathcal{A}$, the only external variable is *vel-out*.

We may express the correctness of $\mathcal{A}$ by showing that it implements an abstract specification automaton *VSpec*, displayed in Figure 6, that simply represents the constraint that the vehicle's velocity is at most vmax. *VSpec* has one ex-

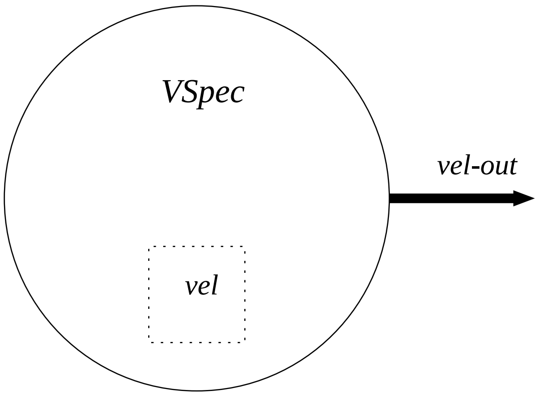

Fig. 6. Specification automaton *VSpec*.

ternal variable *vel-out*, one state variable *vel*, and the sets of states and initial states both consist of all valuations satisfying $vel \leq \mathsf{vmax}$. Both variables have type R and dynamic type equal to the (pasting closure of the) continuous functions. *VSpec* has no actions. The trajectories of *VSpec* are those that satisfy:

$$\mathit{vel\text{-}out} = \mathit{vel} \tag{17}$$

We may argue that $\mathcal{A}$ implements *VSpec* using a simulation relation R. Most of the work has already been done by proving invariants, in Example 5.9. Relation R relates states $\mathbf{x}_\mathcal{A}$ of $\mathcal{A}$ and $\mathbf{x}_\mathcal{B}$ of $\mathcal{B} \triangleq \mathit{VSpec}$ exactly if $\mathbf{x}_\mathcal{A}$ is a reachable state of $\mathcal{A}$ and $\mathbf{x}_\mathcal{B}(\mathit{vel}) = \mathbf{x}_\mathcal{A}(\mathit{vel})$. It is easy to see that R satisfies the start condition of the simulation relation definition. The discrete step condition follows because discrete actions of $\mathcal{A}$ do not change *vel*. For the trajectory condition, assume $\mathbf{x}_\mathcal{A}\ R\ \mathbf{x}_\mathcal{B}$ and τ is a trajectory of $\mathcal{A}$ with first state $\mathbf{x}_\mathcal{A}$. The definition of R implies that $\mathbf{x}_\mathcal{A}$ is a reachable state of $\mathcal{A}$. Therefore all states in trajectory τ are also reachable states of $\mathcal{A}$. Therefore, the invariant $\mathit{vel} \le \mathsf{vmax}$, which was proved for $\mathcal{A}$ in Example 5.9, is also true of all states in τ. Now define the corresponding execution fragment of $\mathcal{B}$ to consist of the single trajectory τ' such that $\tau' \downarrow \mathit{vel} = \tau' \downarrow \mathit{vel\text{-}out} = \tau \downarrow \mathit{vel}$. This satisfies all the required properties. ■

Example 5.14 (Sensor and discrete controller) We describe how to implement the *Controller* of Example 4.3, which receives continuous information about the vehicle's velocity through *vel-out* and suggests accelerations, using two other components: a *Sensor*, which periodically samples the continuous velocity information and produces discrete velocity reports, and a *DiscreteController*, which uses the discrete velocity reports and immediately suggests accelerations. These two components are displayed in Figure 7.

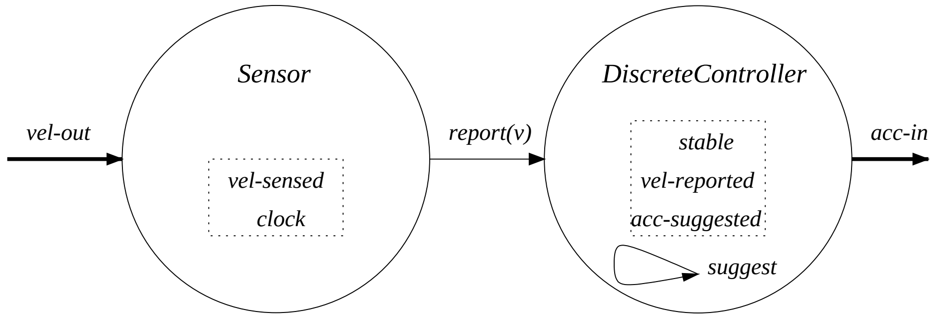

Fig. 7. The hybrid automata *Sensor* and *DiscreteController*.

The *Sensor* automaton has state variables *clock* and *vel-sensed*, both initially 0, and external variable *vel-out*. All variables have type R and dynamic type equal to the (pasting closure of the) continuous functions. The set Q of states consists of all valuations in which $\mathit{clock} \le \mathsf{d}$. *Sensor* also has external actions $\mathit{report}(v)$, $v \in \mathsf{R}$. D consists of $\mathit{report}(v)$ steps specified by:

$$\begin{aligned} clock &= \mathsf{d} && (18)\\ clock' &= 0 && (19)\\ v &= vel\text{-}sensed && (20)\end{aligned}$$

That is, when the clock reaches d, the *Sensor* may reset the clock to 0 and report the current velocity. Set $\mathcal{T}$ consists of trajectories that satisfy:

$$\begin{aligned} \dot{clock} &= 1 && (21)\\ vel\text{-}sensed(t) &= vel\text{-}out(t) \quad \text{for } t > 0 && (22)\end{aligned}$$

That is, the clock increases at rate 1 and the velocity sensed is exactly what is seen in *vel-out*.

The *DiscreteController* HA has state variables *vel-reported* and *acc-suggested*, both discrete variables of type R, initially 0, a discrete Boolean state variable *stable*, initially true, and one external variable *acc-in*, of type R and dynamic type equal to (the pasting closure of) the continuous functions. The state consists of all valuations of the internal variables. The *DiscreteController* also has external actions $report(v)$, $v \in \mathsf{R}$, and an internal action *suggest*. D includes $report(v)$ steps that satisfy:

$$\begin{aligned} vel\text{-}reported' &= v && (23)\\ stable' &= \mathsf{false} && (24)\end{aligned}$$

and *suggest* steps that satisfy:

$$\begin{aligned} stable &= \mathsf{false} && (25)\\ stable' &= \mathsf{true} && (26)\\ vel\text{-}reported + (acc\text{-}suggested' + \epsilon)\mathsf{d} &\leq \mathsf{vmax} && (27)\end{aligned}$$

That is, a new velocity report sets the flag that triggers the *DiscreteController* to recalculate the suggested acceleration. Trajectories satisfy:

$$\begin{aligned} stable(t) &= stable(0) && (28)\\ stable(t) &= \mathsf{true} \qquad \text{for } t > 0 && (29)\\ \dot{acc\text{-}suggested} &= 0 && (30)\\ acc\text{-}in &= acc\text{-}suggested && (31)\end{aligned}$$

That is, the *DiscreteController* does not allow time to pass if $stable = \mathsf{false}$; it must perform a *suggest* action after receiving a *report* input and before time can pass. The *DiscreteController* does not change the suggested acceleration during a trajectory, and submits it accurately to its environment. Now define

$$\mathcal{A} = \mathsf{ActHide}(\{report(v) \mid v \in \mathsf{R}\}, Sensor \| DiscreteController).$$

We claim that $\mathcal{A}$ implements $\mathcal{B} \triangleq Controller$. We may argue this using the simulation relation R that relates states $\mathbf{x}_{\mathcal{A}}$ of $\mathcal{A}$ and $\mathbf{x}_{\mathcal{B}}$ of $Controller$ provided that $\mathbf{x}_{\mathcal{A}}$ is a reachable state of $\mathcal{A}$, $\mathbf{x}_{\mathcal{B}}(vel\text{-}sensed) = \mathbf{x}_{\mathcal{A}}(vel\text{-}sensed)$, $\mathbf{x}_{\mathcal{B}}(acc\text{-}suggested) = \mathbf{x}_{A}(acc\text{-}suggested)$ and $\mathbf{x}_{\mathcal{B}}(clock) = \mathbf{x}_{A}(clock)$ if $\mathbf{x}_{A}(stable) = \mathsf{true}$, else d. A key to the argument is that a *suggest* step occurs in $\mathcal{B}$ when *suggest* occurs in $\mathcal{A}$, rather than when a *report* occurs.

Since $\mathcal{A} \leq Controller$, Theorem 5.8 implies $\mathcal{A} \| Vehicle \leq Controller \| Vehicle$. Then Theorem 5.12 implies

$$\mathsf{VarHide}(\{acc\text{-}in\}, \mathcal{A} \| Vehicle) \leq \mathsf{VarHide}(\{acc\text{-}in\}, Controller \| Vehicle).$$

Since, by Example 5.13, $\mathsf{VarHide}(\{acc\text{-}in\}, Controller \| Vehicle) \leq VSpec$, transitivity of implementation implies that $\mathsf{VarHide}(\{acc\text{-}in\}, \mathcal{A} \| Vehicle)$ implements $VSpec$. ■

6 Hybrid I/O Automata

In this section we refine the hybrid automaton model of Section 4 by distinguishing between input and output actions and between input and output variables. The results on simulation relations and operations for hybrid automata presented in Sections 4.3 and 5 can be extended to this new setting.

6.1 Definition of Hybrid I/O Automata

Definition 6.1 *A* hybrid I/O automaton (HIOA) $\mathcal{A}$ *is a tuple* $(\mathcal{H}, U, Y, I, O)$ *where*

- $\mathcal{H} = (W, X, Q, \Theta, E, H, D, \mathcal{T})$ *is a hybrid automaton.*
- U *and* Y *partition* W *into* input *and* output *variables, respectively.*
  Variables in $Z \triangleq X \cup Y$ *are called* locally controlled*; as before, we write* $V \triangleq W \cup X$.
- I *and* O *partition* E *into* input *and* output *actions, respectively.*
  Actions in $L \triangleq H \cup O$ *are called* locally controlled*; as before we write* $A \triangleq E \cup H$.
- *The following additional axioms are satisfied:*
  E1 *(Input action enabling)*
  For every $\mathbf{x} \in Q$ *and every* $a \in I$*, there exists* $\mathbf{x}' \in Q$ *such that* $\mathbf{x} \xrightarrow{a} \mathbf{x}'$.

E2 *(Input trajectory enabling)*
For every $\mathbf{x} \in Q$ *and every* $\upsilon \in trajs(U)$*, there exists* $\tau \in \mathcal{T}$ *such that* $\tau.fstate = \mathbf{x}$*,* $\tau \downarrow U \leq \upsilon$*, and either*
(1) $\tau \downarrow U = \upsilon$*, or*
(2) τ *is closed and some* $l \in L$ *is enabled in* $\tau.lstate$*.*

Input action enabling is the input enabling condition of ordinary I/O automata. Input trajectory enabling is a new, corresponding condition for interaction over time intervals. It says that an HIOA should be able to accept any input trajectory, that is, any trajectory for the input variables, either by letting time advance for the entire duration of the input trajectory, or by reacting with a locally controlled action after some part of the input trajectory has occurred. In Section 7, we will see that by repeated application of axiom **E2** a HIOA is able to fully accept any input trajectory, possibly interleaved with locally controlled actions, provided the HIOA does not exhibit unwanted Zeno behavior.

Note the role of dynamic types in axiom **E2**. Input trajectory enabling means that an automaton cannot restrict the inputs. The problem we hit is that with absolutely no way of restricting the inputs, the inputs were just too ill-behaved. In examples, we typically want to be able to integrate the input to get the value of internal variables, but we cannot do this unless the input is integrable. Axiom **E2** states that a HIOA needs to be able to accept any input trajectory in $trajs(U)$. By definition, the trajectories in $trajs(U)$, when projected on an individual variable $u \in U$, must be in agreement with the dynamic type of u. For instance, by taking as the dynamic type of variables in U the set of piecewise smooth functions, we impose some rather minimal constraints on the input trajectories that allow us to give meaningful automaton definitions involving integrals, differential equations, etc.

In control theory it is customary to require *causality*, that is, the output at time t depends only upon the input trajectory up to, and possibly including, time t [71]. In our setting, there is no need to enforce causality explicitly since it is implied already by the closure of the set of trajectories under prefix and concatenation. Assume that in a trajectory τ the output at time t "depends" on the input trajectory after t. By prefix closure of trajectories (axiom **T1**), $\tau \trianglelefteq t$ is also a trajectory. Let $\mathbf{x}$ be the state of τ at time t, and let υ be any input trajectory. By axiom **E2** there exists a trajectory τ' with first state $\mathbf{x}$ that agrees with υ (at least up to a certain point). By axiom **T3** the concatenation of $\tau \trianglelefteq t$ and τ' is again a trajectory. The output of this trajectory at time t agrees with the output of τ at time t, even though the subsequent inputs will in general be different. It follows that in τ the output at time t does not depend on the input after t, a contradiction. Also note that our definition does not enforce functional dependence of outputs from inputs: HIOAs may be nondeterministic, allowing for several possible outputs for any given input trajectory.

It will sometimes be convenient for us to consider automata in which inputs and outputs are distinguished, but that do not necessarily satisfy the properties **E1** or **E2**. We call such an automaton a *pre-HIOA*.

Notation: As we did for HAs, we denote the components of a (pre-)HIOA $\mathcal{A}$ by $\mathcal{H}_\mathcal{A}, U_\mathcal{A}, Y_A, \ldots, W_\mathcal{A}, X_\mathcal{A}, Q_A, \Theta_\mathcal{A}$, etc., and those of a (pre-)HIOA $\mathcal{A}_i$ by $H_i, U_i, Y_i, \ldots, W_i, X_i$, Q_i, Θ_i, etc. We sometimes omit these subscripts, where no confusion is likely. We abuse notation slightly by referring to a (pre-)HIOA $\mathcal{A}$ as an HA when we intend to refer to $\mathcal{H}_A$.

Example 6.2 (Vehicle and controller HIOAs) The *Vehicle* HA of Example 4.2 can be converted into an HIOA by classifying *acc-in* as an input variable and *vel-out* as an output variable. Property **E1**, input action enabling, holds vacuously. It is also easy to see that **E2** holds, in fact, the first alternative always holds—from any state the *Vehicle* automaton can accept any input trajectory. Note that, in order for **E2** to hold, it is essential that we do not require inclusion (2) to hold for initial states of trajectories.

Similarly, the *Controller* HA of Example 4.3 can be converted into an HIOA by classifying *vel-out* as an input variable and *acc-in* as an output variable. Again, **E1** holds vacuously. To see **E2**, consider a state $\mathbf{x}$, and an input trajectory v. The definition of Q implies that $\mathbf{x}(\mathit{clock}) \leq \mathsf{d}$. Then the definition of the *Controller* trajectories implies that there is some trajectory τ starting from $\mathbf{x}$ that is consistent with v and that either spans all of v or stops short, at a valuation $\mathbf{v}$ in which $\mathit{clock} = \mathsf{d}$. Then the definition of the *suggest* transitions implies that this locally controlled action is enabled in $\mathbf{v} \lceil X$, as needed. ■

Example 6.3 (Sensor and discrete controller HIOAs) The *Sensor* automaton from Example 5.14 can be converted into an HIOA by classifying *vel-out* as an input variable and the *report* actions as output actions. The argument that *Sensor* is actually an HIOA is similar to the argument for the *Controller* in Example 6.2.

Similarly, the *DiscreteController* automaton from Example 5.14 can be converted into an HIOA by classifying the *report* actions as input actions and the *acc-in* variable as an output variable. It is straightforward to verify **E1**. **E2** is not completely trivial, even though the automaton has no input variables: from any state $\mathbf{x}$ we must consider "null" input trajectories, which map a time interval to the empty valuation (the valuation for no variables). If $\mathbf{x}(\mathit{stable}) = \mathsf{true}$, then the *DiscreteController* can accept the entire input trajectory, and if $\mathbf{x}(\mathit{stable}) = \mathsf{false}$, then *suggest* is enabled in $\mathbf{x}$. This implies **E2**. ■

An *execution* of a pre-HIOA $\mathcal{A}$ is defined to be an execution of $\mathcal{H}_{\mathcal{A}}$, a *trace* of $\mathcal{A}$ is a trace of $\mathcal{H}_{\mathcal{A}}$, and similarly for execution fragments and trace fragments. We extend the notation $execs_{\mathcal{A}}$, etc. to pre-HIOAs in the obvious way. Two pre-HIOAs $\mathcal{A}_1$ and $\mathcal{A}_2$ are *comparable* if their inputs and outputs coincide, that is, if $I_1 = I_2$, $O_1 = O_2$, $U_1 = U_2$, and $Y_1 = Y_2$. If $\mathcal{A}_1$ and $\mathcal{A}_2$ are comparable, then $\mathcal{A}_1 \leq \mathcal{A}_2$ is defined to mean that the traces of $\mathcal{A}_1$ are included among those of $\mathcal{A}_2$: $\mathcal{A}_1 \leq \mathcal{A}_2 \triangleq traces_{\mathcal{A}_1} \subseteq traces_{\mathcal{A}_2}$.

Lemma 6.4 *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two comparable pre-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. ■

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

Theorem 6.5 *If $\mathcal{A}_1$ and $\mathcal{A}_2$ are comparable pre-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 4.12, and Lemma 6.4. ■

6.3 Composition

The definition of composition for HIOAs is based on the corresponding definition for HAs, but also takes the input/output structure into account. Just as for HAs, we allow an HIOA $\mathcal{A}_1$ to be composed with an HIOA $\mathcal{A}_2$ only if the sets of internal actions and variables of $\mathcal{A}_1$ are disjoint from the sets of actions and variables, respectively, of $\mathcal{A}_2$. In addition, in order that the composition operation might satisfy certain desirable properties (see, for example, the results in Section 6.5), we require that at most one component should "control" any given action or variable; that is, we allow $\mathcal{A}_1$ and $\mathcal{A}_2$ to be composed only if the sets of output actions of $\mathcal{A}_1$ and $\mathcal{A}_2$ are disjoint and the sets of output variables of $\mathcal{A}_1$ and $\mathcal{A}_2$ are disjoint.

Formally, we say that pre-HIOAs $\mathcal{A}_1$ and $\mathcal{A}_2$ are *compatible* if $\mathcal{H}_1$ and $\mathcal{H}_2$ are compatible and

$$Y_1 \cap Y_2 = O_1 \cap O_2 = \emptyset.$$

Lemma 6.6 *If $\mathcal{A}_1$ and $\mathcal{A}_2$ are compatible pre-HIOAs, then $\mathcal{H}_1$ and $\mathcal{H}_2$ are compatible HAs.*

Proof: Immediate from the definitions. ■

If $\mathcal{A}_1$ and $\mathcal{A}_2$ are compatible pre-HIOAs then their *composition* $\mathcal{A}_1 \| \mathcal{A}_2$ is defined to be the tuple $\mathcal{A} = (\mathcal{H}, U, Y, I, O)$ where

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

Thus, an external action or variable of the composition is classified as an output if it is an output of one of the component automata, and otherwise it is classified as an input.

The composition of two HIOAs (or pre-HIOAs) is guaranteed to be a pre-HIOA:

Theorem 6.7 *If $\mathcal{A}_1$ and $\mathcal{A}_2$ are pre-HIOAs then $\mathcal{A}_1 \| \mathcal{A}_2$ is a pre-HIOA.*

Proof: Let $\mathcal{A}$ denote $\mathcal{A}_1 \| \mathcal{A}_2$. Lemma 5.2 implies that $\mathcal{H} = \mathcal{H}_1 \| \mathcal{H}_2$ is an HA. By construction, U and Y form a partition of W and I and O form a partition of E. This suffices. ■

Example 6.8 (Interfaces for compositions of HIOAs) When the *Vehicle* and *Controller* HIOAs from Example 6.2 are composed, the external interface of the resulting pre-HIOA consists of $U = I = O = \emptyset$ and $Y = \{\mathit{acc\text{-}in}, \mathit{vel\text{-}out}\}$. When the *Sensor* and *DiscreteController* from Example 6.3 are composed, the external interface of the resulting pre-HIOA consists of $U = \{\mathit{vel\text{-}out}\}$, $Y = \{\mathit{acc\text{-}in}\}$, $I = \emptyset$, and $O = \{\mathit{report}(v) \mid v \in \mathsf{R}\}$. ■

Composition of pre-HIOAs satisfies the following substitutivity result:

Theorem 6.9 *Suppose $\mathcal{A}_1$ and $\mathcal{A}_2$ are comparable pre-HIOAs with $\mathcal{A}_1 \leq \mathcal{A}_2$. Suppose $\mathcal{B}$ is a pre-HIOA that is compatible with each of $\mathcal{A}_1$ and $\mathcal{A}_2$. Then $\mathcal{A}_1 \| B$ and $\mathcal{A}_2 \| B$ are comparable and $\mathcal{A}_1 \| \mathcal{B} \leq \mathcal{A}_2 \| \mathcal{B}$.*

Proof: The fact that $\mathcal{A}_1$ and $\mathcal{A}_2$ are comparable and the definition of composition for pre-HIOAs implies that $\mathcal{A}_1 \| \mathcal{B}$ and $\mathcal{A}_2 \| \mathcal{B}$ are comparable.

Since $\mathcal{A}_1$ and $\mathcal{A}_2$ are comparable and $\mathcal{A}_1 \leq \mathcal{A}_2$, Lemma 6.4 implies that $\mathcal{H}_{\mathcal{A}_1}$

and $\mathcal{H}_{\mathcal{A}_2}$ are comparable and $\mathcal{H}_{\mathcal{A}_1} \leq \mathcal{H}_{\mathcal{A}_2}$. Lemma 6.6 implies that $\mathcal{H}_{\mathcal{A}_1}$ and $\mathcal{H}_{\mathcal{B}}$ are compatible HAs and $\mathcal{H}_{\mathcal{A}_2}$ and $\mathcal{H}_{\mathcal{B}}$ are compatible HAs. Theorem 5.8 then implies that $\mathcal{H}_{\mathcal{A}_1} \| \mathcal{H}_{\mathcal{B}} \leq \mathcal{H}_{\mathcal{A}_2} \| \mathcal{H}_{\mathcal{B}}$. By the definition of composition, it follows that $\mathcal{H}_{\mathcal{A}_1 \| \mathcal{B}} \leq \mathcal{H}_{\mathcal{A}_2 \| \mathcal{B}}$. Then the definition of implementation for pre-HIOAs implies that $\mathcal{A}_1 \| \mathcal{B} \leq \mathcal{A}_2 \| \mathcal{B}$. ■

We would like to show that the composition of two HIOAs is an HIOA; however, this is not true in general. Property **E1** is preserved by composition:

Lemma 6.10 *If $\mathcal{A}_1$ and $\mathcal{A}_2$ are pre-HIOAs that satisfy* **E1***, then the composition $\mathcal{A}_1 \| A_2$ also satisfies* **E1***.*

Proof: Let $\mathcal{A} = \mathcal{A}_1 \| A_2$. Assume that $\mathcal{A}_1$ and A_2 satisfy **E1**. We verify that $\mathcal{A}$ satisfies **E1**. Consider $\mathbf{x} \in Q$ and $a \in I$. We distinguish three cases.

(1) $a \in I_1 \cap I_2$. By definition of composition, $\mathbf{x} \lceil X_i \in Q_i$ for $i \in \{1, 2\}$. Then by **E1** applied to $\mathcal{A}_i$, there exists a state $\mathbf{x}'_i$ of $\mathcal{A}_i$ such that $(\mathbf{x} \lceil X_i) \xrightarrow{a}_i \mathbf{x}'_i$. Let $\mathbf{x}' \triangleq \mathbf{x}'_1 \cup \mathbf{x}'_2$. We know that $\mathbf{x}'$ is well defined since, by compatibility, $X_1 \cap X_2 = \emptyset$. Then by definition of composition, $\mathbf{x}' \in Q$ and $\mathbf{x} \xrightarrow{a} \mathbf{x}'$.
(2) $a \in I_1 - I_2$. By definition of composition, $\mathbf{x} \lceil X_1 \in Q_1$. By **E1** applied to $\mathcal{A}_1$, there exists a state $\mathbf{x}'_1$ of $\mathcal{A}_1$ such that $(\mathbf{x} \lceil X_1) \xrightarrow{a}_1 \mathbf{x}'_1$. Let $\mathbf{x}' \triangleq \mathbf{x}'_1 \cup (\mathbf{x} \lceil X_2)$. We know that $\mathbf{x}'$ is well defined since, by compatibility, $X_1 \cap X_2 = \emptyset$. Then by definition of parallel composition, $\mathbf{x}' \in Q$ and $\mathbf{x} \xrightarrow{a} \mathbf{x}'$.
(3) $a \in I_2 - I_1$. Symmetric to the previous case. ■

However, **E2** is not necessarily preserved by composition:

Example 6.11 (Two HIOAs whose composition does not satisfy E2) Suppose that $\mathcal{A}_1$ has no discrete actions, no state variables, one output variable v_1 and one input variable v_2. All variables are of type R and dynamic type the (pasting closure of the) continuous functions. The sets Q_1 and Θ_1 of states and start states consist of the unique valuation of the empty set of variables. The trajectories are all those functions that satisfy $v_1(t) = v_2(t) + 1$ for $t > 0$. It is easy to check that $\mathcal{A}_1$ is an HIOA. Define $\mathcal{A}_2$ symmetrically, with output variable v_2 and input variable v_1; $\mathcal{A}_2$'s trajectories are those that satisfy $v_2(t) = v_1(t) + 1$ for $t > 0$.

The composition pre-HIOA, $\mathcal{A}_1 \| \mathcal{A}_2$, does not satisfy **E2**. Satisfying **E2** would require (since the composition has no discrete actions) that the composition include at least one trajectory with limit time ∞ starting from the initial state. However, no such trajectory exists, because the combined constraints are inconsistent for every $t > 0$. ■

As a way out of the difficulties noted in Example 6.11, we might consider introducing a static *dependency relation* $\prec_{\mathcal{A}}$ between the external variables of a hybrid automaton. If $x \prec_{\mathcal{A}} y$ then the value of y is allowed to depend without delay on the value of x. As an additional condition for compatibility of $\mathcal{A}$ and $\mathcal{B}$, we would then require that $\mathcal{A}$ and $\mathcal{B}$ do not share variables x and y such that $x \prec \mathcal{A} y$ and $y \prec_{\mathcal{B}} x$. This approach, which is followed, for example, in the Masaccio language of [33], would rule out the above example. However, it would also rule out any form of dynamic feedback as studied in control theory (for instance, PID control) [79]. We therefore think that this static approach is overly restrictive. Within control theory there is no generally applicable syntactic criterion to test whether combinations of differential and algebraic equations are well-defined; consequently, we have no simple criterion to test whether the composition of two HIOAs satisfies **E2**.

As a technical way out of the difficulty, we define a stronger notion of compatibility. Namely, we say that compatible pre-HIOAs $\mathcal{A}_1$ and $\mathcal{A}_2$ are *strongly compatible* if $\mathcal{A}_1 \| \mathcal{A}_2$ satisfies axiom **E2**. Strong compatibility says that any input trajectory υ of the composition must be acceptable by the composition: the two component automata are able to evolve together, following the input trajectory υ, in such a way that either they accept all of υ or else they accept part of υ, up to a point where one of them can interrupt with a locally controlled action.

Theorem 6.12 *If $\mathcal{A}_1$ and $\mathcal{A}_2$ are strongly compatible HIOAs, then $\mathcal{A}_1 \| \mathcal{A}_2$ is an HIOA.*

Proof: Lemma 6.7 implies that the composition is a pre-HIOA. Lemma 6.10 implies that the composition satisfies **E1**. Property **E2** follows immediately from strong compatibility. ■

Strong compatibility is a technical notion. By itself, it does not seem to be very useful, because checking it involves verifying compatibility between the continuous dynamics of two systems. In Section 6.5, we give some sufficient conditions for strong compatibility that are easier to check.

6.4 Hiding

The definitions of variable and action hiding extend to any pre-HIOA $\mathcal{A}$. For input/output automata, we allow hiding outputs only (but not inputs):

(1) If $O \subseteq O_{\mathcal{A}}$, then $\mathsf{ActHide}(O, \mathcal{A})$ is the pre-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 pre-HIOA $\mathcal{B}$ given by:

- $\mathcal{H}_{\mathcal{B}} = \mathsf{VarHide}(Y, \mathcal{H}_{\mathcal{A}})$.
- $Y_{\mathcal{B}} = Y_{\mathcal{A}} - Y$.
- $U_{\mathcal{B}} = U_{\mathcal{A}}$, $I_{\mathcal{B}} = I_{\mathcal{A}}$, and $O_{\mathcal{B}} = O_{\mathcal{A}}$.

Lemma 6.13 *Suppose $\mathcal{A}$ is a pre-HIOA, $O \subseteq O_{\mathcal{A}}$ and $Y \subseteq Y_{\mathcal{A}}$. Then:*

(1) $\mathsf{ActHide}(O, \mathcal{A})$ *and* $\mathsf{VarHide}(Y, \mathcal{A})$ *are pre-HIOAs.*
(2) If $\mathcal{A}$ satisfies **E1** *then so do* $\mathsf{ActHide}(O, \mathcal{A})$ *and* $\mathsf{VarHide}(Y, \mathcal{A})$.
(3) If $\mathcal{A}$ satisfies **E2** *then so do* $\mathsf{ActHide}(O, \mathcal{A})$ *and* $\mathsf{VarHide}(Y, \mathcal{A})$.

Lemma 6.14 *Let $\mathcal{A}$ be a pre-HIOA.*

(1) If $O \subseteq O_{\mathcal{A}}$ then $traces_{\mathsf{ActHide}(O,\mathcal{A})} = \{\beta \lceil (E_{\mathcal{A}} - O, V_{\mathcal{A}}) \mid \beta \in traces_{\mathcal{A}}\}$.
(2) If $Y \subseteq Y_{\mathcal{A}}$ then $traces_{\mathsf{VarHide}(Y,\mathcal{A})} = \{\beta \lceil (A_{\mathcal{A}}, W_{\mathcal{A}} - Y) \mid \beta \in traces_{\mathcal{A}}\}$.

Proof: Straightforward, see also the proof of Lemma 5.11. ∎

Theorem 6.15 *Suppose $\mathcal{A}$ and $\mathcal{B}$ are pre-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* $\mathsf{VarHide}(Y, \mathcal{A}) \leq \mathsf{VarHide}(Y, \mathcal{B})$.

Proof: Straightforward, using Lemma 6.14. ∎

Example 6.16 (Interfaces for automata with hiding) In Example 5.14, we defined the HA $\mathcal{B} \triangleq \mathsf{VarHide}(\{\mathit{acc\text{-}in}\}, \mathcal{A} \| \mathit{Vehicle})$, where

$$\mathcal{A} \triangleq \mathsf{ActHide}(\{\mathit{report}(v) \mid v \in \mathsf{R}\}, \mathit{Sensor} \| \mathit{DiscreteController}).$$

This models the three-way composition of the sensor, discrete controller, and vehicle, with the internal report actions and acceleration suggestions hidden. If we interpret the three automata as HIOAs, then these definitions still make sense because the actions and variables that are hidden are outputs. The external interface for $\mathcal{A}$ is given by $U_{\mathcal{A}} = \{\mathit{vel\text{-}out}\}$, $Y_{\mathcal{A}} = \{\mathit{acc\text{-}in}\}$, and $I_{\mathcal{A}} = O_{\mathcal{A}} = \emptyset$, and the external interface for $\mathcal{B}$ is given by $U_{\mathcal{B}} = I_{\mathcal{B}} = O_{\mathcal{B}} = \emptyset$ and $Y_{\mathcal{B}} = \{\mathit{vel\text{-}out}\}$. ∎

6.5 Sufficient Conditions for Strong Compatibility

Checking strong compatibility of two HIOAs can be difficult because it requires checking compatibility between the continuous dynamics of two systems. However, for certain restricted classes of HIOAs, strong compatibility is implied by compatibility, which is easy to check.

Example 6.17 (HIOAs for which compatibility implies strong compatibility) It is routine to verify that two HIOAs without input variables are strongly compatible if and only if they are compatible. In the classical control theory setting, a system without input variables is uninteresting because it cannot be controlled. However, in the hybrid setting, such a system can still interact with its environment via discrete input actions. *Linear hybrid automata* as described in [4,3], for instance, have no input variables.

Symmetrically, two HIOAs without output variables are strongly compatible if and only if they are compatible. The same equivalence holds if one of the HIOAs has no input variables and the other has no output variables, or if one has no external variables at all. ■

The following theorem generalizes all the claims in Example 6.17. It applies to pairs of HIOAs that cannot mutually affect each other because the output variables of one are disjoint from the input variables of the other.

Theorem 6.18 *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two compatible HIOAs such that $U_1 \cap Y_2 = \emptyset$. Then $\mathcal{A}_1$ and $\mathcal{A}_2$ are strongly compatible.*

Proof: Let $\mathcal{A}$ denote $\mathcal{A}_1 \| \mathcal{A}_2$. We need to show that $\mathcal{A}$ satisfies **E2**. Let $\mathbf{x}$ be a state of $\mathcal{A}$ and let υ be a trajectory in $trajs(U)$. Since $U_1 \cap Y_2 = \emptyset$, the definition of composition implies that $U_1 \subseteq U$. By **E2** applied to $\mathcal{A}_1$, there exists a trajectory $\tau_1 \in \mathcal{T}_1$, with $\tau_1.fstate = \mathbf{x} \lceil X_1$ that is pointwise compatible with υ and such that either $dom(\tau_1) = dom(\upsilon)$, or else $dom(\tau_1) \subset dom(\upsilon)$, τ_1 is closed, and a locally controlled action of $\mathcal{A}_1$ is enabled in $\tau_1.lstate$.

Let υ_2 be $((\upsilon \lceil dom(\tau_1)) \ \dot{\cup} \ \tau_1) \downarrow U_2$. That is, υ_2 is an input trajectory for $\mathcal{A}_2$. Each input variable of $\mathcal{A}_2$ is either an input variable of $\mathcal{A}$ or an output variable of $\mathcal{A}_1$; the valuations in υ_2 for those that are inputs of $\mathcal{A}$ are obtained from υ, whereas the valuations for those that are output variables of $\mathcal{A}_1$ are obtained from τ_1. By **E2** applied to $\mathcal{A}_2$, there exists a trajectory $\tau_2 \in \mathcal{T}_2$, with $\tau_2.fstate = \mathbf{x} \lceil X_2$, that is pointwise compatible with υ_2 and such that either $dom(\tau_2) = dom(\upsilon_2)$, or else $dom(\tau_2) \subset dom(\upsilon_2)$, τ_2 is closed, and a locally controlled action of $\mathcal{A}_2$ is enabled in $\tau_2.lstate$.

In the second case, $(\tau_1 \lceil dom(\tau_2)) \ \dot{\cup} \ \tau_2$ is a trajectory of $\mathcal{T}$ that starts from $\mathbf{x}$, is pointwise compatible with υ, is closed, and enables a locally controlled action of $\mathcal{A}$ (in particular, of $\mathcal{A}_2$) in its last state. In the first case, $\tau_1 \ \dot{\cup} \ \tau_2$ is a trajectory of $\mathcal{T}$ that starts from $\mathbf{x}$, is pointwise compatible with υ, and either spans all of υ or is closed and enables a locally controlled action of $\mathcal{A}$

(in particular, of $\mathcal{A}_1$) in its last state. This shows that $\mathcal{A}$ satisfies **E2**. ■

We can also consider HIOAs that do not exhibit any dependencies between inputs and outputs during a trajectory. In particular, the values of the input variables should affect neither the values of the output variables nor the amount of time that elapses until a locally controlled action is enabled. Formally, we say that an HIOA $\mathcal{A}$ is *oblivious* if it satisfies the following axiom:

OBL For all $\tau \in \mathcal{T}$ and $\upsilon \in \mathit{trajs}(U)$ with $\mathit{dom}(\tau) = \mathit{dom}(\upsilon)$, there exists $\tau' \in \mathcal{T}$ such that:
(1) $\tau' \downarrow U = \upsilon$.
(2) $\tau' \downarrow Y = \tau \downarrow Y$.
(3) If τ is closed and some locally controlled action is enabled in $\tau.\mathit{lstate}$ then some locally controlled action is enabled in $\tau'.\mathit{lstate}$.

Theorem 6.19 *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two compatible HIOAs and suppose that $\mathcal{A}_1$ is oblivious. Then $\mathcal{A}_1$ and $\mathcal{A}_2$ are strongly compatible.*

Proof: Let $\mathcal{A}$ denote $\mathcal{A}_1 \| \mathcal{A}_2$. We need to show that $\mathcal{A}$ satisfies **E2**. Let $\mathbf{x}$ be a state of $\mathcal{A}$ and let υ be a trajectory in $\mathit{trajs}(U)$. Let υ_1 be any trajectory of $\mathit{trajs}(U_1)$ that is pointwise compatible with υ and such that $\mathit{dom}(\upsilon_1) = \mathit{dom}(\upsilon)$. By **E2** applied to $\mathcal{A}_1$, there exists a trajectory $\tau_1 \in \mathcal{T}_1$, with $\tau_1.\mathit{fstate} = \mathbf{x} \lceil X_1$, that is pointwise compatible with υ_1 and such that either $\mathit{dom}(\tau_1) = \mathit{dom}(\upsilon_1)$, or else $\mathit{dom}(\tau_1) \subset \mathit{dom}(\upsilon_1)$, τ_1 is closed, and a locally controlled action of $\mathcal{A}_1$ is enabled in $\tau_1.\mathit{lstate}$.

Let υ_2 be $((\upsilon \lceil \mathit{dom}(\tau_1)) \ \dot{\cup} \ \tau_1) \downarrow U_2$. By **E2** applied to $\mathcal{A}_2$, there exists a trajectory $\tau_2 \in \mathcal{T}_2$, with $\tau_2.\mathit{fstate} = \mathbf{x} \lceil X_2$, that is pointwise compatible with υ_2 and such that either $\mathit{dom}(\tau_2) = \mathit{dom}(\upsilon_2)$, or else $\mathit{dom}(\tau_2) \subset \mathit{dom}(\upsilon_2)$, τ_2 is closed, and a locally controlled action of $\mathcal{A}_2$ is enabled in $\tau_2.\mathit{lstate}$.

Let υ_1' be $((\upsilon \lceil \mathit{dom}(\tau_2)) \ \dot{\cup} \ \tau_2) \downarrow U_1$. By **OBL** applied to $\mathcal{A}_1$, there exists a trajectory $\tau_1' \in \mathcal{T}_1$ such that $\tau_1' \downarrow U_1 = \upsilon_1'$, $\tau_1' \downarrow Y_1 = (\tau_1 \lceil \mathit{dom}(\tau_2)) \downarrow Y_1$, and if $\tau_1 \lceil \mathit{dom}(\tau_2)$ is closed and some locally controlled action of $\mathcal{A}_1$ is enabled in its last state, then some locally controlled action is also enabled in $\tau_1'.\mathit{lstate}$. It follows that τ_1' and τ_2 are pointwise compatible, and that $\tau_1' \dot{\cup} \tau_2$ is a trajectory in $\mathcal{T}$ that starts from $\mathbf{x}$ and is pointwise compatible with υ. We claim that $\tau_1' \ \dot{\cup} \ \tau_2$ satisfies the requirements for **E2**. We consider cases:

(1) $\mathit{dom}(\tau_2) \subset \mathit{dom}(\upsilon_2)$.
Then $\tau_1' \ \dot{\cup} \ \tau_2$ is closed and enables a locally controlled action (of $\mathcal{A}_2$) in its last state, which satisfies the requirements for **E2**.

(2) $\mathit{dom}(\tau_2) = \mathit{dom}(\upsilon_2)(= \mathit{dom}(\tau_1))$.
We consider two subcases. First, if $\mathit{dom}(\tau_1) \subset \mathit{dom}(\upsilon)$, then τ_1 is closed and enables some locally controlled action (of $\mathcal{A}_1$) in its last state. By

axiom **OBL**, some locally controlled action is also enabled in $\tau_1' \dot\cup \tau_2.lstate$, which suffices for **E2**. In the other subcase, if $dom(\tau_1) = dom(v)$, then $\tau_1' \dot\cup \tau_2$ spans all of v, which again suffices for **E2**.

■

Example 6.20 (Oblivious controller) The *Controller* HIOA of Example 4.3 and 6.2 satisfies **OBL**. During any trajectory τ of *Controller*, velocity information arrives in *vel-out* but does not affect the *Controller*'s output; the output is only changed when a (locally controlled) *suggest* transition occurs. Enabling of the *suggest* action is not affected by changes in *vel-out*, but only by the value of *clock*.

Because *Controller* is oblivious and compatible with the *Vehicle* HIOA, Theorem 6.19 implies that *Vehicle* and *Controller* are strongly compatible. It follows that their composition, *Vehicle*$\|$*Controller*, is an HIOA. ■

Example 6.21 (Plant and controller) Figure 8 displays a standard scenario studied in control theory involving a plant $\mathcal{P}$ controlled by a digital controller $\mathcal{C}$. The interface from the controller to the plant is given by a digital/analog

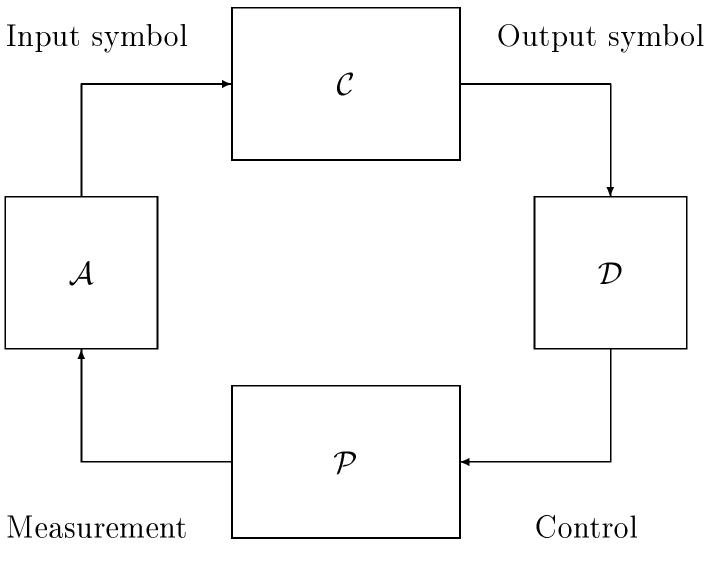

Fig. 8. Hybrid Control System.

converter $\mathcal{D}$, while the interface from the plant to the controller is given by an analog/digital converter $\mathcal{A}$. The controller $\mathcal{C}$ monitors the input variables and changes its output variables only at the clock ticks via some discrete transitions. Thus, $\mathcal{C}$ satisfies **OBL**. The output variables of $\mathcal{A}$ are disjoint from the input variables of both $\mathcal{P}$ and $\mathcal{D}$, and the output variables of $\mathcal{P}$

are disjoint from the input variables of $\mathcal{D}$. Thus, if $\mathcal{P}, \mathcal{C}, \mathcal{A}, \mathcal{D}$ are pairwise compatible, then $\mathcal{P}$ and $\mathcal{A}$ are strongly compatible (by Theorem 6.18), $P \| A$ and $\mathcal{D}$ are strongly compatible (by Theorem 6.18), and $((\mathcal{P} \| \mathcal{A}) \| \mathcal{D})$ and $\mathcal{C}$ are strongly compatible (by Theorem 6.19). Hence, $((\mathcal{P} \| \mathcal{A}) \| \mathcal{D}) \| \mathcal{C}$ is an HIOA. ■

Example 6.22 (Lipschitz HIOAs) We may define a subclass of HIOAs called *Lipschitz HIOAs*, in which some of the state variables are discrete "mode" variables, and in which, for each mode, the rest of the variables evolve according to a system of differential equations based on globally Lipschitz functions. We may restrict this class further by imposing a bound on the range of the input variables (by restricting their dynamic types), thus obtaining the set of *input-bounded Lipschitz HIOAs*. Then it is possible to show that two compatible input-bounded Lipschitz HIOAs are strongly compatible, which implies that the composition of two compatible input-bounded Lipschitz HIOAs is a (Lipschitz) HIOA. A careful development will be reserved for another paper. ■

7 Receptive Hybrid I/O Automata

In this section, we define the notion of *receptiveness* for HIOAs. An HIOA will be defined to be receptive provided that it admits a *strategy* for resolving its nondeterministic choices that never generates infinitely many locally controlled actions in finite time. This notion has two important consequences: First, a receptive HIOA provides some response from any state, for any sequence of discrete input actions and input trajectories. This implies that the automaton has a nontrivial set of execution fragments, in fact, it has execution fragments that accommodate any inputs from the environment. The automaton cannot simply stop at some point and refuse to allow time to elapse; it must allow time to pass to infinity if the environment does so. Second, receptiveness is closed under composition. Previous studies of receptiveness properties include [21,1,74,54].

If HIOA $\mathcal{A}$ implements HIOA $\mathcal{B}$ and if $\mathcal{A}$ is receptive, then besides preservation of "may" properties (any trace of $\mathcal{A}$ is also a trace of $\mathcal{B}$) we also have preservation of "must" properties. For instance, if in $\mathcal{B}$ an input action a always must be followed by an output b within 10 time units, then this property will also hold for $\mathcal{A}$: (1) since $\mathcal{A}$ is input enabled it will always accept input a, (2) since $\mathcal{A}$ is receptive it will never end up in a time deadlock or a Zeno execution; time can always advance, (3) $\mathcal{A}$ must always perform a b before or at time 10 since otherwise a trace is generated that is not allowed by $\mathcal{B}$.

We formally define receptiveness by first defining what it means for an HIOA to be *progressive*. A progressive HIOA *never* generates infinitely many locally controlled actions in finite time. Thus, in all of its execution fragments, it allows time to pass to infinity provided that its environment also does so. We then define a *strategy* for resolving nondeterministic choices, and define receptiveness in terms of the existence of a progressive strategy.

The treatment of receptiveness in this paper is much simpler than that in previous papers. One reason is that we address only the generation of admissible executions here, rather than general liveness properties. Also, we formulate strategies as restricted automata, rather than introducing separate definitions based on two-player games.

7.1 Progressive HIOAs

We say that an execution fragment of a pre-HIOA is *locally-Zeno* if it is Zeno and contains infinitely many locally controlled actions, or equivalently, if it has finite limit time and contains infinitely many locally controlled actions. A pre-HIOA $\mathcal{A}$ is *progressive* if it has no locally-Zeno execution fragments.

The following lemma says that any progressive pre-HIOA that satisfies **E2**, and therefore any HIOA, is capable of following any input trajectory.

Lemma 7.1 *Let $\mathcal{A}$ be a progressive pre-HIOA that satisfies property* **E2**, *let* $\mathbf{x}$ *be a state of $\mathcal{A}$, and let $\upsilon \in trajs(U)$. Then there exists an execution fragment α of $\mathcal{A}$ such that $\alpha.fstate = \mathbf{x}$ and $\alpha \lceil (I, U) = \upsilon$. (Here υ denotes the hybrid sequence consisting of the single trajectory υ. Recall that we write a for a sequence consisting of just a.)*

Proof: We construct a finite or infinite sequence $\alpha_0, \alpha_1, \ldots$ of execution fragments of $\mathcal{A}$ such that:

(1) $\alpha_0.fstate = \mathbf{x}$.
(2) For every nonfinal index i, $\alpha_i.lstate = \alpha_{i+1}.fstate$.
(3) For every $i \geq 0$, $(\alpha_0 \frown \alpha_1 \frown \cdots \frown \alpha_i) \lceil (I, U) \leq \upsilon$.
(4) For every $i \geq 0$, either $(\alpha_0 \frown \alpha_1 \frown \cdots \frown \alpha_i) \lceil (I, U) = \upsilon$ or α_i includes a locally controlled action.

The construction is carried out recursively. To define α_0, we begin with state $\mathbf{x}$ and use **E2** either to span all of υ, or to span a prefix of υ and then perform a locally controlled action. For $i > 0$ (assuming that we have not already spanned all of υ), we define α_i by beginning with $\alpha_{i-1}.lstate$ and using **E2** either to span the entire suffix of υ starting from $\alpha_0 \frown \cdots \frown \alpha_{i-1}.ltime$, or to span a prefix of that suffix and then perform a locally controlled action.

Now we consider two cases:

(1) The construction ends after a finite number of stages, having spanned all of υ, say with α_k as the last execution fragment in the sequence.
   In this case, the concatenation $\alpha_0 \frown \alpha_1 \frown \cdots \frown \alpha_k$ satisfies the conditions of the lemma.
(2) The construction proceeds through infinitely many stages.
   In this case, the execution fragment $\alpha \triangleq \alpha_0 \frown \alpha_1 \frown \cdots$ contains infinitely many locally controlled actions. Since $\mathcal{A}$ is progressive, it must be the case that $\alpha.\mathit{ltime} = \infty$, and therefore $\alpha \lceil (I, U).\mathit{ltime} = \infty$. Since the set of trajectories for U is a cpo, $\alpha \lceil (I, U) \leq \upsilon$. Since $\alpha \lceil (I, U) \leq \upsilon$, and $\alpha \lceil (I, U).\mathit{ltime} = \infty$, it follows that $\alpha \lceil (I, U) = \upsilon$, as needed.

■

The following theorem says that a progressive HIOA is capable of following not just individual input trajectories, but entire input hybrid sequences.

Theorem 7.2 *Let $\mathcal{A}$ be a progressive HIOA with state $\mathbf{x}$, and let β be an (I, U)-sequence. Then there exists an execution fragment α of $\mathcal{A}$ such that $\alpha.\mathit{fstate} = \mathbf{x}$ and $\alpha \lceil (I, U) = \beta$.*

Proof: Let $\beta = \tau_0 a_1 \tau_1 a_2 \tau_2 \ldots$. We define a finite or infinite sequence $\alpha_0, \alpha_1, \ldots$ of execution fragments of $\mathcal{A}$ such that:

(1) $\alpha_0.\mathit{fstate} = \mathbf{x}$.
(2) For every nonfinal index i, $\alpha_i.\mathit{lstate} = \alpha_{i+1}.\mathit{fstate}$.
(3) For every i, $(\alpha_0 \frown \alpha_1 \frown \cdots \frown \alpha_i) \lceil (I, U) = \tau_0\, a_1\, \tau_1\, a_2\, \tau_2 \ldots \tau_i$.

The construction is carried out recursively. To define α_0, we begin with $\mathbf{x}$ and use Lemma 7.1 to span τ_0. For $i > 0$, we define α_i by starting with $\alpha_{i-1}.\mathit{lstate}$, using property **E1** to perform action a_i and move to a new state, and then using Lemma 7.1 to span τ_i.

Let $\alpha = \alpha_0 \frown \alpha_1 \frown \cdots$. By Lemma 3.8 we conclude that $\alpha \lceil (I, U) = \beta$, as needed. ■

The property asserted in Theorem 7.2 has been called *I/O feasibility* elsewhere in the literature [59]. Thus, we define a pre-HIOA to be *I/O feasible* provided that, for each state $\mathbf{x}$ and each (I, U)-sequence β, there is some execution fragment α such that $\alpha.\mathit{fstate} = \mathbf{x}$ and $\alpha \lceil (I, U) = \beta$. Theorem 7.2 may then be restated as:

Corollary 7.3 *Every progressive HIOA is I/O feasible.*

I/O feasibility implies that any finite execution fragment can be extended to an admissible execution in response to any admissible input from the environment. A related, weaker property that has also been studied is *feasibility* [57]. In terms of our model, we may say that a pre-HIOA is *feasible* provided that, for each state $\mathbf{x}$, there is some admissible execution fragment α such that $\alpha.fstate = \mathbf{x}$.

Feasibility implies that any finite execution fragment can be extended to some admissible execution fragment—no constraints are imposed on the inputs. Observe that any I/O feasible HIOA must be feasible, as long as the dynamic type of each input variable includes at least one admissible trajectory. Feasibility should be regarded as a minimal liveness requirement that any reasonable HIOA should satisfy. I/O feasibility is a strengthened version of feasibility that takes inputs into account.

Closure under composition is easy to show:

Theorem 7.4 *If $\mathcal{A}_1$ and $\mathcal{A}_2$ are compatible progressive pre-HIOAs, then their composition is also progressive.*

Proof: Let $\mathcal{A}$ be $\mathcal{A}_1 \| \mathcal{A}_2$. Suppose for the sake of contradiction that $\mathcal{A}$ is not progressive. Then, by definition, $\mathcal{A}$ has a locally-Zeno execution fragment α, that is, α contains infinitely many locally controlled actions of $\mathcal{A}$. Therefore, α contains either infinitely many locally controlled actions of $\mathcal{A}_1$ or infinitely many locally controlled actions of $\mathcal{A}_2$. Suppose without loss of generality that α contains infinitely many locally controlled actions of $\mathcal{A}_1$. Then, by Lemma 5.3 and the definition of restriction, $\alpha \lceil (A_1, V_1)$ is a time-bounded execution fragment of $\mathcal{A}_1$ with infinitely many locally controlled actions, that is, a locally-Zeno execution fragment of $\mathcal{A}_1$. This contradicts the assumption that $\mathcal{A}_1$ is progressive. ■

Example 7.5 (Progressive and non-progressive pre-HIOAs) The *Vehicle* HIOA is obviously progressive because it has no discrete actions. The *Controller* and *Sensor* HIOAs are progressive because their locally controlled actions are separated in time. The *DiscreteController* HIOA is not progressive, because if *report* inputs arrive in a Zeno fashion, the *DiscreteController* may respond by performing *suggest* internal actions in a Zeno fashion. However, the composition *Sensor*$\|$*DiscreteController* is progressive.

Consider a more nondeterministic version of *Sensor*, *NSensor*, that is allowed to perform *report* actions for any value of *clock* ($\leq \mathsf{d}$), rather than just for *clock* $= \mathsf{d}$. Formally, *NSensor* is identical to *Sensor* except that condition (18) is dropped. *NSensor* is not progressive, because it may perform infinitely many *report* actions in finite time. Also, the composition of *NSensor* with

DiscreteController is not progressive. ■

7.2 Strategies

In this subsection, we define the notion of a *strategy*, which provides a way to resolve some of the nondeterministic choices in a pre-HIOA. We will use strategies in the next subsection to define receptiveness.

We define a *strategy* for a pre-HIOA $\mathcal{A}$ to be an HIOA $\mathcal{A}'$ that differs from $\mathcal{A}$ only in that $D' \subseteq D$ and $\mathcal{T}' \subseteq \mathcal{T}$. That is, we require:

- $D' \subseteq D$.
- $\mathcal{T}' \subseteq \mathcal{T}$.
- $W = W'$, $X = X'$, $Q = Q'$, $\Theta = \Theta'$, $E = E'$, $H = H'$, $U = U'$, $Y = Y'$, $I = I'$, and $O = O'$.

Our strategies are nondeterministic and memoryless. They serve to choose some of the evolutions that are possible from each state $\mathbf{x}$ of $\mathcal{A}$. The fact that the state set Q' of $\mathcal{A}'$ is the same as the state set Q of $\mathcal{A}$ implies that $\mathcal{A}'$ chooses evolutions from every state of $\mathcal{A}$.

Strategy notions have been used elsewhere in defining receptiveness, for example, in [21,1,74]. In this earlier work, strategies have been formalized using two-player games rather than restricted automata. Defining strategies using automata instead of two-player games allows us to avoid introducing extra mathematical machinery. A drawback of our approach is that it is not applicable in a setting with general liveness properties.

Lemma 7.6 *If $\mathcal{A}'$ is a strategy for $\mathcal{A}$, then every execution fragment of $\mathcal{A}'$ is also an execution fragment of $\mathcal{A}$.*

Theorem 7.7 *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two compatible pre-HIOAs with strongly compatible strategies $\mathcal{A}_1'$ and $\mathcal{A}_2'$, respectively. Then $\mathcal{A}_1' \| \mathcal{A}_2'$ is a strategy for $\mathcal{A}_1 \| \mathcal{A}_2$.*

Proof: Let $\mathcal{A}$ denote $\mathcal{A}_1 \| \mathcal{A}_2$ and let $\mathcal{A}'$ denote $\mathcal{A}_1' \| \mathcal{A}_2'$. Since $\mathcal{A}_1'$ and $\mathcal{A}_2'$ are strongly compatible, Theorem 6.12 implies that $\mathcal{A}'$ is an HIOA. From the definitions of composition and strategy, $\mathcal{A}'$ differs from $\mathcal{A}$ only in that $D' \subseteq D$ and $\mathcal{T}' \subseteq \mathcal{T}$. Then the definition of strategy implies that $\mathcal{A}'$ is a strategy for $\mathcal{A}$. ■

Lemma 7.8 *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two compatible pre-HIOAs with strongly compatible strategies $\mathcal{A}_1'$ and $\mathcal{A}_2'$, respectively. Then $\mathcal{A}_1$ and $\mathcal{A}_2$ are strongly compatible.*

Proof: Let $\mathcal{A}$ denote $\mathcal{A}_1 \| \mathcal{A}_2$ and let $\mathcal{A}'$ denote $\mathcal{A}'_1 \| \mathcal{A}'_2$. Theorem 7.7 implies that $\mathcal{A}'$ is a strategy for $\mathcal{A}$. Since $\mathcal{A}'_1$ and $\mathcal{A}'_2$ are strongly compatible, their composition $\mathcal{A}'$ satisfies **E2**. We show that also $\mathcal{A}$ satisfies **E2**.

Let $\mathbf{x} \in Q$ and let $v \in \mathit{trajs}(U)$. Then since $\mathcal{A}'$ is a strategy for $\mathcal{A}$, we have $Q' = Q$ and $U' = U$, $Y' = Y$, and so $\mathbf{x} \in Q'$ and $v \in \mathit{trajs}(U')$. Since $\mathcal{A}'$ satisfies **E2**, there exists $\tau \in \mathcal{T}'$ such that $\tau.\mathit{fstate} = \mathbf{x}$, $\tau \downarrow U' \leq v$, and either $\tau \downarrow U' = v$, or else τ is closed and some $l \in L'$ is enabled (in $\mathcal{A}'$) in $\tau.\mathit{lstate}$.

Since $\mathcal{A}'$ is a strategy for $\mathcal{A}$, it follows that also $\tau \in \mathcal{T}$, $\tau \downarrow U \leq v$, and either $\tau \downarrow U = v$, or else τ is closed and some $l \in L$ is enabled (in $\mathcal{A}$) in $\tau.\mathit{lstate}$. Therefore, $\mathcal{A}$ satisfies **E2**, that is, $\mathcal{A}_1$ and $\mathcal{A}_2$ are strongly compatible. ∎

Example 7.9 (Strategy for nondeterministic sensor) The *Sensor* HIOA defined in Example 5.14 is a strategy for the *NSensor* HIOA defined in Example 7.5. ∎

7.3 Receptive HIOAs

Finally, we define a pre-HIOA to be *receptive* if it has a progressive strategy.

Example 7.10 (Receptive and non-receptive HIOAs) The *NSensor* HIOA of Example 7.5 is not progressive, but it is receptive. That is because the original *Sensor* HIOA, as defined in Example 5.14, is a progressive strategy for *NSensor*.

The *DiscreteController* HIOA is not receptive: because any strategy for it must satisfy **E1** and **E2**, such a strategy must be able to perform discrete steps in response to any *report* input, and so must be capable of performing infinitely many *suggest* actions in finite time.

Consider a variant *NDController* of *DiscreteController* that has its own clock and may wait any amount of time, up to a fixed d' (> 0), to respond to each *report* input with a new *suggest*. (Several reports may occur in succession; a single *suggest* may be used to handle all of them, as long as it occurs within time d' of the first of these reports.) *NDController* is not progressive, because it has the option of responding immediately to reports, and thus may generate infinitely many suggestions in finite time. It is receptive, however, using a progressive strategy that always waits the maximum allowed time before generating a suggestion. ∎

The two most important general properties of receptive HIOAs are expressed by the following two theorems. The first expresses nontriviality—that any receptive HIOA (or pre-HIOA) can respond to any inputs from the environment. The second theorem shows that receptiveness is preserved by composition.

Theorem 7.11 *Every receptive pre-HIOA is I/O feasible.*

Proof: Let $\mathcal{A}$ be a receptive pre-HIOA. By definition of receptive, there exists a progressive strategy $\mathcal{A}'$ for $\mathcal{A}$. Since $\mathcal{A}'$ is a progressive HIOA, Corollary 7.3 implies that $\mathcal{A}'$ is I/O feasible. We show that also $\mathcal{A}$ is I/O feasible.

Let $\mathbf{x} \in Q$ and let β be an (I, U)-sequence. Then since $\mathcal{A}'$ is a strategy for $\mathcal{A}$, we have $Q' = Q$, $I' = I$, and $U' = U$, and so $\mathbf{x} \in Q'$ and β is an (I', U')-sequence. Since $\mathcal{A}'$ is I/O feasible, there is some execution fragment α of $\mathcal{A}'$ such that $\alpha.\mathit{fstate} = \mathbf{x}$ and $\alpha \lceil (I', U') = \beta$. By Lemma 7.6, α is also an execution fragment of $\mathcal{A}$. Since $\mathcal{A}'$ is a strategy for $\mathcal{A}$, it follows that $\alpha \lceil (I, U) = \beta$. Therefore, $\mathcal{A}$ is I/O feasible. ■

The question of whether the converse of Theorem 7.11 holds is still open. Finally, we have our theorem about composability of receptive HIOAs:

Theorem 7.12 *Let $\mathcal{A}_1$ and $\mathcal{A}_2$ be two compatible receptive HIOAs with strongly compatible progressive strategies $\mathcal{A}_1'$ and $\mathcal{A}_2'$, respectively. Then $\mathcal{A}_1 \| \mathcal{A}_2$ is a receptive HIOA with progressive strategy $\mathcal{A}_1' \| \mathcal{A}_2'$.*

Proof: Let $\mathcal{A}$ and $\mathcal{A}'$ denote $\mathcal{A}_1 \| \mathcal{A}_2$ and $\mathcal{A}_1' \| \mathcal{A}_2'$, respectively. The fact that $\mathcal{A}$ is an HIOA follows from Lemma 7.8 and Theorem 6.12. Theorem 7.7 implies that $\mathcal{A}'$ is a strategy for $\mathcal{A}$. Theorem 7.4 and the fact that $\mathcal{A}_1'$ and $\mathcal{A}_2'$ are progressive implies that $\mathcal{A}'$ is progressive. Thus, $\mathcal{A}$ is a receptive HIOA and $\mathcal{A}'$ is a progressive strategy for $\mathcal{A}$. ■

Example 7.13 (Composition of receptive sensor and receptive discrete controller) As noted in Example 7.10, both *NSensor* and *NDController* are receptive, using progressive strategies that always wait the maximum allowed amount of time. These two strategies are strongly compatible, by Theorem 6.18. Therefore, by Theorem 7.12, the composition *NSensor*$\|$*NDController* is a receptive HIOA with a progressive strategy that is the composition of the two progressive strategies for the two pieces. ■

8 Conclusions

In this paper, we have defined a new hybrid I/O automaton (HIOA) modeling framework for describing and reasoning about the behavior of hybrid systems. Many future research directions remain.

First, the expressive and analytical power of the new model should be tested further by using it to describe and analyze many more examples. These should include many of the examples that have been used as illustrations elsewhere in the hybrid systems literature. The automated transportation examples studied using the previous version of the HIOA model should be revisited using the new model to see what changes arise, and new and more ambitious case studies should be attempted.

It would be interesting to define and prove formal relationships between the HA and HIOA models of this paper and other models of hybrid systems, including those of [63,3,13,8,14,38]. Also, one can define a timed input/output automaton model by simply restricting the HIOA model of this paper so that it does not include any external variables. It remains to consider the formal relationship between this model and other timed automaton models, for example, those of [1,5,60,74,65].

It would also be useful to incorporate additional analysis methods, including assume-guarantee reasoning [16,36] and a variety of methods from control theory, into the HIOA framework. Control theory methods to consider should include Lyapunov stability analysis methods [79] and robust control methods [23]. Results about these methods should be formulated in terms of HIOAs, and the methods should be extended where necessary in order to accommodate a combination of discrete and continuous behavior.

Other extensions of the HIOA framework are also desirable. In some prior work (e.g., [21,1,74]), strategies are used to describe how a system interacts with its environment to guarantee that the outcome of the interaction satisfies a target liveness property. In this paper, we do not consider general liveness properties, but only the special case of admissibility. It remains to extend the theory to more general liveness properties. Another important extension would be the addition of probabilities, which would make it possible to model and analyze probabilistic hybrid systems. Such an extension could be used, for example, to prove bounds on the probability of errors in safety-critical real-time systems. This extension appears to be a very challenging problem.

Future work will include tool support for modeling and analysis as described in this paper. This will include a formal modeling language based on HIOA, with constructs similar to those used in the examples of this paper, and connections to a theorem prover. A preliminary language proposal appears in [68].

Acknowledgments: We thank Ekaterina Dolginova, Carl Livadas, John Lygeros, Sayan Mitra, and Natasha Neogi for working with versions of our HIOA model while it was evolving; their questions and observations have helped us greatly in completing the development of the model. We also thank Paul Attie for reading and commenting on an earlier version of the paper, and finding a bug in a definition. Finally, we thank the referees for their insightful reports.

References

[1] M. Abadi and L. Lamport. Composing specifications. *ACM Transactions on Programming Languages and Systems*, 1(15):73–132, 1993.

[2] R. Alur. Timed automata. In *NATO-ASI Summer School on Verification of Digital and Hybrid Systems*. Springer-Verlag, 1998.

[3] 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.

[4] 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. [28], pages 209–229.

[5] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[6] R. Alur, R. Grosu, I. Lee, and O. Sokolsky. Compositional refinement of hierarchical hybrid systems. In Di Benedetto and Sangiovanni-Vincentelli [20], pages 33–48.

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

[8] 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.

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

[10] P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, editors. *Hybrid Systems IV (Fourth International Conference on Hybrid Systems, Ithaca, NY, October 1996)*, volume 1273 of *Lecture Notes in Computer Science*. Springer-Verlag, 1997.

[11] D.J.B. Bosscher, I. Polak, and F.W. Vaandrager. Verification of an audio control protocol. In Langmaack et al. [41], pages 170–192.

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

[13] 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.

[14] M.S. Branicky. Analyzing and synthesizing hybrid control systems. In Rozenberg and Vaandrager [73], pages 74–113.

[15] M.S. Branicky, E. Dolginova, and N.A. Lynch. A toolbox for proving and maintaining hybrid specifications. In Antsaklis et al. [10], pages 18–30.

[16] E.M. Clarke, O. Grumberg, and D. Peled. *Model Checking.* MIT Press, Cambridge, Massachusetts, 1999.

[17] 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.

[18] 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.

[19] Roberto DePrisco, Butler Lampson, and Nancy Lynch. Revisiting the Paxos algorithm. In Marios Mavronicolas and Philippas Tsigas, editors, *Distributed Algorithms* 11th International Workshop, WDAG'97, Saarbrücken, Germany, September 1997 Proceedings, volume 1320 of *Lecture Notes in Computer Science*, pages 111–125, Berlin-Heidelberg, 1997. Springer-Verlag.

[20] M.D. Di Benedetto and A.L. Sangiovanni-Vincentelli, editors. *Proceedings Fourth International Workshop on Hybrid Systems: Computation and Control (HSCC'01),* Rome, Italy, volume 2034 of *Lecture Notes in Computer Science.* Springer-Verlag, March 2001.

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

[22] E. Dolginova and N.A. Lynch. Safety verification for automated platoon maneuvers: A case study. In Maler [62], pages 154–170.

[23] P. Dorato, editor. *Robust Control.* IEEE Press, New York, 1987.

[24] A. Fehnker. Automotive control revisited — linear inequalities as approximation of reachable sets. In Henzinger and Sastry [34], pages 110–125.

[25] Alan Fekete, Nancy Lynch, and Alex Shvartsman. Specifying and using a partitionable group communication service. *ACM Transactions on Computer Systems*, 19(2):171–216, May 2001.

[26] R. W. Floyd. Assigning meanings to programs. *Mathematical Aspects of Computer Science*, pages 19–32, 1967. From *Proceedings of Symp. Appl. Math. 19.*

[27] R. Gawlick, R. Segala, J.F. Søgaard-Andersen, and N.A. Lynch. Liveness in timed and untimed systems. In S. Abiteboul and E. Shamir, editors, *Proceedings* 21^{th} *ICALP,* Jerusalem, volume 820 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994. A full version appears as MIT Technical Report number MIT/LCS/TR-587.

[28] 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.

[29] C.A. Gunter. *Semantics of Programming Languages: Structures and Techniques.* MIT Press, Cambridge, Massachusetts, 1992.

[30] Constance Heitmeyer and Nancy Lynch. The generalized railroad crossing: A case study in formal verification of real-time systems. In *Proceedings of the Real-Time Systems Symposium*, pages 120–131, San Juan, Puerto Rico, December 1994. IEEE.

[31] Constance Heitmeyer and Nancy Lynch. Formal verification of real-time systems using timed automata. In Constance Heitmeyer and Dino Mandrioli, editors, *Formal Methods for Real-Time Computing*, Trends in Software, chapter 4, pages 83–106. John Wiley & Sons Ltd, April 1996.

[32] M. Hennessy. *Algebraic Theory of Processes.* MIT Press, Cambridge, Massachusetts, 1988.

[33] T.A. Henzinger, M. Minea, and V. Prabhu. Assume-guarantee reasoning for hierarchical hybrid systems. In Di Benedetto and Sangiovanni-Vincentelli [20], pages 275–290.

[34] 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.

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

[36] C. B. Jones. *Development Methods for Computer Programs including a Notion of Interference.* PhD thesis, Oxford University, June 1981. Printed as Programming Research Group, Technical Monograph 25.

[37] A. Kapur, T.A. Henzinger, Z. Manna, and A. Pnueli. Proving safety properties of hybrid systems. In Langmaack et al. [41], pages 431–454.

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

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

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

[41] 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*. Springer-Verlag, 1994.

[42] 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.

[43] C. Livadas, J. Lygeros, and N.A. Lynch. High-level modelling and analysis of tcas. In *Proceedings of the IEEE Real-Time Systems Symposium (RTSS'99).* IEEE Computer Society Press, 1999.

[44] C. Livadas and N.A. Lynch. Formal verification of safety-critical hybrid systems. In Henzinger and Sastry [34], pages 253–272.

[45] Victor Luchangco, Ekrem Söylemez, Stephen Garland, and Nancy Lynch. Verifying timing properties of concurrent algorithms. In Dieter Hogrefe and Stefan Leue, editors, *Formal Description Techniques VII: Proceedings of the 7th IFIP WG6.1 International Conference on Formal Description Techniques* (FORTE'94, Berne, Switzerland, October 1994), pages 259–273. Chapman and Hall, 1995.

[46] 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.

[47] J. Lygeros and N.A. Lynch. Strings of vehicles: Modeling and safety conditions. In Henzinger and Sastry [34], pages 273–288.

[48] N.A. Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers, Inc., San Fransisco, California, 1996.

[49] N.A. Lynch. Modelling and verification of automated transit systems, using timed automata, invariants and simulations. In Alur et al. [9], pages 449–463.

[50] 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.

[51] N.A. Lynch and B.H. Krogh, editors. *Proceedings Third International Workshop on Hybrid Systems: Computation and Control (HSCC 2000),* Pittsburgh, PA, USA, volume 1790 of *Lecture Notes in Computer Science*. Springer-Verlag, March 2000.

[52] N.A. Lynch, R. Segala, and F.W. Vaandrager. Hybrid I/O automata revisited. In Di Benedetto and Sangiovanni-Vincentelli [20], pages 403–417.

[53] N.A. Lynch, R. Segala, F.W. Vaandrager, and H.B. Weinberg. Hybrid I/O automata. In Alur et al. [9], pages 496–510.

[54] N.A. Lynch, R. Segala, F.W. Vaandrager, and H.B. Weinberg. Hybrid I/O automata. Report CSI-R9907, Computing Science Institute, University of Nijmegen, April 1999.

[55] 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.

[56] N.A. Lynch and M.R. Tuttle. An introduction to input/output automata. *CWI Quarterly*, 2(3):219–246, September 1989.

[57] N.A. Lynch and F.W. Vaandrager. Forward and backward simulations for timing-based systems. In de Bakker et al. [17], pages 397–446.

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

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

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

[61] N.A. Lynch and H.B. Weinberg. Proving correctness of a vehicle maneuver: Deceleration. In Bouajjani and Maler [12].

[62] 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.

[63] O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In de Bakker et al. [17], pages 447–484.

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

[65] Michael Merritt, Francemary Modugno, and Mark R. Tuttle. Time constrained automata. In J. C. M. Baeten and J. F. Goote, editors, *CONCUR'91: 2nd International Conference on Concurrency Theory* (Amsterdam, The Netherlands, August 1991), volume 527 of *Lecture Notes in Computer Science*, pages 408–423. Springer-Verlag, 1991.

[66] R. Milner. *Communication and Concurrency.* Prentice-Hall International, Englewood Cliffs, 1989.

[67] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, Part I + II. *Information and Computation*, 100(1):1–77, 1992.

[68] Sayan Mitra, Yong Wang, Nancy Lynch, and Eric Feron. Safety verification of model helicopter controller using hybrid input/output automata. In *Hybrid Systems: Computation and Control (HSCC'03),* Prague, the Czech Republic, pages 259–273. LNCS, Springer Verlag, 2003.

[69] A. Pnueli. Development of hybrid systems. In Langmaack et al. [41], pages 77–85.

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

[71] J.W. Polderman and J.C. Willems. *Introduction to Mathematical Systems Theory: A Behavioural Approach*, volume 26 of *Texts in Applied Mathematics.* Springer-Verlag, 1998.

[72] W.P. de Roever and K. Engelhardt. *Data Refinement: Model-Oriented Proof Methods and their Comparison.* Cambridge Tracts in Theoretical Computer Science 47. Cambridge University Press, 1998.

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

[74] 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.

[75] Mark Smith. Formal verification of communication protocols. In Reinhard Gotzhein and Jan Bredereke, editors, *Formal Description Techniques IX: Theory, Applications, and Tools* FORTE/PSTV'96: Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols, and Protocol Specification, Testing, and Verification, Kaiserslautern, Germany, October 1996, pages 129–144. Chapman & Hall, 1996.

[76] Mark Smith. Formal verification of TCP. In *Proceedings of The Second Technical Conference on Telecommunications R&D in Massachusetts*, pages 279–299, Lowell, MA, March 1996.

[77] Mark Smith. Reliable message delivery and conditionally-fast transactions are not possible without accurate clocks. In *Proceedings of the 17th Annual ACM Symposium on the Principles of Distributed Computing*, pages 163–171, June 1998.

[78] J. Søgaard-Andersen, S. Garland, J. Guttag, N.A. Lynch, and A. Pogosyants. Computer-assisted simulation proofs. In C. Courcoubetis, editor, *Proceedings of the 5th International Conference on Computer Aided Verification,* Elounda, Greece, volume 697 of *Lecture Notes in Computer Science*, pages 305–319. Springer-Verlag, 1993.

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

[80] 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.

[81] 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.

[82] 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.

[83] H.B. Weinberg, N.A. Lynch, and N. Delisle. Verification of automated vehicle protection systems. In Alur et al. [9], pages 101–113.

A Notational Conventions

a, b	action
c, d	element of some set
f, g, h	function
i, j	index
k	natural number
l	locally controlled action
t	time point
u	input variable
v	variable
w	external variable
x	internal variable
y	output variable
z	local variable
A	set of actions
D	set of discrete transitions
E	set of external actions
F	set of functions
H	set of internal (hidden) actions
I	set of input actions or index set
J	interval or index set
K	set of time points
L	set of locally controlled actions
O	set of output actions
P	set of elements in cpo
Q	set of automaton states
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
$\mathbf{x}$	state
$\mathbf{v}$	valuation
$\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$	hybrid (I/O) automaton
$\mathcal{H}$	hybrid automaton
$\mathcal{T}$	set of trajectories
N	the natural numbers
R	the real numbers
T	the time axis
Z	the integers
V	the universe of variables
α, β, δ	hybrid sequence
γ	sequence
λ	the empty sequence
π	projection function
ρ, σ	sequence
τ, υ	trajectory
Θ	set of start states