Interchange Semantics For Hybrid System Models

Alessandro Pinto[1], Luca P. Carloni[2], Roberto Passerone[3] and Alberto Sangiovanni-Vincentelli[1]

[1] University of California at Berkeley

apinto,alberto@eecs.berkeley.edu

[2] Columbia University

luca@cs.columbia.edu

[3] University of Trento, Italy

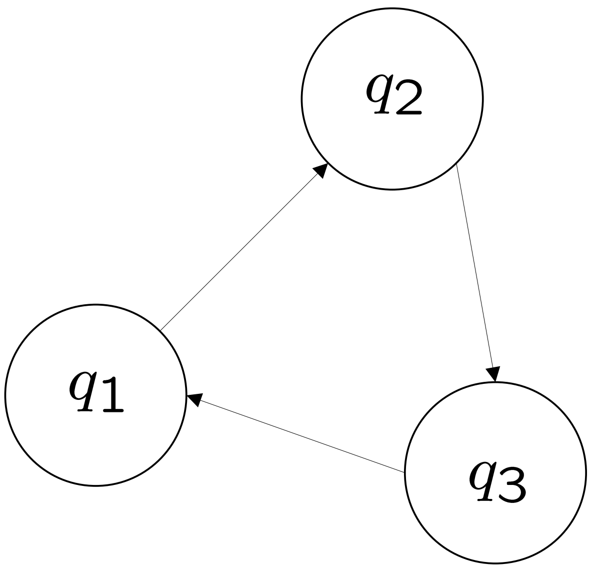
roby@dit.unitn.it

March 3, 2006

Hybrid systems

A hybrid system is a tuple

$$\mathcal{H} = (\mathbf{Q}, \mathbf{U}_D, E, X, U, V, \mathcal{S}, Inv, R, G)$$



$$\mathcal{S}(q_1) = (\dot{x} = f(x, u, t))$$

$$G(q_1, q_2) = (x_1 \geq x_2 \wedge v_1 > v_2)$$

$$Inv(q_1) = (\bar{G}(q_1, q_2))$$

$$R(q_1, q_2) = (x' = r(x, u))$$

Hybrid system semantics

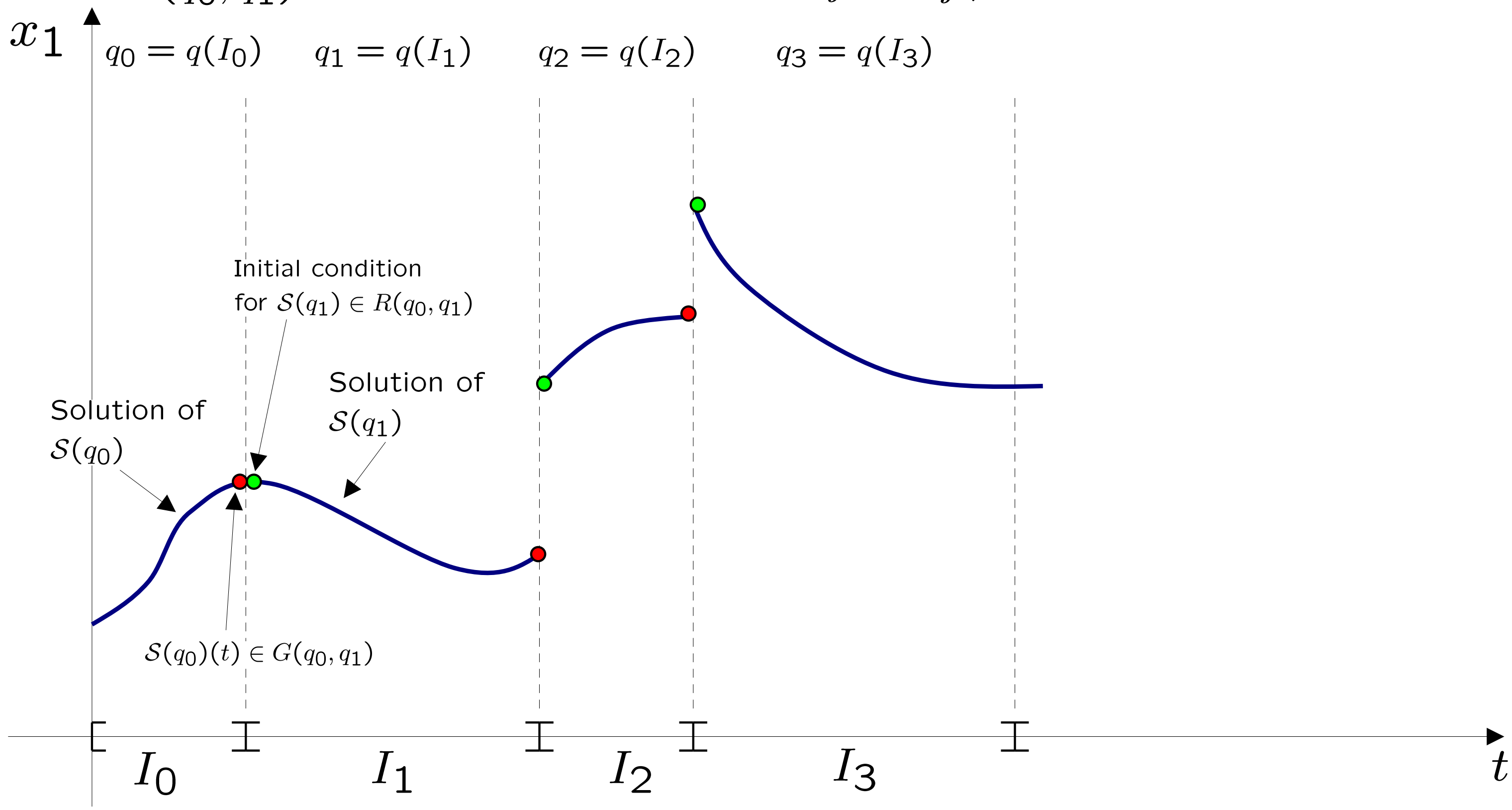
A hybrid time basis τ is a finite or an infinite sequence of intervals

$$I_j = \{t \in \mathbb{R} : t_j \leq t \leq t'_j\}$$

where $t_j \leq t'_j$ and $t'_j = t_{j+1}$

Hybrid systems execution

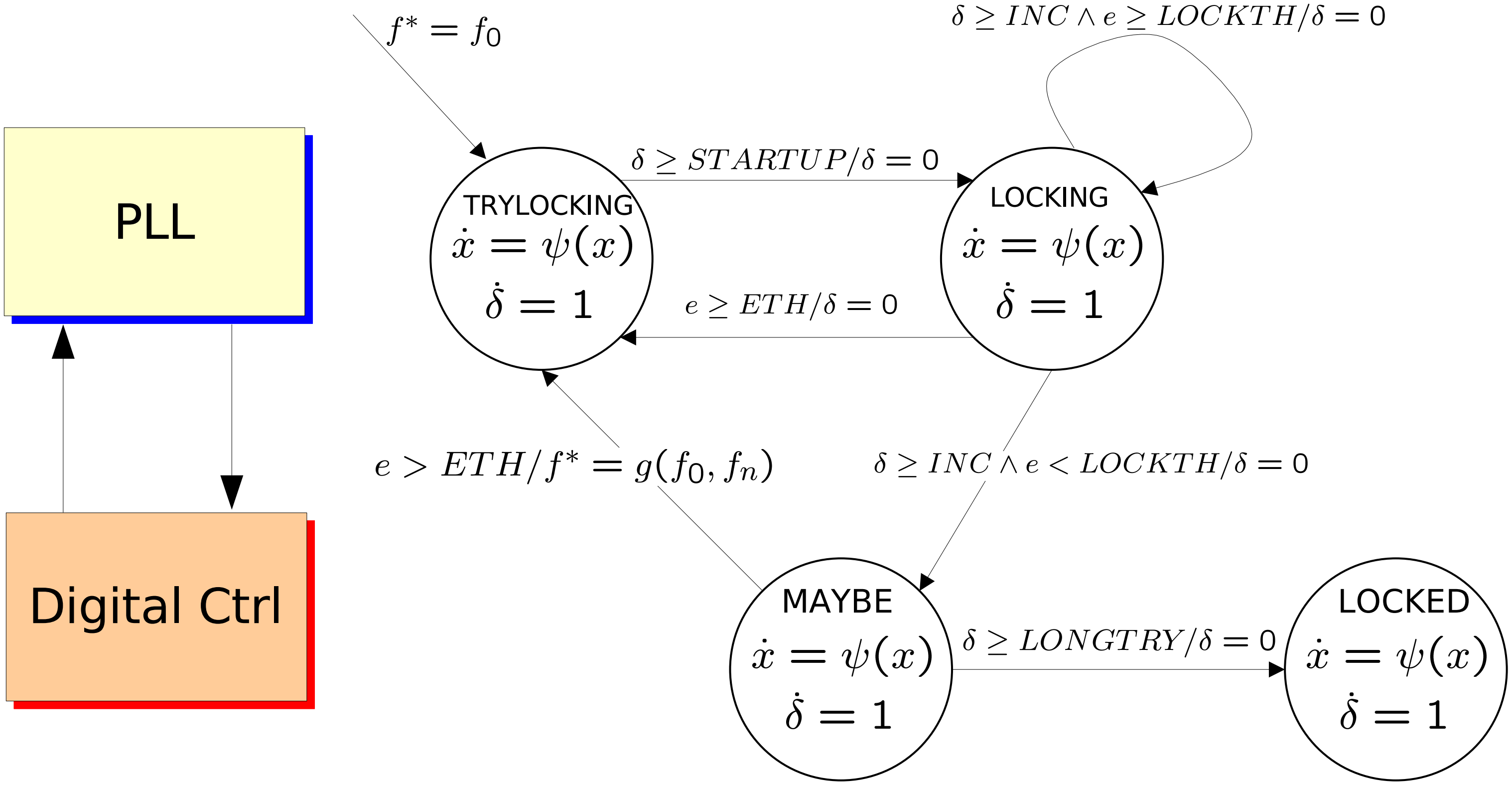
$(q_0, q_1) \in E$ $\qquad$ $(q(I_j), q(I_{j+1})) \in E, \quad \forall j$



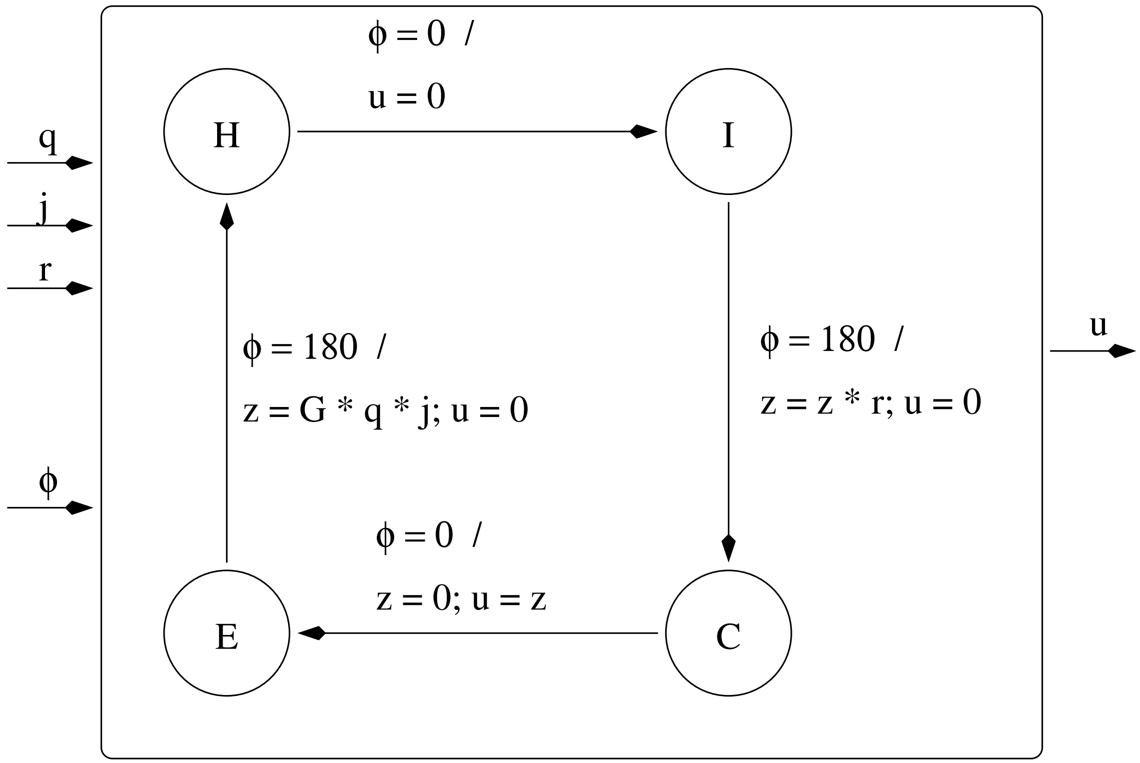
Applications

- Abstract uninteresting dynamics

Digitally controlled PLL



A car engine



Need for an interchange format

- Hybrid Systems (HS) have proven to be a powerful design representation for system -level design.
- There has been a proliferation of tool for simulation, verification and synthesis of HS but...
- ... all based on different models.
- The need for and interchange format (IF) is very much felt.

Need for an interchange format

- Hybrid Systems (HS) have proven to be a powerful design representation for system -level design.
- There has been a proliferation of tool for simulation, verification and synthesis of HS but...
- ... all based on different models.
- The need for and interchange format (IF) is very much felt.

- We presented a proposal for an IF (HSCC2005).
- We have defined its semantics (HSCC2006).
- Here we summarize our findings and justify our choices.

Outline

1. Interchange Formats in EDA
2. Interchange Format Syntax and Abstract Semantics
3. Composition and Hierarchy
4. Conclusions

Interchange Formats in EDA

Interchange Formats in EDA: a Brief History

- Electronic Design Interchange Format (EDIF)

Interchange Formats in EDA: a Brief History

- Electronic Design Interchange Format (EDIF)

- Library Exchange Format / Design Exchange Format (LEF / DEF)

Interchange Formats in EDA: a Brief History

- Electronic Design Interchange Format (EDIF)
- Library Exchange Format / Design Exchange Format (LEF / DEF)
- Berkeley Logic Interchange Format (BLIF)

Interchange Formats in EDA: a Brief History

- Electronic Design Interchange Format (EDIF)
- Library Exchange Format / Design Exchange Format (LEF / DEF)
- Berkeley Logic Interchange Format (BLIF)
- Hybrid System Interchange Format (HSIF)

Interchange Format: Approaches

- An interchange format only defines the syntax of a common data structure (first version of EDIF):
  - very flexible,
  - ambiguous.

Interchange Format: Approaches

- An interchange format only defines the syntax of a common data structure (first version of EDIF):
  - very flexible,
  - ambiguous.
- An interchange Format defines the common model of computation (BLIF,HSIF):
  - unambiguous. BLIF uses a model that is universal for logic but it is used for a restricted domain (boolean algebra). HSIF allows direct analysis of models, but in HS there is a great degree of semantic differences across tools.
  - Not flexible. It reduces the degree of freedom of the tools that share the date using the format.

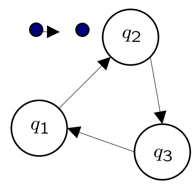
Interchange Format: Approaches

- An interchange format only defines the syntax of a common data structure (first version of EDIF):
  - very flexible,
  - ambiguous.
- An interchange Format defines the common model of computation (BLIF,HSIF):
  - unambiguous. BLIF uses a model that is universal for logic but it is used for a restricted domain (boolean algebra). HSIF allows direct analysis of models, but in HS there is a great degree of semantic differences across tools.
  - Not flexible. It reduces the degree of freedom of the tools that share the date using the format.
- An interchange format has a formal *abstract semantics* that can be refined into *concrete semantics*
  - An interchange format must be capable of capturing the largest possible class of models in use today and even tomorrow
  - At the same time has to have precise semantics to avoid ambiguity.

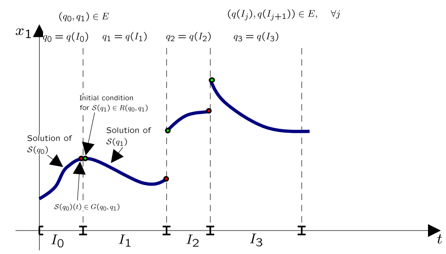
The Big Picture

$$\mathcal{H} = (\mathbf{Q}, \mathbf{U}_D, E, X, U, V, \mathcal{S}, Inv, R, G)$$

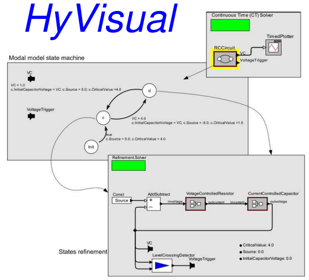
HS-Denotational Description



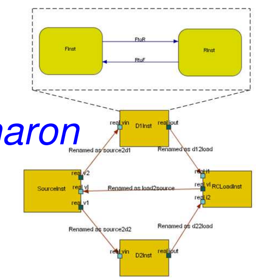
$$\mathcal{S}(q_1) = (\dot{x} = f(x,u,t))$$
$$G(q_1,q_2) = (x_1 \geq x_2 \wedge v_1 > v_2)$$
$$Inv(q_1) = (\bar{G}(q_1,q_2))$$
$$R(q_1,q_2) = (x' = r(x,u))$$



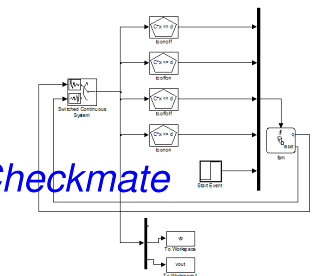
HyVisual



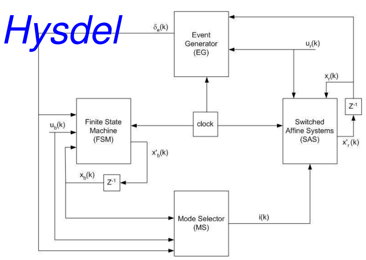
Charon

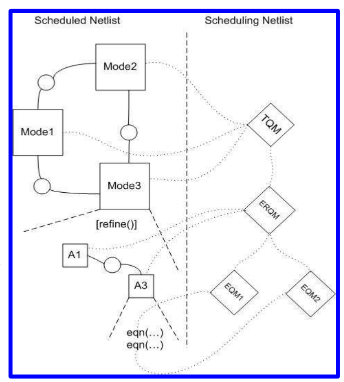


Checkmate



Hysdel





Interchange Format

Interchange Format Syntax and Abstract Semantics

Preliminary Definitions

Valuations of variables : Given a variable with name v, its value is denoted by $val(v)$.

Valuation of tuples of variables : If V is the tuple $(v_1, ..., v_n)$ then $val(V) = (val(v_1), ..., val(v_n))$.

Valuation of sets of variables : If V is the set $\{v_1, ..., v_n\}$ then its valuation is the multi-set $val(V) = \{val(v_1), ..., val(v_n)\}$.

Valuations domain : For a set of variables V, the set of all possible valuations of V is denoted by $\mathcal{R}(V)$.

Lifting : Given a subset $D \subseteq \mathcal{R}(V)$ and $V' \supseteq V$, the lifting of D to V' is given by the operator $\mathcal{L}(V')(D) = \{p' \in \mathcal{R}(V') : p'|_V \in \mathcal{R}V\}$.

Definition of a Hybrid System: Syntax

A hybrid system is a tuple $H = (V, E, \mathcal{D}, I, \sigma, \omega, \rho)$ where:

Definition of a Hybrid System: Syntax

A hybrid system is a tuple $H = (V, E, \mathcal{D}, I, \sigma, \omega, \rho)$ where:

- $V = \{v_1, ..., v_n\}$ is a set of variables,

Definition of a Hybrid System: Syntax

A hybrid system is a tuple $H = (V, E, \mathcal{D}, I, \sigma, \omega, \rho)$ where:

- $V = \{v_1, ..., v_n\}$ is a set of variables,
- $E = \{e_1, ..., e_m\}$ is a set of equations in the variables V,

Definition of a Hybrid System: Syntax

A hybrid system is a tuple $H = (V, E, \mathcal{D}, I, \sigma, \omega, \rho)$ where:

- $V = \{v_1, ..., v_n\}$ is a set of variables,
- $E = \{e_1, ..., e_m\}$ is a set of equations in the variables V,
- $\mathcal{D} \subseteq 2^{\mathcal{R}(V)}$ is a set of domains, or regions, of the possible valuations of the variables V,

Definition of a Hybrid System: Syntax

A hybrid system is a tuple $H = (V, E, \mathcal{D}, I, \sigma, \omega, \rho)$ where:

- $V = \{v_1, ..., v_n\}$ is a set of variables,
- $E = \{e_1, ..., e_m\}$ is a set of equations in the variables V,
- $\mathcal{D} \subseteq 2^{\mathcal{R}(V)}$ is a set of domains, or regions, of the possible valuations of the variables V,
- $I \subseteq \mathbb{N}$ is a set of indexes,
- $\sigma : 2^{\mathcal{R}(V)} \longrightarrow 2^I$ is a function that associates a set of indexes to each domain
- $\omega : I \longrightarrow 2^E$ is a function that associates a set of equations to each index,

Definition of a Hybrid System: Syntax

A hybrid system is a tuple $H = (V, E, \mathcal{D}, I, \sigma, \omega, \rho)$ where:

- $V = \{v_1, \ldots, v_n\}$ is a set of variables,
- $E = \{e_1, \ldots, e_m\}$ is a set of equations in the variables V,
- $\mathcal{D} \subseteq 2^{\mathcal{R}(V)}$ is a set of domains, or regions, of the possible valuations of the variables V,
- $I \subseteq \mathbb{N}$ is a set of indexes,
- $\sigma : 2^{\mathcal{R}(V)} \longrightarrow 2^I$ is a function that associates a set of indexes to each domain
- $\omega : I \longrightarrow 2^E$ is a function that associates a set of equations to each index,
- $\rho : 2^{\mathcal{R}(V)} \times 2^{\mathcal{R}(V)} \times \mathcal{R}(V) \longrightarrow 2^{\mathcal{R}(V)}$ is a function to reset the values of the variables.

Definition of a Hybrid System: Syntax

A hybrid system is a tuple $H = (V, E, \mathcal{D}, I, \sigma, \omega, \rho)$ where:

- $V = \{v_1, ..., v_n\}$ is a set of variables,
- $E = \{e_1, ..., e_m\}$ is a set of equations in the variables V,
- $\mathcal{D} \subseteq 2^{\mathcal{R}(V)}$ is a set of domains, or regions, of the possible valuations of the variables V,
- $I \subseteq \mathbb{N}$ is a set of indexes,
- $\sigma : 2^{\mathcal{R}(V)} \longrightarrow 2^I$ is a function that associates a set of indexes to each domain
- $\omega : I \longrightarrow 2^E$ is a function that associates a set of equations to each index,
- $\rho : 2^{\mathcal{R}(V)} \times 2^{\mathcal{R}(V)} \times \mathcal{R}(V) \longrightarrow 2^{\mathcal{R}(V)}$ is a function to reset the values of the variables.
- $V_t = \{v_{t1}, ..., v_{tn}\}$ is a set of temporary variables,
- $\pi : E \longrightarrow \{1, 2, \ldots, |E|\}$ is an equation ordering function.

Syntax Example

A bouncing ball can be modeled as a hybrid system with $V = \{y, v\}$, $E = \{\dot{v} = -g, \dot{y} = v\}$. There are two domains: $D_1 = \{\{val(y), val(v)\} : val(y) \leq 0 \wedge val(v) < 0\}$ and $D_2 = \{\{val(y), val(v)\} : val(y) > 0\}$, hence $\mathcal{D} = \{D_1, D_2\}$; $I = \{1\}$, $\sigma(D_1) = \sigma(D_2) = \{1\}$, $\omega(1) = E$. The reset function is defined as follows: $\rho(D_2, D_1, val(V)) = \{val(y), -\epsilon val(v)\}$.

Syntax Example

A bouncing ball can be modeled as a hybrid system with $V = \{y, v\}$, $E = \{\dot{v} = -g, \dot{y} = v\}$. There are two domains: $D_1 = \{\{val(y), val(v)\} : val(y) \leq 0 \wedge val(v) < 0\}$ and $D_2 = \{\{val(y), val(v)\} : val(y) > 0\}$, hence $\mathcal{D} = \{D_1, D_2\}$; $I = \{1\}$, $\sigma(D_1) = \sigma(D_2) = \{1\}$, $\omega(1) = E$. The reset function is defined as follows: $\rho(D_2, D_1, val(V)) = \{val(y), -\epsilon val(v)\}$.

Syntax Example

A bouncing ball can be modeled as a hybrid system with $V = \{y, v\}$, $E = \{\dot{v} = -g, \dot{y} = v\}$. There are two domains: $D_1 = \{\{val(y), val(v)\} : val(y) \leq 0 \wedge val(v) < 0\}$ and $D_2 = \{\{val(y), val(v)\} : val(y) > 0\}$, hence $\mathcal{D} = \{D_1, D_2\}$; $I = \{1\}$, $\sigma(D_1) = \sigma(D_2) = \{1\}$, $\omega(1) = E$. The reset function is defined as follows: $\rho(D_2, D_1, val(V)) = \{val(y), -\epsilon val(v)\}$.

Syntax Example

A bouncing ball can be modeled as a hybrid system with $V = \{y, v\}$, $E = \{\dot{v} = -g, \dot{y} = v\}$. There are two domains: $D_1 = \{\{val(y), val(v)\} : val(y) \leq 0 \wedge val(v) < 0\}$ and $D_2 = \{\{val(y), val(v)\} : val(y) > 0\}$, hence $\mathcal{D} = \{D_1, D_2\}$; $I = \{1\}$, $\sigma(D_1) = \sigma(D_2) = \{1\}$, $\omega(1) = E$. The reset function is defined as follows: $\rho(D_2, D_1, val(V)) = \{val(y), -\epsilon val(v)\}$.

Syntax Example

A bouncing ball can be modeled as a hybrid system with $V = \{y, v\}$, $E = \{\dot{v} = -g, \dot{y} = v\}$. There are two domains: $D_1 = \{\{val(y), val(v)\} : val(y) \leq 0 \wedge val(v) < 0\}$ and $D_2 = \{\{val(y), val(v)\} : val(y) > 0\}$, hence $\mathcal{D} = \{D_1, D_2\}$; $I = \{1\}$, $\sigma(D_1) = \sigma(D_2) = \{1\}$, $\omega(1) = E$. The reset function is defined as follows: $\rho(D_2, D_1, val(V)) = \{val(y), -\epsilon val(v)\}$.

Syntax Example

A bouncing ball can be modeled as a hybrid system with $V = \{y, v\}$, $E = \{\dot{v} = -g, \dot{y} = v\}$. There are two domains: $D_1 = \{\{val(y), val(v)\} : val(y) \leq 0 \wedge val(v) < 0\}$ and $D_2 = \{\{val(y), val(v)\} : val(y) > 0\}$, hence $\mathcal{D} = \{D_1, D_2\}$; $I = \{1\}$, $\sigma(D_1) = \sigma(D_2) = \{1\}$, $\omega(1) = E$. The reset function is defined as follows: $\rho(D_2, D_1, val(V)) = \{val(y), -\epsilon val(v)\}$.

Syntax Example

A bouncing ball can be modeled as a hybrid system with $V = \{y, v\}$, $E = \{\dot{v} = -g, \dot{y} = v\}$. There are two domains: $D_1 = \{\{val(y), val(v)\} : val(y) \leq 0 \wedge val(v) < 0\}$ and $D_2 = \{\{val(y), val(v)\} : val(y) > 0\}$, hence $\mathcal{D} = \{D_1, D_2\}$; $I = \{1\}$, $\sigma(D_1) = \sigma(D_2) = \{1\}$, $\omega(1) = E$. The reset function is defined as follows: $\rho(D_2, D_1, val(V)) = \{val(y), -\epsilon val(v)\}$.

Definition of a Hybrid System: Semantics

The semantics of a hybrid system H is defined by the tuple $(H, B, T, \texttt{resolve}, \texttt{init}, \texttt{update})$ where:

Definition of a Hybrid System: Semantics

The semantics of a hybrid system H is defined by the tuple $(H, B, T, \texttt{resolve}, \texttt{init}, \texttt{update})$ where:

- H is a hybrid system;

Definition of a Hybrid System: Semantics

The semantics of a hybrid system H is defined by the tuple $(H, B, T, \texttt{resolve}, \texttt{init}, \texttt{update})$ where:

- H is a hybrid system;
- B is a set of pairs (γ, t) where $\gamma \in \mathcal{R}(H.V)$ is a multi-set of possible values of the hybrid system variables and $t \in \mathbb{R}_+$ is a time stamp;

Definition of a Hybrid System: Semantics

The semantics of a hybrid system H is defined by the tuple $(H, B, T, \mathtt{resolve}, \mathtt{init}, \mathtt{update})$ where:

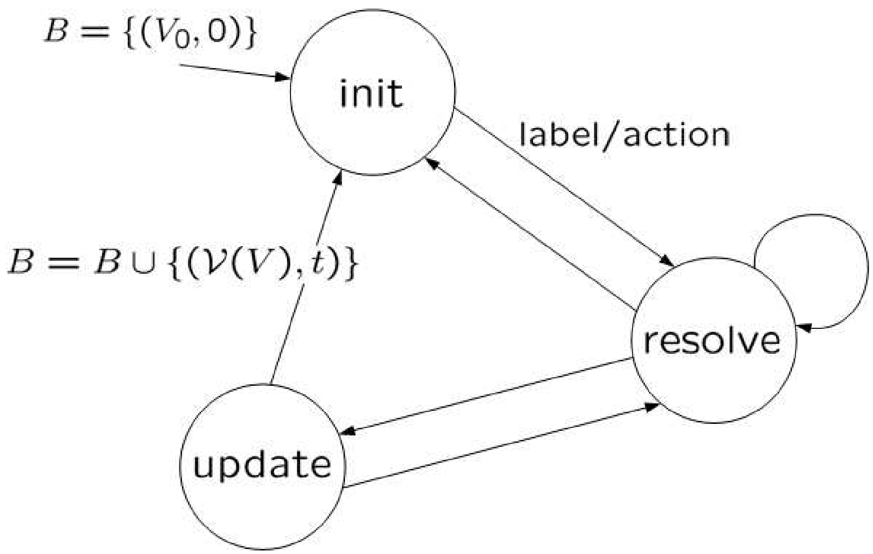
- H is a hybrid system;
- B is a set of pairs (γ, t) where $\gamma \in \mathcal{R}(H.V)$ is a multi-set of possible values of the hybrid system variables and $t \in \mathbb{R}_+$ is a time stamp;
- T is a time stamper:
  - selects the next time stamp,
  - decides whether a new pair (val, t) can be added to the set B;

Definition of a Hybrid System: Semantics

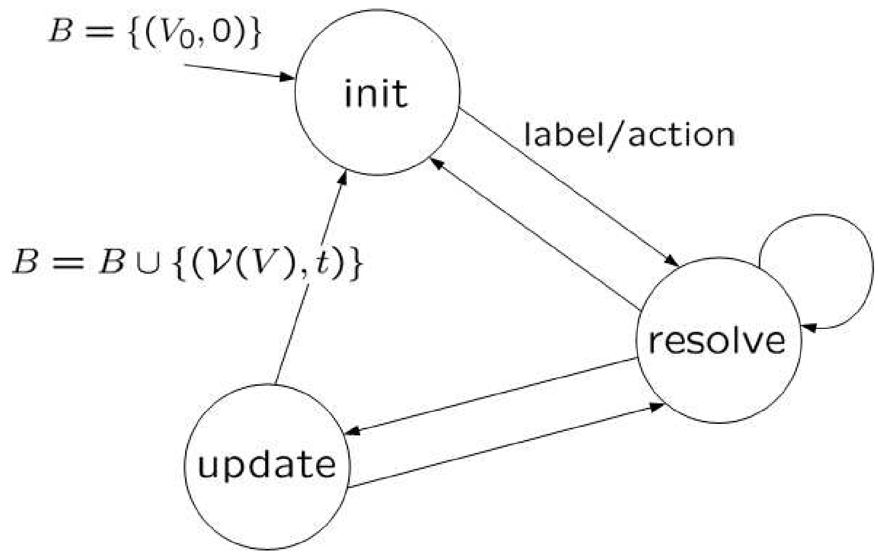
The semantics of a hybrid system H is defined by the tuple $(H, B, T, \mathtt{resolve}, \mathtt{init}, \mathtt{update})$ where:

- H is a hybrid system;
- B is a set of pairs (γ, t) where $\gamma \in \mathcal{R}(H.V)$ is a multi-set of possible values of the hybrid system variables and $t \in \mathbb{R}_+$ is a time stamp;
- T is a time stamper:
  - selects the next time stamp,
  - decides whether a new pair (val, t) can be added to the set B;
- `resolve`, `init` and `update` are three algorithms.

Time Stamper Automaton

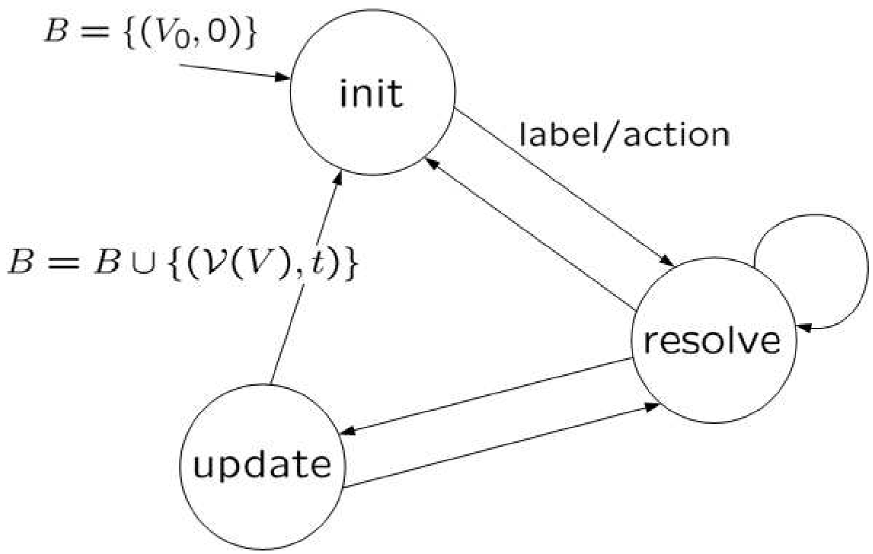


Time Stamper Automaton



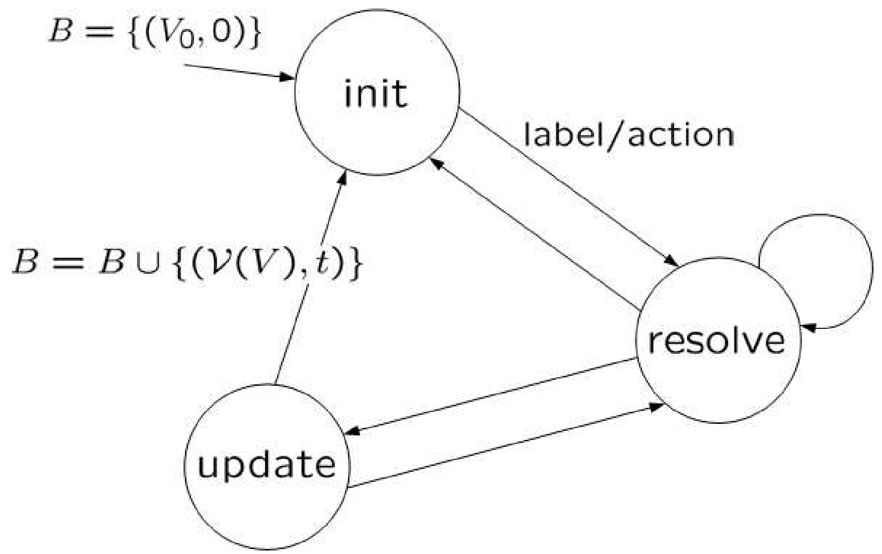
- Initialization: $B = (V_0, 0)$.

Time Stamper Automaton



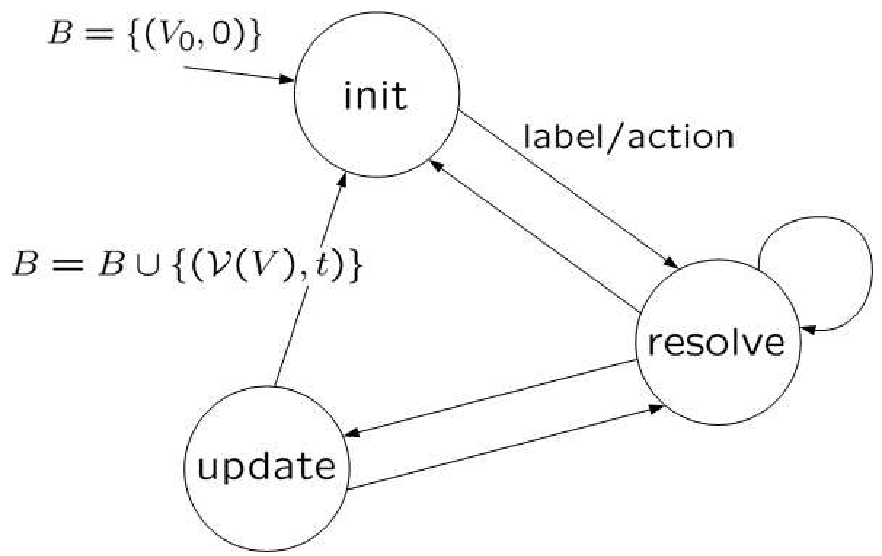
- Initialization: $B = (V_0, 0)$.
- Set of actions: `next`, `resolve`, `init`, `update`.

Time Stamper Automaton



- Initialization: $B = (V_0, 0)$.
- Set of actions: `next`, `resolve`, `init`, `update`.
- Set of conditions: `true`, `false`, error thresholds, `domainchange`.

Time Stamper Automaton



- Initialization: $B = (V_0, 0)$.
- Set of actions: `next`, `resolve`, `init`, `update`.
- Set of conditions: `true`, `false`, error thresholds, `domainchange`.
- Valuation and time stamp acceptance: $B = B \cup (\mathcal{V}(V), t)$.

Resolve Algorithm

resolve(t)

$\mathcal{D}' \Leftarrow \{D \in \mathcal{D} \mid val(V_t) \in D\}$ *// Compute the set of active domains.*

$I \Leftarrow \emptyset,\ E_t \Leftarrow \emptyset$

$I \Leftarrow \cup_{D \in \mathcal{D}'} \sigma(D)$ *// Collect all active dynamics and components.*

for all $i \in I$ **do**

  $E_t = E_t \cup \omega(i)$ *// Collect all active equations.*

end for

$\texttt{sort}(E_t, \pi)$ *// Order the equations.*

for all $e_i \in E_t$ **do**

  $\texttt{solve}(e_i, t)$

end for

$\mathcal{D}'' \Leftarrow \{D \in \mathcal{D} \mid val(V_t) \in D\}$ *// Set of active domains after the computation.*

$\texttt{markchange}(D', D'')$ *// Check if the set of active domains has changed.*

Resolve Algorithm

resolve(t)
$\mathcal{D}' \Leftarrow \{D \in \mathcal{D} \mid val(V_t) \in D\}$ *// Compute the set of active domains.*
$I \Leftarrow \emptyset,\ E_t \Leftarrow \emptyset$
$I \Leftarrow \cup_{D \in \mathcal{D}'} \sigma(D)$ *// Collect all active dynamics and components.*
for all $i \in I$ **do**
  $E_t = E_t \cup \omega(i)$ *// Collect all active equations.*
end for
$\texttt{sort}(E_t, \pi)$ *// Order the equations.*
for all $e_i \in E_t$ **do**
  $\texttt{solve}(e_i, t)$
end for
$\mathcal{D}'' \Leftarrow \{D \in \mathcal{D} \mid val(V_t) \in D\}$ *// Set of active domains after the computation.*
$\texttt{markchange}(D', D'')$ *// Check if the set of active domains has changed.*

Resolve Algorithm

resolve(t)
$\mathcal{D}' \Leftarrow \{D \in \mathcal{D} \mid val(V_t) \in D\}$ *// Compute the set of active domains.*
$I \Leftarrow \emptyset$, $E_t \Leftarrow \emptyset$
$I \Leftarrow \cup_{D \in \mathcal{D}'} \sigma(D)$ *// Collect all active dynamics and components.*
for all $i \in I$ **do**
  $E_t = E_t \cup \omega(i)$ *// Collect all active equations.*
end for
`sort`(E_t, π) *// Order the equations.*
for all $e_i \in E_t$ **do**
  `solve`(e_i, t)
end for
$\mathcal{D}'' \Leftarrow \{D \in \mathcal{D} \mid val(V_t) \in D\}$ *// Set of active domains after the computation.*
`markchange`(D', D'') *// Check if the set of active domains has changed.*

Resolve Algorithm

resolve(t)
$\mathcal{D}' \Leftarrow \{D \in \mathcal{D} \mid val(V_t) \in D\}$ *// Compute the set of active domains.*
$I \Leftarrow \emptyset,\ E_t \Leftarrow \emptyset$
$I \Leftarrow \cup_{D \in \mathcal{D}'} \sigma(D)$ *// Collect all active dynamics and components.*
for all $i \in I$ **do**
  $E_t = E_t \cup \omega(i)$ *// Collect all active equations.*
end for
$\mathtt{sort}(E_t, \pi)$ *// Order the equations.*
for all $e_i \in E_t$ **do**
  $\mathtt{solve}(e_i, t)$
end for
$\mathcal{D}'' \Leftarrow \{D \in \mathcal{D} \mid val(V_t) \in D\}$ *// Set of active domains after the computation.*
$\mathtt{markchange}(D', D'')$ *// Check if the set of active domains has changed.*

Resolve Algorithm

resolve(t)

$\mathcal{D}' \Leftarrow \{D \in \mathcal{D} \mid val(V_t) \in D\}$ *// Compute the set of active domains.*

$I \Leftarrow \emptyset,\ E_t \Leftarrow \emptyset$

$I \Leftarrow \cup_{D \in \mathcal{D}'} \sigma(D)$ *// Collect all active dynamics and components.*

for all $i \in I$ **do**

  $E_t = E_t \cup \omega(i)$ *// Collect all active equations.*

end for

`sort`$(E_t,\ \pi)$ *// Order the equations.*

for all $e_i \in E_t$ **do**

  `solve`(e_i, t)

end for

$\mathcal{D}'' \Leftarrow \{D \in \mathcal{D} \mid val(V_t) \in D\}$ *// Set of active domains after the computation.*

`markchange`(D', D'') *// Check if the set of active domains has changed.*

Resolve Algorithm

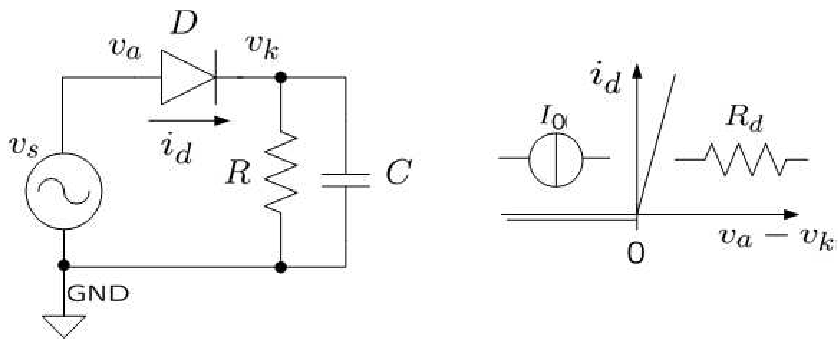
resolve(t)
$\mathcal{D}' \Leftarrow \{D \in \mathcal{D} \mid val(V_t) \in D\}$ *// Compute the set of active domains.*
$I \Leftarrow \emptyset,\ E_t \Leftarrow \emptyset$
$I \Leftarrow \cup_{D \in \mathcal{D}'} \sigma(D)$ *// Collect all active dynamics and components.*
for all $i \in I$ **do**
  $E_t = E_t \cup \omega(i)$ *// Collect all active equations.*
end for
`sort`(E_t, π) *// Order the equations.*
for all $e_i \in E_t$ **do**
  `solve`(e_i, t)
end for
$\mathcal{D}'' \Leftarrow \{D \in \mathcal{D} \mid val(V_t) \in D\}$ *// Set of active domains after the computation.*
`markchange`(D', D'') *// Check if the set of active domains has changed.*

Refinement into Concrete Semantics

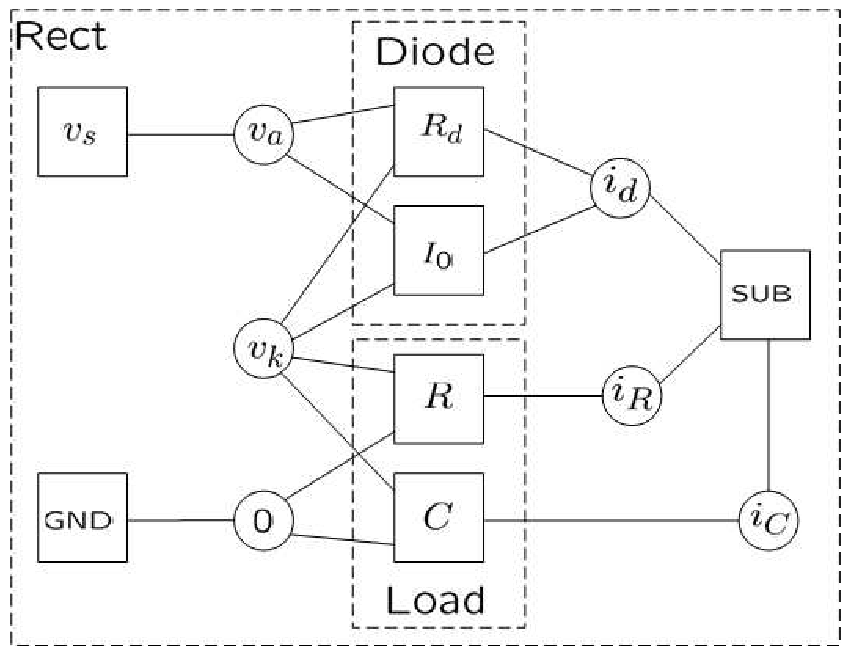
- Define the time stamper automaton: conditions and actions
  - Multiple iterations for fixed point or event detection
- Define the `next` function
  - Different algorithm to decide the next time stamp
- Define the `domainchange` function
  - Different transition semantics
- Define the `solve` function
  - Different integration methods

Composition and Hierarchy

Running Example



(a) Half-wave rectifier



(b) Block diagram of the half-wave rectifier

Composition of Hybrid Systems

Given $H_1 = (V_1, V_{t1}, E_1, \mathcal{D}_1, I_1, \sigma_1, \omega_1, \rho_1, \pi_1)$ and $H_2 = (V_2, V_{t2}, E_2, \mathcal{D}_2, I_2, \sigma_2, \omega_2, \rho_2, \pi_2)$, $H = H_1 \mid\mid H_2$ is such that:

Composition of Hybrid Systems

Given $H_1 = (V_1, V_{t1}, E_1, \mathcal{D}_1, I_1, \sigma_1, \omega_1, \rho_1, \pi_1)$ and $H_2 = (V_2, V_{t2}, E_2, \mathcal{D}_2, I_2, \sigma_2, \omega_2, \rho_2, \pi_2)$, $H = H_1 \mid\mid H_2$ is such that:

- $V = V_1 \cup V_2,\ V_t = V_{t1} \cup V_{t2},\ E = E_1 \cup E_2,\ \mathcal{D} = \mathcal{L}(V)(\mathcal{D}_1) \cup \mathcal{L}(V)(\mathcal{D}_2)$
- $I = \{1, ..., |I_1| + |I_2|\}$
- $\forall D \in 2^{\mathcal{R}(V)},\ \sigma(D) = \sigma_1(D|_{V_1}) \cup (\sigma_2 + |I_1| + 1)(D|_{V_2})$ where $(\sigma + k)(D) = \{n + k : n \in \sigma(D)\}$ is a shifting of the indexes;

Composition of Hybrid Systems

Given $H_1 = (V_1, V_{t1}, E_1, \mathcal{D}_1, I_1, \sigma_1, \omega_1, \rho_1, \pi_1)$ and $H_2 = (V_2, V_{t2}, E_2, \mathcal{D}_2, I_2, \sigma_2, \omega_2, \rho_2, \pi_2)$, $H = H_1 \mid\mid H_2$ is such that:

- $V = V_1 \cup V_2,\ V_t = V_{t1} \cup V_{t2},\ E = E_1 \cup E_2,\ \mathcal{D} = \mathcal{L}(V)(\mathcal{D}_1) \cup \mathcal{L}(V)(\mathcal{D}_2)$
- $I = \{1, ..., |I_1| + |I_2|\}$
- $\forall D \in 2^{\mathcal{R}(V)},\ \sigma(D) = \sigma_1(D|_{V_1}) \cup (\sigma_2 + |I_1| + 1)(D|_{V_2})$ where $(\sigma + k)(D) = \{n + k : n \in \sigma(D)\}$ is a shifting of the indexes;
- $$\begin{array}{ll} \omega(i) = \omega_1(i), & \text{if } 1 \leq i \leq |I_1|, \\ \omega(i) = \omega_2(i - |I_1|), & \text{if } |I_1| + 1 \leq i \leq |I_1| + |I_2| \end{array}$$

Composition of Hybrid Systems

Given $H_1 = (V_1, V_{t1}, E_1, \mathcal{D}_1, I_1, \sigma_1, \omega_1, \rho_1, \pi_1)$ and $H_2 = (V_2, V_{t2}, E_2, \mathcal{D}_2, I_2, \sigma_2, \omega_2, \rho_2, \pi_2)$, $H = H_1 \,||\, H_2$ is such that:

- $V = V_1 \cup V_2,\ V_t = V_{t1} \cup V_{t2},\ E = E_1 \cup E_2,\ \mathcal{D} = \mathcal{L}(V)(\mathcal{D}_1) \cup \mathcal{L}(V)(\mathcal{D}_2)$
- $I = \{1, ..., |I_1| + |I_2|\}$
- $\forall D \in 2^{\mathcal{R}(V)},\ \sigma(D) = \sigma_1(D|_{V_1}) \cup (\sigma_2 + |I_1| + 1)(D|_{V_2})$ where $(\sigma + k)(D) = \{n + k : n \in \sigma(D)\}$ is a shifting of the indexes;
- $\begin{array}{ll} \omega(i) = \omega_1(i), & \text{if } 1 \leq i \leq |I_1|, \\ \omega(i) = \omega_2(i - |I_1|), & \text{if } |I_1| + 1 \leq i \leq |I_1| + |I_2| \end{array}$
- $\pi(e) = \begin{cases} \pi_1(e) & if\ e \in E_1 \\ \pi_2(e) + |I_2| + 1 & if\ e \in E_2 \end{cases}$

Composition of Hybrid Systems

Given $H_1 = (V_1, V_{t1}, E_1, \mathcal{D}_1, I_1, \sigma_1, \omega_1, \rho_1, \pi_1)$ and $H_2 = (V_2, V_{t2}, E_2, \mathcal{D}_2, I_2, \sigma_2, \omega_2, \rho_2, \pi_2)$, $H = H_1 \mid\mid H_2$ is such that:

- $V = V_1 \cup V_2,\ V_t = V_{t1} \cup V_{t2},\ E = E_1 \cup E_2,\ \mathcal{D} = \mathcal{L}(V)(\mathcal{D}_1) \cup \mathcal{L}(V)(\mathcal{D}_2)$
- $I = \{1, ..., |I_1| + |I_2|\}$
- $\forall D \in 2^{\mathcal{R}(V)},\ \sigma(D) = \sigma_1(D|_{V_1}) \cup (\sigma_2 + |I_1| + 1)(D|_{V_2})$ where $(\sigma + k)(D) = \{n + k : n \in \sigma(D)\}$ is a shifting of the indexes;
- $$\begin{array}{ll} \omega(i) = \omega_1(i), & \text{if } 1 \leq i \leq |I_1|, \\ \omega(i) = \omega_2(i - |I_1|), & \text{if } |I_1| + 1 \leq i \leq |I_1| + |I_2| \end{array}$$
- $\pi(e) = \begin{cases} \pi_1(e) & if\ e \in E_1 \\ \pi_2(e) + |I_2| + 1 & if\ e \in E_2 \end{cases}$
- $$\begin{array}{ll} \rho(D_i, D_j, val(V)) = & \mathcal{L}(V)(\rho_1(D_i|_{V_1}, D_j|_{V_1}, val(V_1)) \cup \\ & \mathcal{L}(V)(\rho_2(D_i|_{V_2}, D_j|_{V_2}, val(V_2)) \end{array}$$

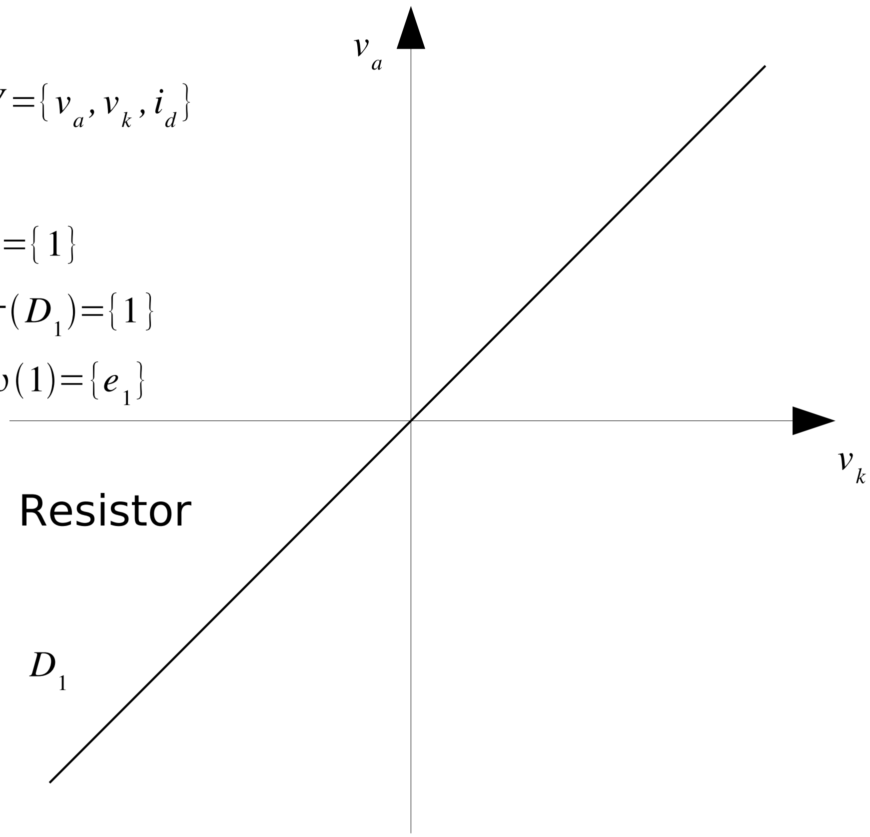
Composition Example

$V = \{v_a, v_k, i_d\}$

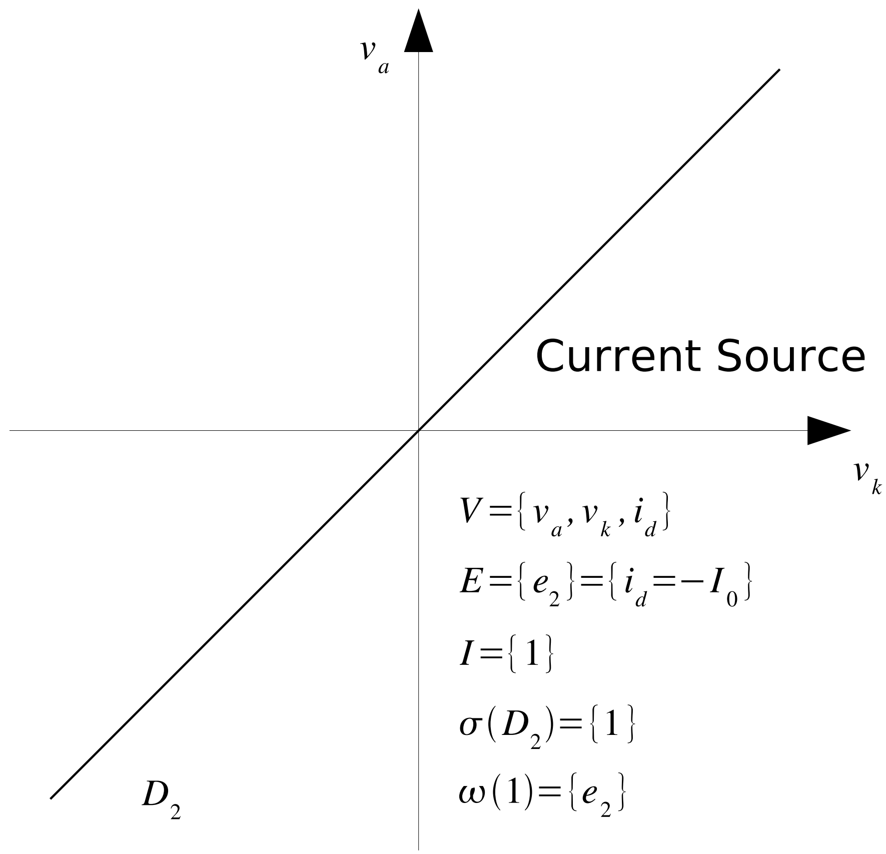
$I = \{1\}$

$\sigma(D_1) = \{1\}$

$\omega(1) = \{e_1\}$



Composition Example



$$V=\{v_a, v_k, i_d\}$$

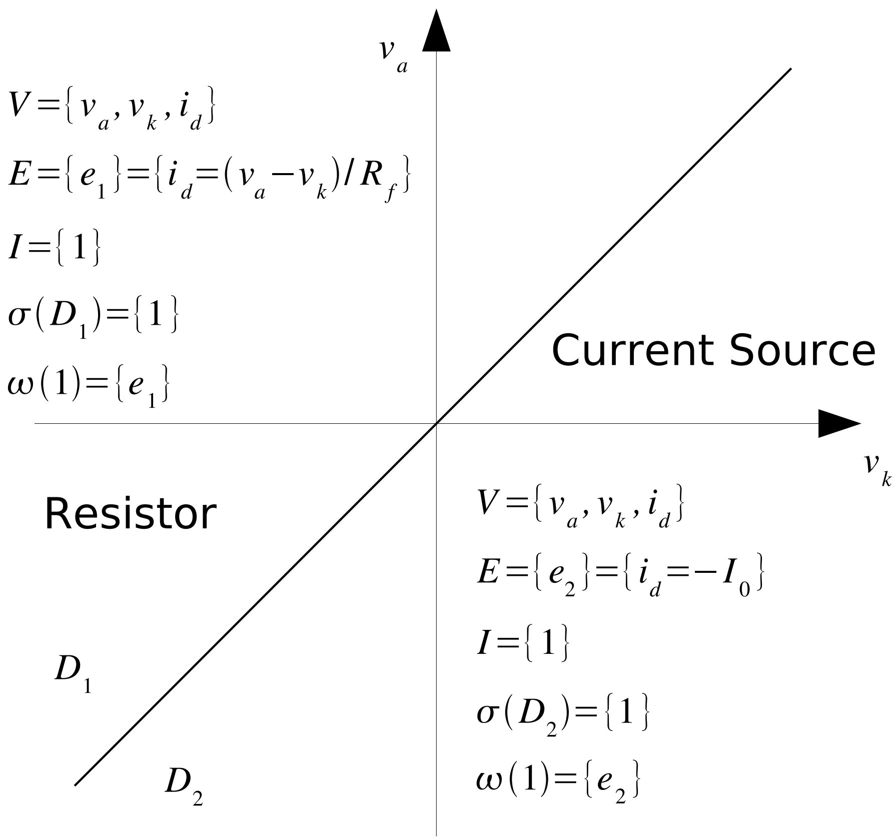
$$E=\{e_2\}=\{i_d=-I_0\}$$

$$I=\{1\}$$

$$\sigma(D_2)=\{1\}$$

$$\omega(1)=\{e_2\}$$

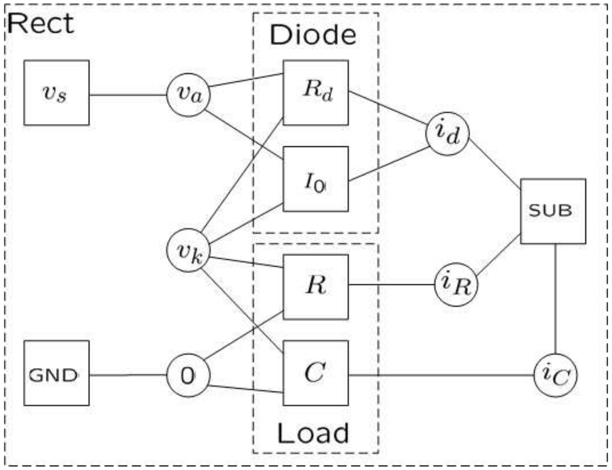
Composition Example



$diode = R_d \mid\mid I_d$

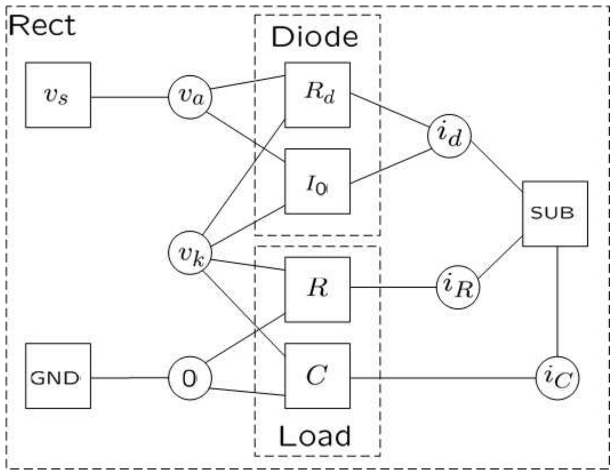
- $diode.V = \{v_a, v_k, i_d\}$,
- $diode.E = \{e_1, e_2\}$,
- $diode.\mathcal{D} = \{D_1, D_2\}$,
- $I = \{1, 2\}$,
- $diode.\sigma(D_1) = \{1\}$,
  $diode.\sigma(D_2) = \{2\}$,
- $\omega(1) = e_1$, $\omega(2) = e_2$,

Representing Hierarchy

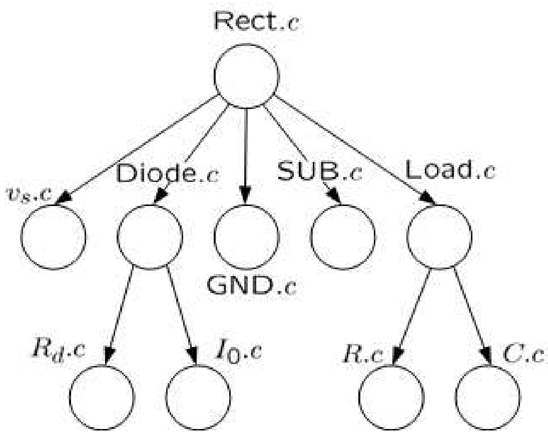


(c) $Rect = v_s \,||\, (R_d \,||\, I_0) \,||\, \mathrm{GND} \,||\, \mathrm{SUB} \,||\, (R \,||\, C)$

Representing Hierarchy

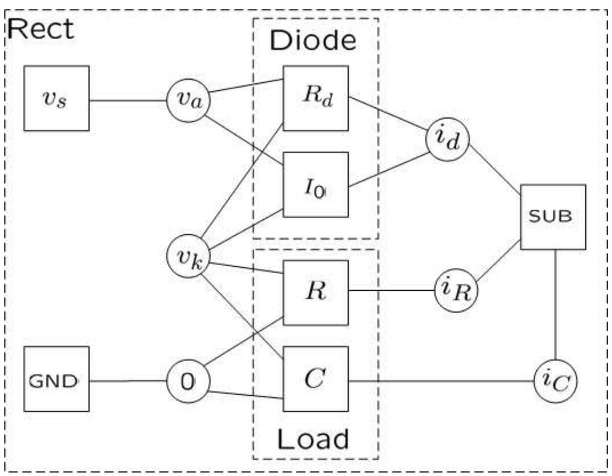


(c) $Rect = v_S \,||\, (R_d \,||\, I_0) \,||\, \mathrm{GND} \,||\, \mathrm{SUB} \,||\, (R \,||\, C)$

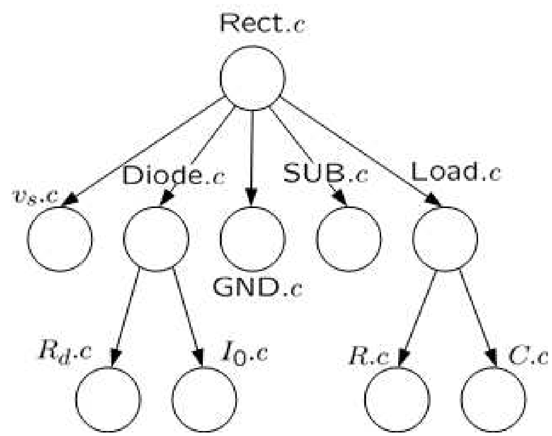


(d) Component Tree

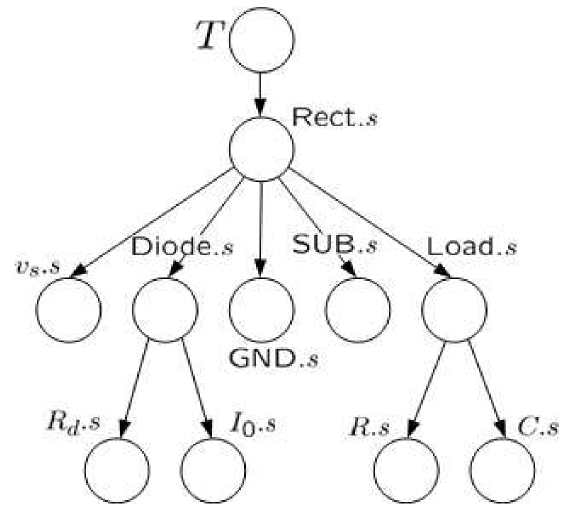
Representing Hierarchy



(c) $Rect = v_s \,||\, (R_d \,||\, I_0) \,||\, \mathrm{GND} \,||\, \mathrm{SUB} \,||\, (R \,||\, C)$

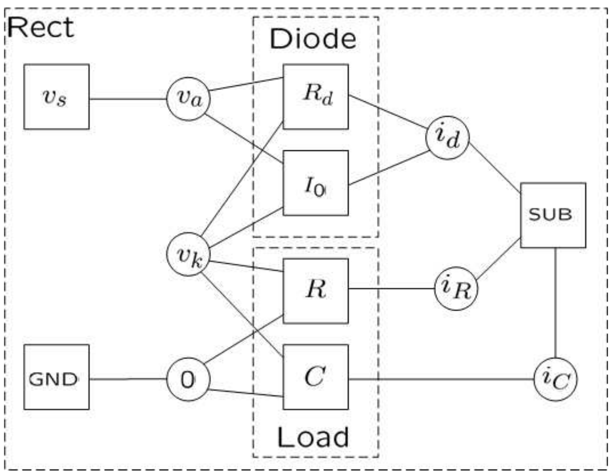


(d) Component Tree

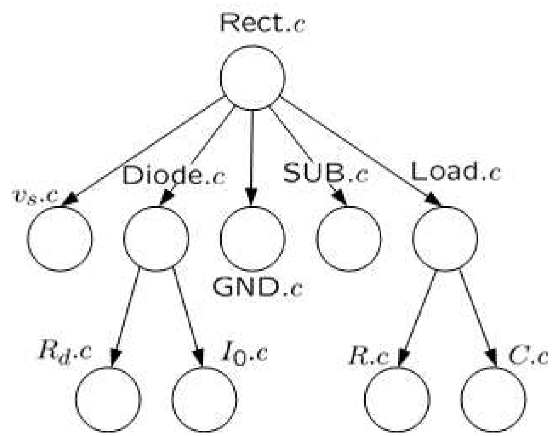


(e) Scheduler Tree

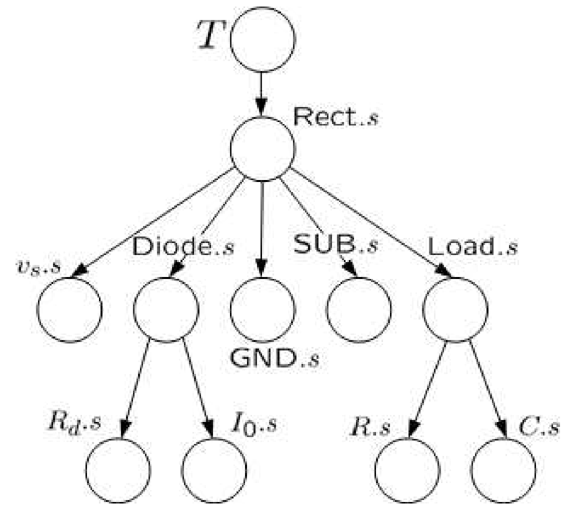
Representing Hierarchy



(c) $Rect = v_s \,||\, (R_d \,||\, I_0) \,||\, \mathrm{GND} \,||\, \mathrm{SUB} \,||\, (R \,||\, C)$



(d) Component Tree



(e) Scheduler Tree

Let $\mathcal{G} : S_N \longrightarrow 2^{S_N}$ be a function that associates to each scheduler the set of its children, and let $\Pi : S_N \longrightarrow \{1, \ldots, |S_N|\}$ be a global ordering of the nodes. Let $\mathcal{I} : \mathcal{C} \longrightarrow \mathcal{S}$ be a function that associates to each component it's scheduler.

Scheduler's `resolve` Algorithm

resolve(t)
$children \Leftarrow \mathcal{G}(s)$
if $children = \emptyset$ **then**
  // s is a leaf, proceed to solve the equations and end recursion
  $\mathcal{D}' \Leftarrow \{D \in \mathcal{I}^{-1}(s).\mathcal{D} \mid val(\mathcal{I}^{-1}(s).V_t) \in D\}$
  $J \Leftarrow \cup_{D \in \mathcal{D}'} s.\sigma(D)$
  $E_t \Leftarrow \cup_{i \in J} s.\omega(i)$
  $E_t \Leftarrow \texttt{sort}(E_t, s.\pi)$
  for all $e_i \in E_t$ **do**
    $\texttt{solve}(e_i, t)$
  end for
  `markchange` ($\mathcal{D}', val(\mathcal{I}^{-1}(s).V_t)$)
else
  // s is not a leaf, continue the recursion
  $children \Leftarrow \texttt{sort}(children, \Pi)$
  for all $s_i \in children$ **do**
    $s_i.\texttt{resolve}(t)$
  end for
end if

Scheduler's `resolve` Algorithm

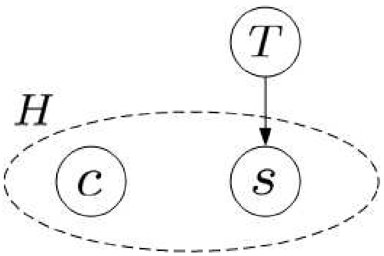
resolve(t)
$children \Leftarrow \mathcal{G}(s)$
if $children = \emptyset$ **then**
  // s is a leaf, proceed to solve the equations and end recursion
  $\mathcal{D}' \Leftarrow \{D \in \mathcal{I}^{-1}(s).\mathcal{D} \mid val(\mathcal{I}^{-1}(s).V_t) \in D\}$
  $J \Leftarrow \cup_{D \in \mathcal{D}'} s.\sigma(D)$
  $E_t \Leftarrow \cup_{i \in J} s.\omega(i)$
  $E_t \Leftarrow \mathtt{sort}(E_t, s.\pi)$
  for all $e_i \in E_t$ **do**
    $\mathtt{solve}(e_i, t)$
  end for
  `markchange` ($\mathcal{D}', val(\mathcal{I}^{-1}(s).V_t)$)
else
  // s is not a leaf, continue the recursion
  $children \Leftarrow \mathtt{sort}(children, \Pi)$
  for all $s_i \in children$ **do**
    $s_i.\mathtt{resolve}(t)$
  end for
end if

Scheduler's `resolve` Algorithm

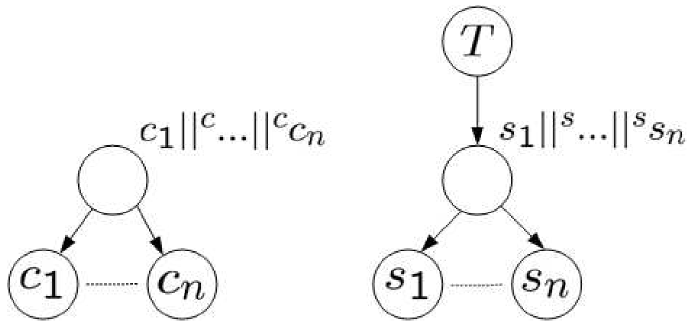
resolve(t)
$children \Leftarrow \mathcal{G}(s)$
if $children = \emptyset$ **then**
  // s is a leaf, proceed to solve the equations and end recursion
  $\mathcal{D}' \Leftarrow \{D \in \mathcal{I}^{-1}(s).\mathcal{D} \mid val(\mathcal{I}^{-1}(s).V_t) \in D\}$
  $J \Leftarrow \cup_{D \in \mathcal{D}'} s.\sigma(D)$
  $E_t \Leftarrow \cup_{i \in J} s.\omega(i)$
  $E_t \Leftarrow \texttt{sort}(E_t, s.\pi)$
  for all $e_i \in E_t$ **do**
    $\texttt{solve}(e_i, t)$
  end for
  $\texttt{markchange}$ ($\mathcal{D}', val(\mathcal{I}^{-1}(s).V_t)$)
else
  // s is not a leaf, continue the recursion
  $children \Leftarrow \texttt{sort}(children, \Pi)$
  for all $s_i \in children$ **do**
    $s_i.\texttt{resolve}(t)$
  end for
end if

Examples and Conclusions

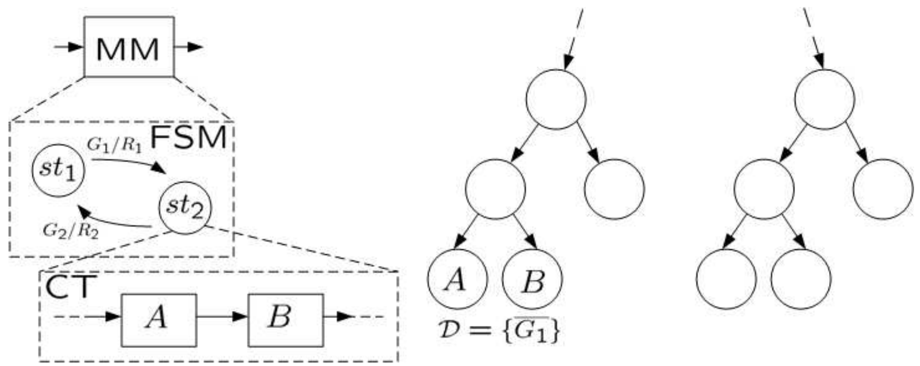
Examples



Structure of CHECKMATE programs



Structure of HYTECH programs



Structure of HYVISUAL programs

Conclusions and Future Work

- We have presented an abstract semantics for hybrid systems. It can be refined by specifying:
  - the time stamper automaton
  - the functions `domainchange`, `solve`, `next`
- We have shown how the structure of hybrid systems can be captured in the interchange semantics.
- We have implemented a prototype of the half-wave rectifier in METROPOLIS and can be downloaded at `http://embedded.eecs.berkeley.edu/hyinfo`.

- Future work:
  - Implementation of a METROPOLIS library for the interchange format;
  - Integration of a DAE solver (in collaboration with Jaijeet Roychowdhury, University of Minnesota)
  - Implementation of translators to/from known tools