HIOA: Language for Specifying Hybrid I/O Automata

Sayan Mitra

Laboratory for Computer Science, MIT

October, 2001

Script Notes:

In this talk I shall first briefly discuss the mathematical framework of hybrid I/O automaton, which has been developed to specify and verify hybrid systems. Then I will discuss the specification language called HIOA which is based on hybrid I/O automaton model, along with a few simple examples.I had started working on this language as a part of my masters thesis, and we are still working on it. Finally, I will present parts of the specification of the Quanzer system in HIOA. This specification is not yet complete, we are working with yong, but I chose to include it just to give you an idea of how the specs of a real system would BE like.

Talk Outline

I. Introduction : The Framework

II. The Hybrid I/O Automaton Model + *Activities*

III. The HIOA Language

- Simple Examples
- Composition

IV. Quanser Systems Specs

I. Introduction : The Framework

Very general : many methods from control theory, computer science can fit.

Mathematical, not language-based

Hybrid Input/Output Automaton

- States
- Discrete and continuous transitions.
- Nondeterministic.

External behavior: Models discrete & continuous interactions of component with environment.

Composition: Synchronize external discrete actions and continuous "flows". Respects external behavior.

Levels of abstraction: Implementation notion, respects external behavior.

Script Notes: Introduction: The framework

Most of our work related to this group, will be centerd around the Hybrid I/O automaton modeling framework, so I think it would be reasonable to spend a little time on discussing the principal aspects of it. All the future proofs and verifications will be based on this framework, so let me first highlight the important characteristics of the hybrid I/O automaton model, and then in the next 4-5 slides I'll discuss some of the details and gloss over most of the technicalities. OK..

The HIOA model is designed to serve as the mathematical foundation for a very general framework for modeling and analyzing hybrid systems.

Intended to be compatible both with control-theoretic and cs-oriented models and analysis methods.
Fundamental object in the model is a kind of Nondeterministic state machine called a "Hybrid Input/Output Automaton", which has states and both discrete and continuous transitions.

The complex system is described as an interaction of several such state machines so HIOAs can be *composed*, to produce a model for a complex hybrid system of interacting components, interacting by means of both discrete actions and continuous "flows".

Also there is a Notion of *external behavior* for a particular state machine, which is a simple mathematical object that describes its *discrete and continuous* interactions with its environment.

This composition notion respects the external behavior notion.

HIOAs can also be used to view systems at multiple *levels of abstraction*. In particular, the HIOA model supports a notion of *implementation*, which also respects external behavior:
That is, if one automaton implements another, then it means that any external behavior that can be exhibited by the implementation is allowed by the specification.

II. The Hybrid I/O automaton Model

V, universal set of *variables*.

Internal / State : Component states

External : Information flow between components.

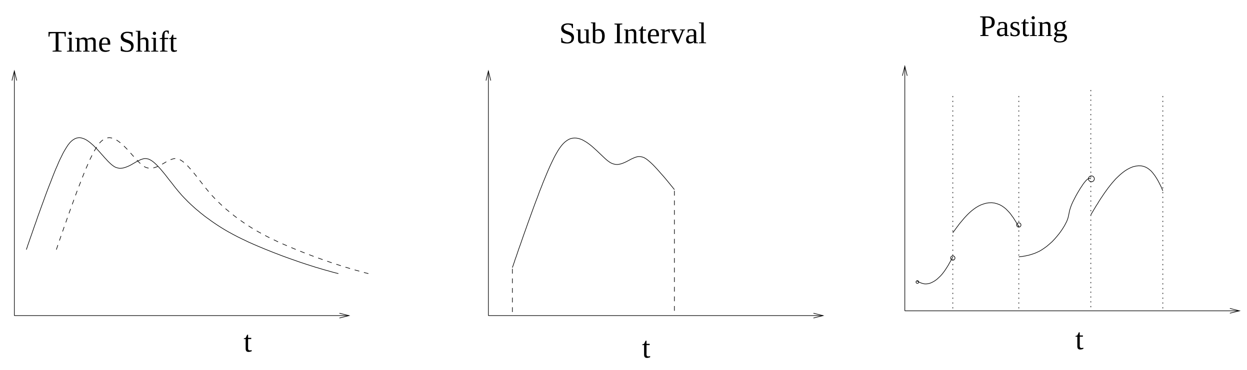
Static type: type(v), set of values v may take on.

Valuation for V: Assigns value in *type*(v) to each $v \in V$.

Dynamic type: dtype(v), allowed "trajectories" for v:

Set of functions from left-closed intervals of $\mathbb{R}$ to *type*(v).

Closed under time shift, subinterval, countable pasting.



Script Notes: II. The Hybrid I/O automaton Model

With those basic features in mind, now we look at the details of the framework. But before describing the state model itself we need to define some terms and build up the basic machinery, as *names inevitable imply different things to different people*.

First, Assume universal set V of *variables*.
Variables represent either a state of the system component (Internal), or information flow from one system component to another (External).

Each variable v has a *(static) type* which is the set of values that it may take. A Valuation for the set of variables is a function which assigns each variable in V to some value in $type(v)$. (V) is the set of all valuations of V.
It is to be noted that variables in the hybrid I/O automaton are not functions of time (unlike state variables), instead we have a function which maps an interval of time to the possible values of a variable. This function is called a trajectory, which we define shortly.

Each variable also has a *dynamic type* which is the set of allowed trajectories for v. But we have not defined a trajectory as yet, so we have to give a more explicit definition of dtype

dtype(v), the *dynamic type* of v. This is a set of functions from left-closed intervals of $\mathbb{R}$ to *type*(v) that is closed under the following operations:

1. (Closure under time shift) : For each $f \in$ *dtype(v)* and $t \in$ T, $f + t \in$ *dtype(v)*. Here $f + t$ is the function given by $(f + t)\ (t') = f(t' - t)$.
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)*. Here $f \lceil J$ is the function obtained by restricting the domain of f to J.
3. (Closure under pasting) : For each sequence $f_0, f_1, f_2, \ldots$ of functions in *dtype(v)* such that
   (a) the domain of each f_i, except possibly for the last one, is right-closed,
   (b) for each nonfinal index i, $\max($*dom*$(f_i)) = \min($*dom*$(f_{i+1}))$,
   the function f given by $f(t) \stackrel{\Delta}{=} f_i(t)$, where i is the smallest index with $t \in$ *dom*(f_i), is in *dtype(v)*.

Pasting closure of constant functions : **discrete dtype**.
Other interesting dtypes : pasting closure of continuous functions, differentiable functions, integrable functions.

Trajectories and Hybrid Sequences

Trajectories model evolution of variables over time intervals.

Let J = left-closed interval of $\mathbb{R}$, left endpoint 0.
J-trajectory for V is a function $\tau : J \rightarrow \boldsymbol{V}$, where $\tau \downarrow v \in$ *dtype(v)*.

Limit time : τ.*ltime* : supremum of *dom*(τ)
First, Limit values : τ.*fval*, τ.*lval*.
Concatenation: τ, τ' trajectories, τ closed. $\tau \frown \tau' = \tau \cup (\tau' \lceil (0, \infty) + \tau.\mathit{ltime})$.

Let A be set of actions, V set of variables.
Then an (A, V)-*sequence* is an alternating sequence
$\tau_0\, a_1\, \tau_1\, a_2\, \tau_2 \cdots$ of trajectories in *trajs*(V) and actions in A.

Models a series of discrete and continuous changes.
Concatenation: α, α' be (A, V)-sequences, α closed.
Then $\alpha \frown \alpha' = init(\alpha)\ (\mathit{last}(\alpha) \frown head(\alpha'))\ \mathit{tail}(\alpha')$.

Script Notes: Trajectories and Hybrid Sequences

Now we define two important structures of this models, first is trajectory, as I mentioned earlier, the variables are not functions of time, instead we have for a Let J be a left-closed interval of $\mathbb{R}$ with left endpoint equal to 0.
Then a J-*trajectory* for V is a function $\tau : J \to \boldsymbol{V}$, such that for each $v \in V$, $\tau \downarrow v \in$ *dtype(v)*.
Here $\tau \downarrow v$ is the function with domain J defined by $(\tau \downarrow v)(t) = \tau(t)(v)$.

τ is *finite* if J is a finite interval, *closed* if J is a (finite) closed interval, and *full* if $J = T^{\geq 0}$.

A trajectory with domain $[0, 0]$ is called a *point* trajectory. If $\mathbf{v}$ is a valuation then $\wp(\mathbf{v})$ denotes the point trajectory that maps 0 to $\mathbf{v}$.
τ.*ltime*, the *limit time* of τ, is the supremum of *dom*(τ).
τ.*fval*, the *first valuation* of τ, is $\tau(0)$. If τ is closed, τ.*lval*, the *last valuation* of τ, to be $\tau(\tau.\mathit{ltime})$.

Prefix ordering: Trajectory τ is a *prefix* of trajectory υ, denoted by $\tau \leq \upsilon$, if τ can be obtained by restricting υ to a non-empty, downward closed subset of its domain.
Formally, $\tau \leq \upsilon$ iff $\tau = \upsilon \lceil \mathit{dom}(\tau)$.

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

Concatenation: Let τ, τ' be trajectories, with τ closed. Then the *concatenation* $\tau \frown \tau'$ is the function obtained by taking the union of τ with the function obtained by shifting the domain of τ' until the start time agrees with $(.\mathit{ltime}\tau)$.
Note there may be a conflict between two values at the boundary between the two trajectories.
Resolve this in favor of the last valuation of the first trajectory τ. Extend concatenation to any (finite or countably infinite) number of arguments.

Next we define **A,V Sequences** An (A, V)-*sequence* is a finite or infinite alternating sequence $\alpha = \tau_0\, a_1\, \tau_1\, a_2\, \tau_2 \cdots$ of trajectories in *trajs*(V) and actions in A, where: If α is a finite sequence then it ends with a trajectory, and If τ_i is not the last trajectory in α then *dom*(τ_i) is closed.

We define a *hybrid sequence* to be an (A, V)-sequence for some A and V.
Concatenation: Let α and α' be (A, V)-sequences, α closed.
Then the *concatenation* $\alpha \frown \alpha'$ is the (A, V)-sequence $init(\alpha)$ (*last*$(\alpha) \frown head(\alpha')$) *tail*$(\alpha')$.

Hybrid I/O Automata

Hybrid I/O Automaton $\mathcal{A}$ consists of:

U, input variables
Y, output variables
X, internal (state) variables
    States = valuations of X
    $Z = X \cup Y$,
    $V = U \cup Y \cup X$

Θ, start states

I, input actions
O, output actions
H, internal actions
    $A = I \cup O \cup H$

$\mathcal{D} \subseteq \boldsymbol{X} \times A \times \boldsymbol{X}$, discrete transitions.

$\mathcal{T}$, a set of trajectories for V.
    Closed under prefix, suffix, countable pasting.

Script Notes: H I/O A

So now we have all the necessary equipment to define the hybrid I/O automaton model. A hybrid I/O Auto consists of 9 components : a set of input vars, output vars and internal (or state) vars

The states of the automaton are actually the valuations of the internal variables

Output and internal variables are called locally controlled variables.

A set of valuations of the internal variables is designated as the start state

In an identical manner, we have a set of input output and internal actions.
A set of discrete transitions $\mathcal{D}$, defining the state change due to the discrete actions. these transitions are assumed to be instantaneous.

And a set of trajectories $\mathcal{T}$, which defines the allowed evolution of the continuous variables in Z over time.

Input Enabling

For input actions:

For all states $\mathbf{x}$ and all input actions a, there is some discrete transition $(\mathbf{x}, a, \mathbf{x}')$.

As for ordinary I/O automata.

For input flows:

For all states $\mathbf{x}$ and all input trajectories $v \in \textit{trajs}(U)$, there is some trajectory τ that starts with $\mathbf{x}$, and either:

1. $\tau \downarrow U = v$, or
2. $\tau \downarrow U \leq v$ and some non-input action is enabled after τ.

Script Notes: Input Enabling

A component is supposed to be able to respond to any inputs that arrive.
Cannot block inputs.

If not, we might have trivial solutions to problems that are deemed to be correct just because they refuse to consider certain inputs.

The analogy here involves two properties, one for enabling input actions and one for input variables (flows).

Input flow enabling

For all $\mathbf{x} \in \boldsymbol{X}$ and $\upsilon \in \textit{trajs}(U)$, there exists $\tau \in \mathcal{T}$ such that $\tau.\textit{fstate} = \mathbf{x}, \tau \downarrow U \leq \upsilon$, and either

1. $\tau \downarrow U = \upsilon$, or
2. there exists $t \in \textit{dom}(\tau)$, $l \in L$ such that l is enabled from $\tau(t)$.

A new condition for continuous interaction.
Says that an HIOA should be able to accept any continuous input flow, either by letting time advance for the entire duration of the input flow, or by reacting with a non-input action after some part of the input flow has occurred.

Executions and Traces

Execution fragment of HIOA $\mathcal{A}$: record discrete and continuous changes

An (A, V)-sequence $\tau_0\, a_1\, \tau_1\, a_2\, \tau_2 \cdots$, where:

Each τ_i is a trajectory of $\mathcal{A}$, and

Each $(\tau_i.\textit{lstate}, a_{i+1}, \tau_{i+1}.\textit{fstate})$ is a discrete step of $\mathcal{A}$.

Only states need match up.

Execution of $\mathcal{A}$: Execution fragment beginning in a start state.
First state $\alpha.fstate(\tau_0.\textit{fstate})$. If closed then last state $\alpha.\textit{lstate}$

Trace of an execution fragment: (E, W)-sequence
Restrict to external actions E, external variables W.

Trace fragment of $\mathcal{A}$, *trace* of $\mathcal{A}$

Behavior/properties of automata are specified in terms of Traces

Script Notes: Executions and Traces

That looks like a lot of notation, but assuming that we can model a hybrid system using the H I/O A, how to reason about it's behavior ? How to model its behavior ? well, the (A,V) sequence defined earlier turns out to be useful.

An execution fragment records all the instantaneous, discrete state changes that occur during a specific evolution of a system, as well as the state changes and external variable changes that occur while time advances.

Note that we only preserve consistency of states, not the external variables.

If α is an execution fragment, with notation as above, then we define the *first state* of α, α.*fstate*, to be *state*(α.*fval*), or equivalently, τ_0.*fstate*.
An execution fragment α is defined to be an *execution* if α.*fstate* is a start state, that is, is in Θ.

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

Recall, I had mentioned earlier a thing about H I/O A having an external behavior ! Well, we define a trace which records only the external actions and the evolution of external variables of an execution fragment.

Formally, if α is an execution fragment, then the *trace* of α, notation *trace*(α), is the (E, W)-restriction of α.
A *trace fragment* of a hybrid automaton $\mathcal{A}$ *from* a state $\mathbf{x}$ of $\mathcal{A}$ is a trace that arises from an execution fragment of $\mathcal{A}$ whose first state is $\mathbf{x}$.
Also, we define a *trace* of $\mathcal{A}$ to be a trace fragment from an initial state, that is, a trace that arises from an execution of $\mathcal{A}$, and write *h-traces*($\mathcal{A}$) for the set of traces of $\mathcal{A}$.

Composition

Compose HIOAs by matching up external actions, external variables.
$U = (U_1 \cup U_2) - (Y_1 \cup Y_2)$; $Y = Y_1 \cup Y_2$; $X = X_1 \cup X_2$
$I = (I_1 \cup I_2) - (O_1 \cup O_2)$; $O = O_1 \cup O_2$; $H = H_1 \cup H_2$

Start states Θ: Projections in Θ_1, Θ_2
Discrete steps $\mathcal{D}$: Projections in $\mathcal{D}_1$, $\mathcal{D}_2$
Trajectories $\mathcal{T}$: Projections in $\mathcal{T}_1$, $\mathcal{T}_2$

Theorems:
Projection: Execution of composition projects to give execution of components.

Pasting: Executions of components that match up give execution of composition.

Substitutivity: If $\mathcal{A}_1$ implements $\mathcal{A}_2$ then $\mathcal{A}_1 \| \mathcal{B}$ implements $A_2 \| \mathcal{B}$.

Script Notes: Composition

The second thing I had mentioned was that we could put together hybrid automatons in a nice way and it would preserve most of its property. well, here we have the composition operator which does just that.
Compose HIOAs by matching up:
Output actions of one automaton with input actions of one or more other automata (or several inputs with each other)
Output variables of one automaton with input variables of other automata (or...)

But in order to preserve the properties, the to-be-composed automata should satisfy certain properties.
Compatibility

$\mathcal{A}_1 \| \mathcal{A}_2$ is defined to be the structure $\mathcal{A}$, where

- $U = (U_1 \cup U_2) - (Y_1 \cup Y_2)$; $Y = Y_1 \cup Y_2$; $X = X_1 \cup X_2$

  $I = (I_1 \cup I_2) - (O_1 \cup O_2)$; $O = O_1 \cup O_2$; $H = H_1 \cup H_2$

- $\Theta = \{\mathbf{x} \in \boldsymbol{X} \mid \mathbf{x}\lceil X_1 \in \Theta_1 \wedge \mathbf{x}\lceil X_2 \in \Theta_2\}$
- For each $\mathbf{x}, \mathbf{x'} \in \boldsymbol{X}$ and each $a \in A$, $\mathbf{x} \xrightarrow{a}_{\mathcal{A}} \mathbf{x'}$ iff for $i = 1, 2$, either (1) $a \in A_i$ and $\mathbf{x}\lceil X_i \xrightarrow{a}_i \mathbf{x'}\lceil X_i$, or (2) $a \notin A_i$ and $\mathbf{x}\lceil X_i = \mathbf{x'}\lceil X_i$
- $\mathcal{T} \subseteq$ *trajs*(V) is given by $\tau \in \mathcal{T} \iff \tau \downarrow V_1 \in \mathcal{T}_1 \wedge \tau \downarrow V_2 \in \mathcal{T}_2$

III HIOA : The Specification Language

Language for specifying Hybrid Input Output Automata

Uniform unambiguous description of system and requirements

Provide platform for using different analysis tools for different parts

Computer aided verification

Present restrictions.

Dynamic types : discrete and differentiable only

Non time varying systems : coefficients independent of time

Activities : structured representation of trajectories

Script Notes: HIOA : Design Policies

Now we discuss the specification language we are developing for this modeling frame work. First a few words on motivation .. why do we need such a language, since it is perfectly ok to write hybrid I/O automatons on paper and prove their properties by hand, like we do for simple finite state machines..
Here are a few reasons:
First, while H I/O A is very general and expressive, adding a little bit of structure to it (particularly with trajectories) will be useful.
Writing a clear, unambiguous and correct specification of the system (in this case a hybrid automaton) and also the target property, is always desirable before we set out to prove the property. *clarity* is provided by having a simple / natural design of the specification language, and unambiguity and correctness can be ensured by a syntactic and semantic checker of the language.

Besides, in an inter-disciplinary area like hybrid systems, (in a group like ours) a specification language can be the mode of communication.

Once we have a specification language, with its front-end, it can be used by different analysis tools independent of each-other. different computer-assisted tools could also be brought to bear on pieces of the problem; the framework and the language can be used to justify why the results fit together correctly.

Given the complexity of hybrid systems, and their verification, I think certain degree of automation in the verification process will be essential to deal with any remotely useful system. Thus a spec language is the first step towards automation of verification.

Current Restrictions on HIOA

- dynamic types: Presently HIOA supports only discrete and differentiable types. Include continuous, differentiable types.
- Non time-varying systems: coefficients of differential equations are independent of time. This can be easily extended
- activities: for structured specification of the trajectories

Extend IOA syntax for specifying discrete transitions

state model notation to specify trajectories. HIOA does not help in solving the equations. Tools for simulating or executing the HIOA specs will have to solve the differential equations to determine the trajectories.

Specifying Trajectories

A trajectory $\tau \in \mathcal{T}$ of $\mathcal{A}$ is specified by the evolution of locally controlled variables Z.

Trajectories specified by State Model

First order differential and algebraic equations

Output Y, Input U, State X.

State eqns : $i = 1, \ldots, n$

$$\dot{\tau(t)}(x_i) = f_i(\tau(t)(x_1), \ldots, \tau(t)(x_n), \tau(t)(u_1), \ldots, \tau(t)(u_m))$$

Output eqns : $i = 1, \ldots, p$

$$\tau(t)(y_i) = g_i(\tau(t)(x_1), \ldots, \tau(t)(x_n), \tau(t)(u_1), \ldots, \tau(t)(u_m))$$

Written as $E : \dot{X} = F(X, U); Y = G(X, U)$.

Discrete internal variables remain constant.

$[\![E]\!]$: set of trajectories defined by E.

Script Notes: Specifying Trajectories

A good part of the design of the HIOA has been derived from the IOA language. In today's presentation I will cursorily discuss the specification of the trajectories and move onto some simple examples.

- A trajectory $\tau \in \mathcal{T}$ of $\mathcal{A}$ is specified by the evolution of its locally controlled variables Z; $\mathcal{A}$ has no control over its input variables U.
- State Model specifies the trajectory of each locally controlled variable $v \in Z$.
  First order differential and algebraic equations
  function of time and the variables in $U \cup X$.
  τ can be specified by stating: $\forall v_i \in Z, \quad \forall t \in dom(\tau), \quad \tau(t)(v_i) = f_i(x_1, \ldots, x_n, u_1, \ldots, u_m, t)$ where $f_i \in dtype(v_i)$.
- This representation consists of first order differential equations describing the dynamic behavior of the state variables and algebraic functions expressing the behavior of the output variables.

  $\dot{x}_i = f_i(x_1, x_2, \ldots, x_n, u_1, u_2, \ldots, u_m) \;\; i = 1, \ldots, n$ - (state equations)
  $y_i = g_i(x_1, x_2, \ldots, x_n, u_1, u_2, \ldots, u_m) \;\; i = 1, \ldots, p$ - (output equations)

  The above state model can be written as $E : \dot{X} = F(X, U); Y = G(X, U)$.
- The state model specifies set of trajectories for each variable $v \in Z$. The state equations specify the trajectories for each $x_i \in X_a$ as the solution of the n first-order differential equations. The output equations specify the trajectories for each $y_k \in Y$ as algebraic functions of the input and the internal variables.
- The discrete internal variables remain constant over trajectories and are not specified in the state model of the system.
- The set of trajectories defined by the state model E is denoted by $[\![E]\!]$.

This arrangement of using the state model description of the system dynamics directly into describing the trajectory looks fine, but there is a slight problem ..

Activities

An **activity** α of automaton $\mathcal{A}$ consists of:

- P^-, Pre condition
- P, Operating condition
- P^+, Post condition, $\quad P^-, P$ and $P^+ \subseteq val(X)$
- E, State model of $\mathcal{A}$

$\alpha : (P^-, P, P^+, E)$ defines a set of trajectories $[\![\alpha]\!]$, such that $\tau \in [\![\alpha]\!]$ if:

- $\tau \in [\![E]\!]$,
- $\tau.fstate \lceil X \in P^-$,
- if $dom(\tau)$ is finite, then $\tau.lstate \lceil X \in P^+$,
- $\forall t \in dom(\tau)$, if $(t \neq 0 \ \wedge t \neq ltime(\tau))$, then $\tau(t) \lceil X \in P$

Augment $\mathcal{A}$ with a finite set of activities Ω which specify $\mathcal{T}$.

$\mathcal{T} = \cup_{\alpha \in \Omega} [\![\alpha]\!]$,

Input Enabling - $\cup_{\alpha \in \Omega} P^-_\alpha = val(X)$,

Nondeterminism - if $\alpha, \beta \in \Omega$, $\mathbf{v} \lceil X \in P^-_\alpha \wedge P^-_\beta$, then, trajectories starting from $\mathbf{v}$ could be either from $[\![\alpha]\!]$ or $[\![\beta]\!]$.

Script Notes: Activities

Typical hybrid systems would have several state models, not just one ! So we introduce the notion of an activity.

An *activity* α of an automaton $\mathcal{A}$ is a quadruple (P^-, P, P^+, E), where P^-, P and P^+ are sets of states and E is a state model of $\mathcal{A}$. An activity α defines a set of trajectories $[\![\alpha]\!]$, such that $\tau \in [\![\alpha]\!]$ if:

- $\tau \in [\![E]\!]$,
- $\tau.fstate\lceil X \in P^-$,
- if $dom(\tau)$ is finite, then $\tau.lstate\lceil X \in P^+$, and
- $\forall t \in dom(\tau)$, if $(t \neq 0 \ \wedge t \neq ltime(\tau))$, then $\tau(t)\lceil X \in P$

The sets P, P^- and P^+ are called the *operating condition*, the *pre-condition* and the *post-condition* of the activity. Intuitively, the operating condition is an invariant which holds throughout the trajectory excluding the first and the last states. The pre and the post conditions specify the initial and the termination conditions.

We augment the earlier definition of a hybrid automaton $\mathcal{A}$ with a set of activities $\Omega = \{\alpha_1, \ldots, \alpha_k\}$, which specify the set of trajectories of $\mathcal{A}$. The set of trajectories $\mathcal{T}$ of $\mathcal{A}$ is the union of all the trajectories defined by the activities in Ω, $i.e.$, $\mathcal{T} = \cup_{i=1}^{k} [\![\alpha_i]\!]$, with $\alpha_i = (P_i^-, P_i, P_i^+, E_i)$. The input trajectory enabling condition is satisfied by requiring Ω to satisfy : $\cup_{i=1}^{k} P_i^- = val(X)$, $i.e.$, $\forall \mathbf{v} \in val(V), \exists \alpha_i \in \Omega$ such that $\mathbf{v}\lceil X \in P_i^-$. Thus, for every possible valuation $\mathbf{v}$ of the variables in V there exists at least one activity of the automaton that specifies a possible trajectory starting from $\mathbf{v}$. Moreover, if for some $\mathbf{v} \in val(V)$, there are two activities that are enabled, $i.e.$, if for some $i, j \in \{1, \ldots, k\}$, $i \neq j$ it is the case that $P_i^- \wedge P_j^-$, then at $\mathbf{v}$ the system has two state models E_i and E_j and for constructing an execution from $\mathbf{v}$ the trajectories are chosen nondeterministically from $[\![\alpha_i]\!]$ and $[\![\alpha_j]\!]$.

HIOA Language

Static types : built-in: `Bool`, `Int`, `Nat`, `Real`, `Char`, and `String`
`tuple` and `enumeration`

Dynamic types : `differentiable` (**analog**), `discrete`

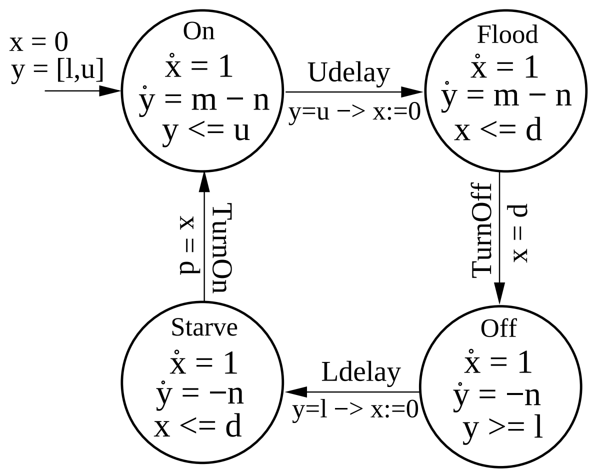
P or P^+ are assumed $val(X)$, if omitted.

P^- if omitted, is assumed to be P_i.

Script Notes: HIOA Language: Transitions & Trajectories

- **signature** declare the actions, their types parameters and values of the parameter for which they are defined
- **variables** declares the variables, their type, dytype, initial values (in case of locally controlled vars)
- **transitions** section specifies at least one transition definition for every action declared in the signature list. Each action entry may have several corresponding transitions; for example, the signature $I(m_1 : M_1, \ldots, m_k : M_k)$ **where** $IPred(m_1, \ldots, m_k)$ defines input actions for all values of $m_1, \ldots, m_k$ satisfying $IPred_i$. Corresponding to these actions, there are transition definitions for $I_i(m_1, \ldots, m_k)$ in the **transitions** section.
- The precondition is a set of states in which the transition is enabled. In its absence the transition is enabled in all states, as in input actions.
- The *effect* clause defines the state transition relation $\mathbf{x} \stackrel{a}{\rightarrow} \mathbf{x}'$ corresponding to each action $a \in A$. The effects are written as a sequence of conditionals and assignments(possibly nondeterministic) to locally controlled variables. Variable assignments for transitions are instantaneous, sequential, and without side-effects.
- The **trajectories** section defines the state models of the automaton in terms of activities.
- If the operating condition or the post condition are omitted from the declaration of an activity, then they are assumed to be `true` denoting the entire state space $val(X)$. Whenever the pre-condition is omitted, it is assumed to be same as the operating condition P_i.
- The state model of an activity is represented in the form of first-order differential equations and algebraic equations. The expressions on the right hand sides of assignments are either algebraic expressions involving internal and input variables or nondeterministic choice expressions.

Very Simple Example : Water Level Controller



Parameters : Upper(u) and lower(l) level sensors,
In-flow(m) and out-flow rates(n), delay(d)

Variables : Time (x), Level (y)

Actions : Udelay, Ldelay, TurnOn, TurnOff

Activities : On, Flood, Off, Starve

Script Notes: LEVEL

Very simple example, warming up, won't spend much time

Alur's notation for hybrid systems: circles locations - set of differential equations in our case they become activities

Lines are discrete transitions : actions

Composition

Keyword **compose** to define compositions

Components $A_i(p_1 : P_1, \ldots, p_{a_i} : P_{a_i})$, $i = 1 \ldots n$, instantiated appropriately

$e.g.$, **hybridautomaton** $\mathcal{A}$
compose
$A_1(p_{11}, \ldots, p_{1a_1})$
$\ldots$
$A_n(p_{n1}, \ldots, p_{na_n})$

Composition Transformations

Signatures, Variables, and Transitions form IOA

Trajectories

Script Notes: Composition in HIOA

In order to specify more interesting systems , we need HIOA to specify composite automata.

- follows from the composition operation defined for H IOA
- the components are instantiated by supplying actual parameters
- Most of the analysis methods are geared towards primitive automata. so we need a way of converting composite automaton to primitive ones..as you can imagine this will result in an explosion in the number of states etc... so this demonstrates the need for automating such routine procedures.
- at the same time must mention that implementing a composer has been found to be quite a difficult task.
- the transformation rules for **signature** , **transitions** are same as IOA
- for transforming the activities we need to merge state models

Merging State Models

Merging of state models required for composing automata.

Disjointedness: State models
$E_1 : \dot{X}_1 = F_1(X_1, U_1); Y_1 = G_1(X_1, U_1)$ and
$E_2 : \dot{X}_2 = F_2(X_2, U_2); Y_2 = G_2(X_2, U_2)$
are *disjoint* if $(X_1 \cup Y_1) \ \cap \ (X_2 \cup Y_2) = \emptyset$.

State models of compatible automata are disjoint

Merging of two disjoint state models E_1 and E_2 is:

$$\begin{aligned} E_1 \otimes E_2 : & \dot{X}_1 = F_1(X_1, U_1); \dot{X}_2 = F_2(X_2, U_2); \\ & Y_1 = G_1(X_1, U_1); Y_2 = G_2(X_2, U_2). \end{aligned}$$

$$\begin{aligned} Y &= Y_1 \cup Y_2 \\ X &= X_1 \cup X_2 \\ U &= (U_1 \cup U_2) - (Y_1 \cup Y_2). \end{aligned}$$

Script Notes: Merging State Models

- Merging state models required to define composition of two hybrid automata.
- Disjointedness : State models $E_1 : \dot{X}_1 = F_1(X_1, U_1); Y_1 = G_1(X_1, U_1)$ and $E_2 : \dot{X}_2 = F_2(X_2, U_2); Y_2 = G_2(X_2, U_2)$ are *disjoint* if the set of dependent variables in their state and output equations are disjoint, $i.e.$, $(X_1 \cup Y_1) \cap (X_2 \cup Y_2) = \emptyset$.
- State models of compatible automata are disjoint
- *Merging* of two disjoint state models E_1 and E_2 defined as as a new state model:

  $$E_1 \otimes E_2 : \dot{X}_1 = F_1(X_1, U_1); \dot{X}_2 = F_2(X_2, U_2); Y_1 = G_1(X_1, U_1); Y_2 = G_2(X_2, U_2).$$

  The set of output and the state variables of $E_1 \otimes E_2$ are $Y_1 \cup Y_2$ and $X_1 \cup X_2$ respectively and the set of input variables is $(U_1 \cup U_2) - (Y_1 \cup Y_2)$.

Composition Transformations for Trajectories

$\mathcal{A}$ composed of A_i components $1 \leq i \leq n$.

Each A_i has h_i activities, $\alpha_{ij} : (P^-_{ij}, P_{ij}, P^+_{ij}, E_{ij})$, $i = 1 \ldots n$, $j = 1 \ldots h_i$.

$\alpha_{ij} : (P^-_{ij}, P_{ij}, P^+_{ij}, E_{ij})$ of A_i
$\alpha_{kl} : (P^-_{kl}, P_{kl}, P^+_{kl}, E_{kl})$ of A_k

$\beta_{jl} : (P^-_{jl}, P_{jl}, P^+_{jl}, E_{jl})$ of $B_{ik} = A_i || A_k$

- $P^-_{jl} = P^-_{ij} \wedge P^-_{kl}$,
- $P^+_{jl} = P^+_{ij} \wedge P^+_{kl}$,
- $P_{jl} = P_{ij} \wedge P_{kl}$,
- $E_{jl} = E_{ij} \otimes E_{kl}$.

Script Notes: Composition Transformation for Trajectories

- $\mathcal{A}$ composed of A_i components $1 \leq i \leq n$.
- Each A_i has h_i activities, $\alpha_{ij} : (P^-_{ij}, P_{ij}, P^+_{ij}, E_{ij})$, $i = 1 \ldots n$, $j = 1 \ldots h_i$.
- The trajectories of $\mathcal{B}$ are obtained by combining the α_{ij} activities of its component automata A_i. This combination is carried out in steps, each of which combines the activities of two components.
- Activities of A_i and A_j are merged to create $h_i \times h_j$ activities of their composition $A_i||A_j$. The activity $\alpha_{ij} : (P^-_{ij}, P_{ij}, P^+_{ij}, E_{ij})$ of automaton A_i is merged with activity $\alpha_{kl} : (P^-_{kl}, P_{kl}, P^+_{kl}, E_{kl})$ of automaton A_k to form the activity $\beta_{jl} : (P^-_{jl}, P_{jl}, P^+_{jl}, E_{jl})$ of the composed automaton $B_{ik} = A_i||A_k$ as follows:
  - $P^-_{jl} = P^-_{ij} \wedge P^-_{kl}$,
  - $P^+_{jl} = P^+_{ij} \wedge P^+_{kl}$,
  - $P_{jl} = P_{ij} \wedge P_{kl}$,
  - $E_{jl} = E_{ij} \otimes E_{kl}$.
- The components A_i and A_k must be compatible in order to be composed and compatibility ensures that $Z_{A_i} \cap Z_{A_k} = \emptyset$. Hence the state models E_{ij} and E_{kl} are disjoint and their merging $E_{ij} \otimes E_{kl}$ is defined (see Section **??**).

Simple Example: Controlled Deceleration

Vehicle initial velocity(vi), (ai = 0)

Target velocity range [vfmin,vfmax], distance (sf)

Vehicle modes

braking mode, $a \in [amin, amax]$

non-braking mode, $a = 0$

Assumptions about controller:

No delay

One Shot

Further Requirements:

(1) $si < sf$

(2) $vi > vfmax \geq vfmin > 0$

(3) $ai = 0$

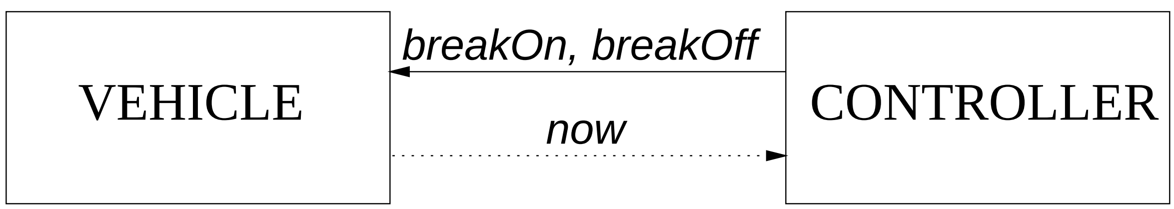
(4) $amin \leq amax < 0$

(5) $sf - si \geq \frac{vfmax^2 - vi^2}{2amax}$

(6) $\frac{vfmax - vi}{amax} \leq \frac{vfmin - vi}{amin}$

Script Notes: Simple Example: Controlled Deceleration[from Weinberg]

- The fifth condition states that the weakest deceleration can bring the vehicle to the highest allowed velocity in the given distance.
- The sixth condition ensures that the least amount of time for which the controller must brake is less than the greatest amount of time for which the controller can brake.



Script Notes: CONTROLLER

- **Timeliness property**: In every hybrid execution : $\exists\, t \geq \mathbb{R}^{\geq 0}$ such that, if $now = t$ for some state, then there is a state of the trace at which $s = sf$.

Safe Controller

Instance of `CONTROLLER` is $safe$ wrt an instance of the `VEHICLE` if their composition hybrid I/O automaton satisfies :

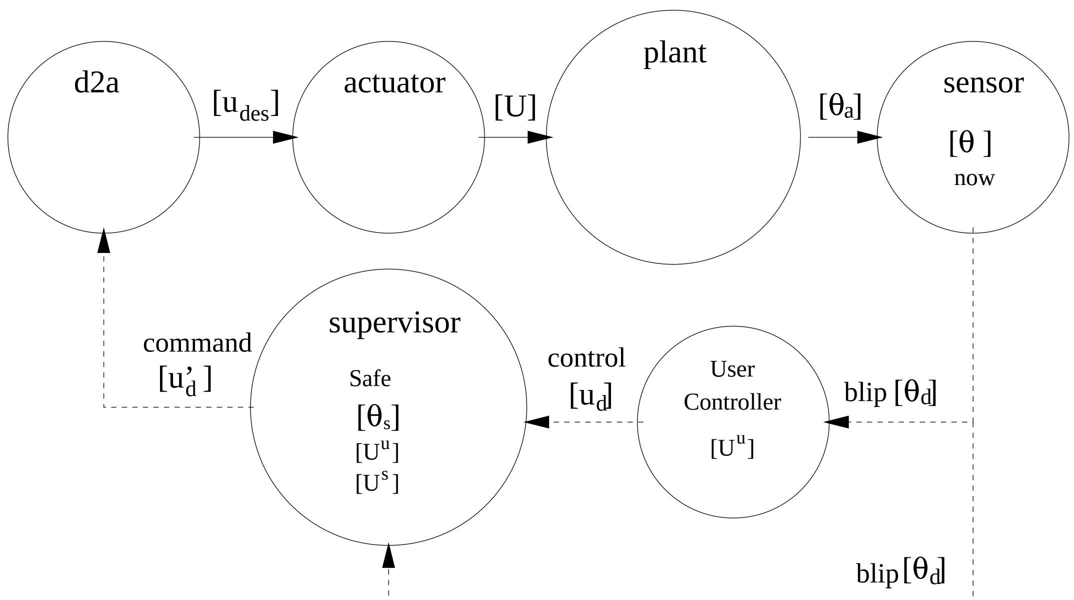
$$s = sf \implies vfmin \leq v \leq vfmax.$$

$$A = \frac{1}{vi}\left(sf - si - \frac{vfmax^2 - vi^2}{2amax}\right) \quad B = \frac{vfmax - vi}{amax} \quad C = \frac{vfmin - vi}{amin}$$

VEHICLE and Safe CONTROLLER composed automaton : `SYSTEM`

`SYSTEM` $\stackrel{comp.tfm.}{\longrightarrow}$ `SYS_PRIMITIVE`.

VI Quanser System Model



$[u_d]$: discrete controlling inputs $\langle u_d, v_d \rangle$
$[u_a]$: analog/continuous controlling input to plant $\langle u_a, v_a \rangle$
$[\theta_a]$: continuous output $\langle \theta_a^0, \phi_a^0, \psi_a^0, \theta_a^1, \phi_a^1, \psi_a^1 \rangle$
$[\theta_d]$: discretized outputs $\langle \theta_d^0, \ldots \rangle$
blip : communicates the value of θ_d to controller and supervisor
control : user controller output with controlling input u_d
command : supervisor output u'_d

Script Notes: Quanser System Model

- SENSOR
  - Delay ignored fro the time
  - Can create a sensor with any sampling rate with parameter Δ
  - Accuracy can be controlled by ϵ
  - Single activity, set of algebraic equations, dynamics not interesting just copies the input from the plan