Divide and Recycle: Types and Compilation for a Hybrid Synchronous Language

Albert Benveniste[1] Benoît Caillaud[1]
Timothy Bourke[1] Marc Pouzet[1,2,3]

1. INRIA

2. Institut Universitaire de France

3. École normale supérieure (LIENS)

LCTES 2011, CPS Week, April 11–14, Chicago, IL, USA

Motivation

Hybrid Systems Modelers

- Simulink
- Ptolemy
- ...

- Platforms for simulation and **development**
- More and more important
  - Semantics
  - Efficiency and predictability
  - Fidelity / Consistency

Motivation

Hybrid Systems Modelers
- Simulink
- Ptolemy
- ...

- Platforms for simulation and **development**
- More and more important
  - Semantics
  - Efficiency and predictability
  - Fidelity / Consistency

Conservative extension of a synchronous data-flow language

Motivation

Hybrid Systems Modelers — Simulink, Ptolemy, ...

- Platforms for simulation and **development**
- More and more important
  - Semantics
  - Efficiency and predictability
  - Fidelity / Consistency

Conservative extension of a synchronous data-flow language

What distinguishes our approach?

- Compilation with existing tools
  (after source-to-source transformation)
- Static typing
- Semantics based on non-standard analysis

Outline

Modeling

Model discrete systems with data-flow equations

Model physical systems with Ordinary Differential Equations (ODEs)

Modeling

Model discrete systems with data-flow equations

Model physical systems with Ordinary Differential Equations (ODEs)

$$\dot{\mathbf{y}}(t) = f(t, \mathbf{y})$$

instantaneous derivatives

variables

$$\mathbf{y}(0) = \mathbf{y}_i$$

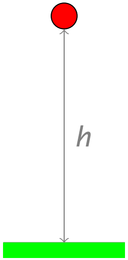
initial values

(Causal) First-order ODEs

- Causal: inputs on right,
  outputs on left
- First-order:
  one equation = one variable

Bouncing ball

model



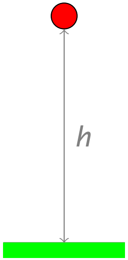
$$F = m \cdot a$$

$$-g = m \cdot \frac{d^2 h(t)}{dt^2}$$

$$\frac{d^2 h(t)}{dt^2} = -g/m$$

Bouncing ball

model



$$F = m \cdot a$$

$$-g = m \cdot \frac{d^2 h(t)}{dt^2}$$

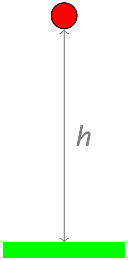
$$\frac{d^2 h(t)}{dt^2} = -g/m$$

$$\dot{v} = -g/m \qquad v(0) = v_0$$

$$\dot{h} = v \qquad h(0) = h_0$$

Bouncing ball

model



$$F = m \cdot a$$

$$-g = m \cdot \frac{d^2 h(t)}{dt^2}$$

$$\frac{d^2 h(t)}{dt^2} = -g/m$$

$$\dot{v} = -g/m \qquad v(0) = v_0$$

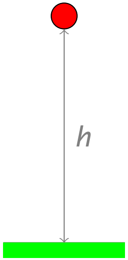
$$\dot{h} = v \qquad h(0) = h_0$$

$$v(t) = v_0 + \int_0^t (-g/m) \, .d\tau$$

$$h(t) = h_0 + \int_0^t v(\tau) \, .d\tau$$

Bouncing ball

model



$$F = m \cdot a$$

$$-g = m \cdot \frac{d^2 h(t)}{dt^2}$$

$$\frac{d^2 h(t)}{dt^2} = -g/m$$

Solver

f

$$\dot{v} = -g/m$$
$$\dot{h} = v$$

y_i

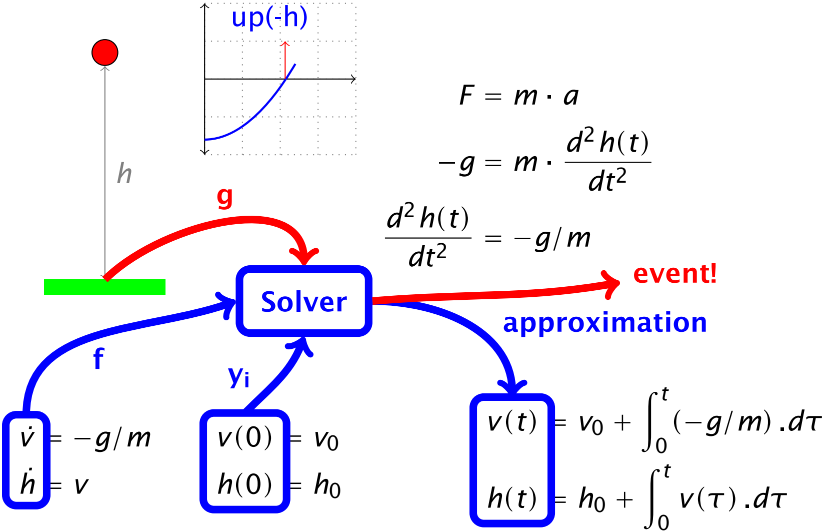
$$v(0) = v_0$$
$$h(0) = h_0$$

approximation

$$v(t) = v_0 + \int_0^t (-g/m)\,.d\tau$$
$$h(t) = h_0 + \int_0^t v(\tau)\,.d\tau$$

Bouncing ball

model



$$F = m \cdot a$$

$$-g = m \cdot \frac{d^2 h(t)}{dt^2}$$

$$\frac{d^2 h(t)}{dt^2} = -g/m$$

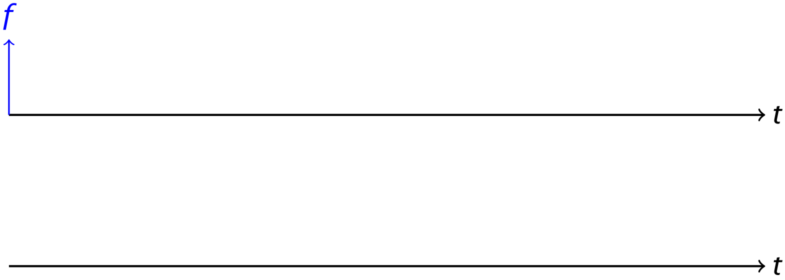
$$\boxed{\begin{matrix}\dot{v}\\ \dot{h}\end{matrix}}\begin{matrix}= -g/m\\ = v\end{matrix}$$

$$\boxed{\begin{matrix}v(0)\\ h(0)\end{matrix}}\begin{matrix}= v_0\\ = h_0\end{matrix}$$

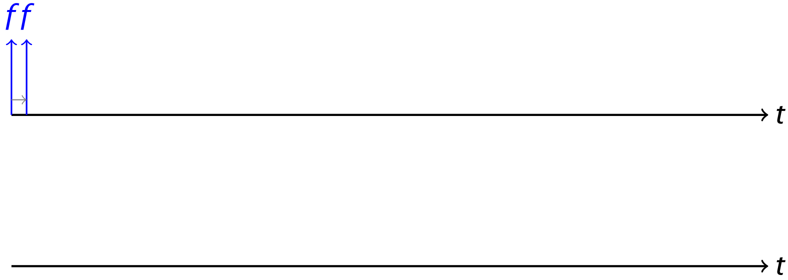
$$\boxed{\begin{matrix}v(t)\\ h(t)\end{matrix}}\begin{matrix}= v_0 + \int_0^t (-g/m)\,.d\tau\\ = h_0 + \int_0^t v(\tau)\,.d\tau\end{matrix}$$

Solver execution

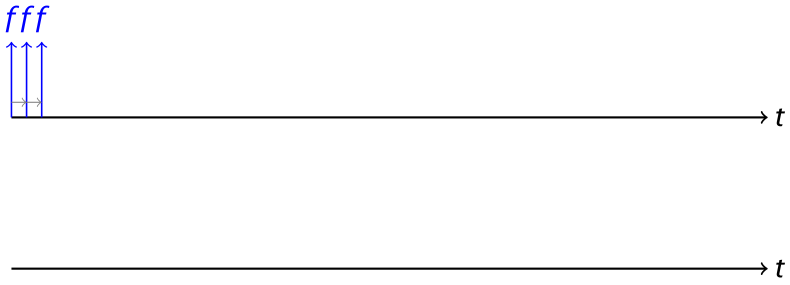
Solver execution



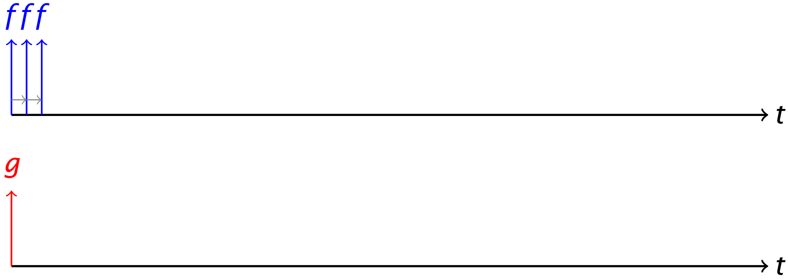
Solver execution



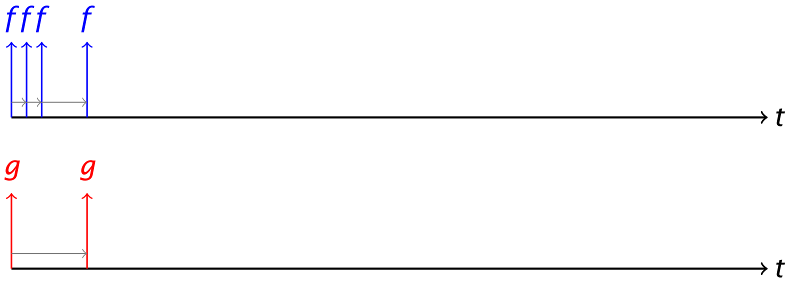
Solver execution



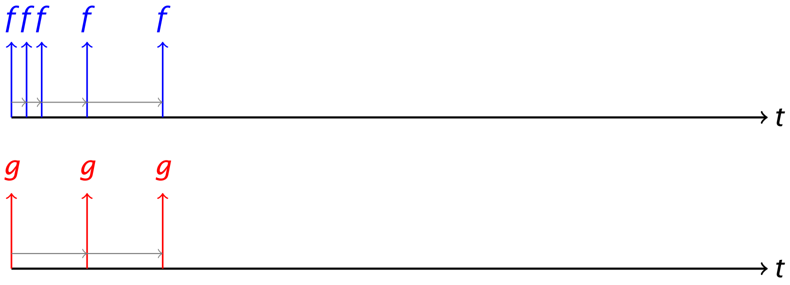
Solver execution



Solver execution



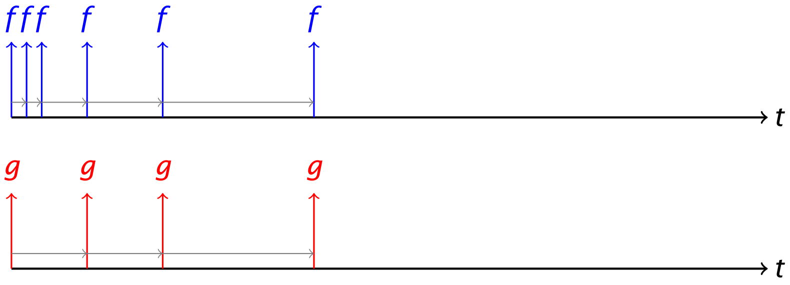
Solver execution



- Bigger and bigger steps (bound by h_{min} and h_{max})

Solver execution

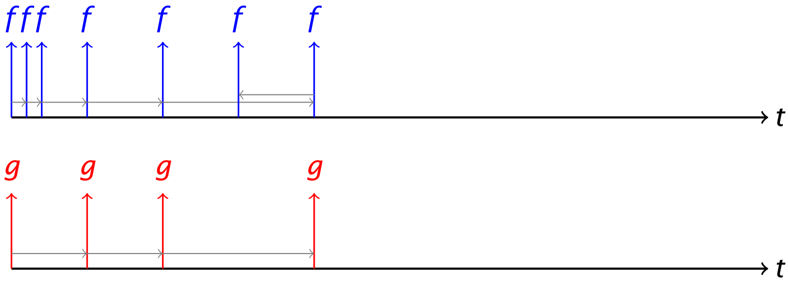
1. approximation error too large



- Bigger and bigger steps (bound by h_{min} and h_{max})

Solver execution

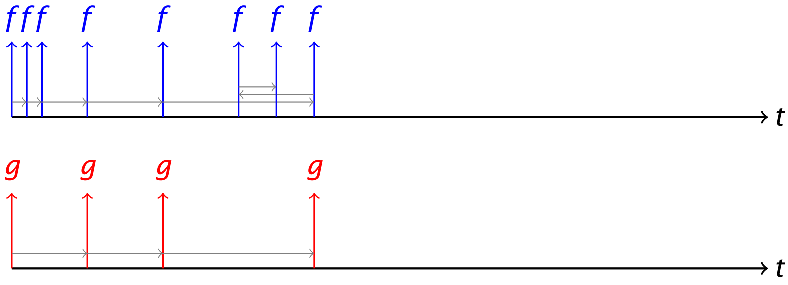
1. approximation error too large



- Bigger and bigger steps (bound by h_{min} and h_{max})

Solver execution

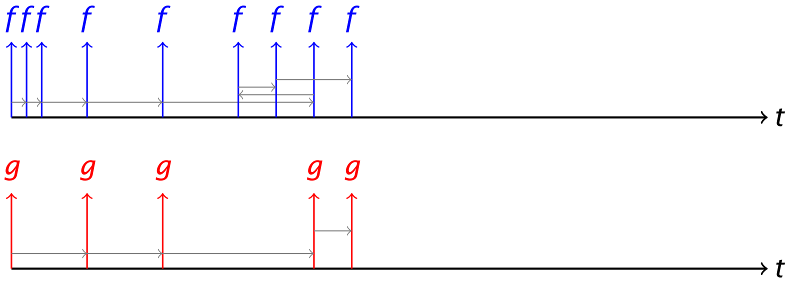
1. approximation error too large



- Bigger and bigger steps (bound by h_{min} and h_{max})

Solver execution

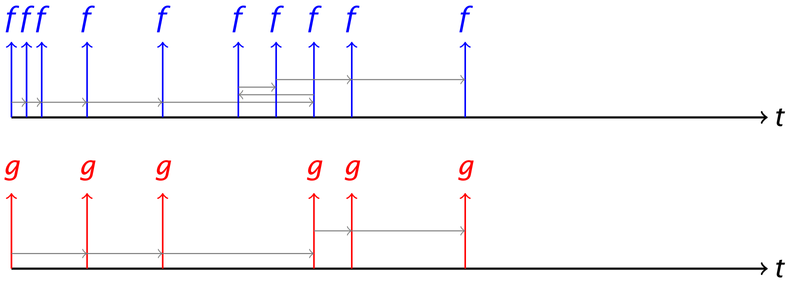
1. approximation error too large



- Bigger and bigger steps (bound by h_{min} and h_{max})

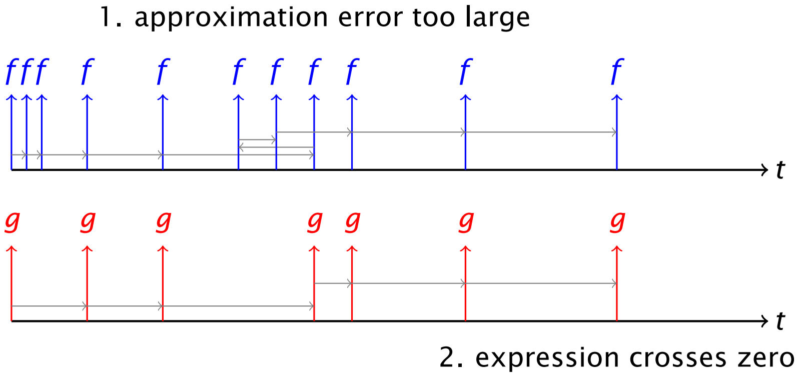
Solver execution

1. approximation error too large



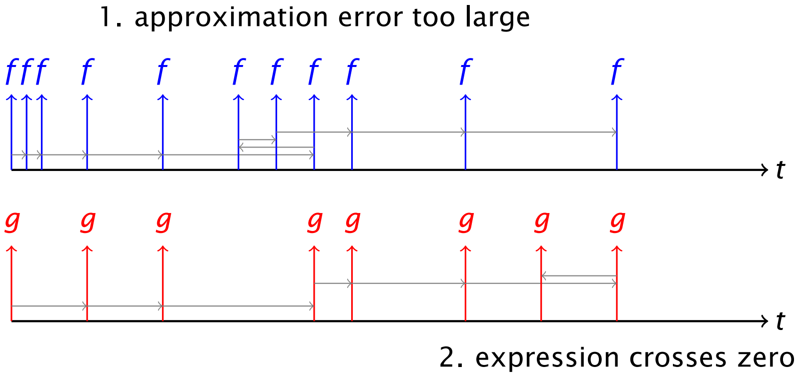
- Bigger and bigger steps (bound by h_{min} and h_{max})

Solver execution



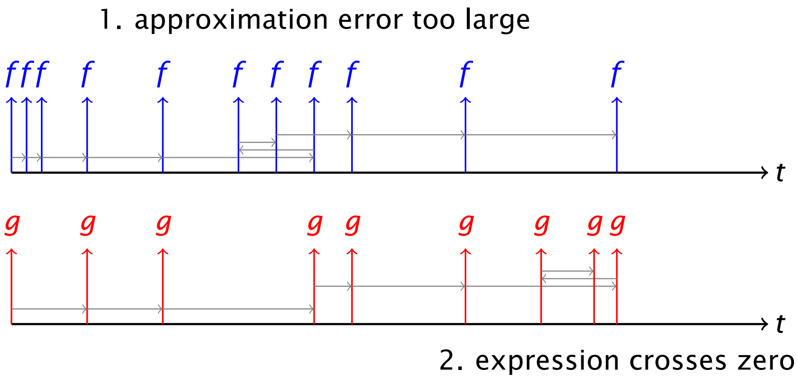
- Bigger and bigger steps (bound by h_{min} and h_{max})

Solver execution



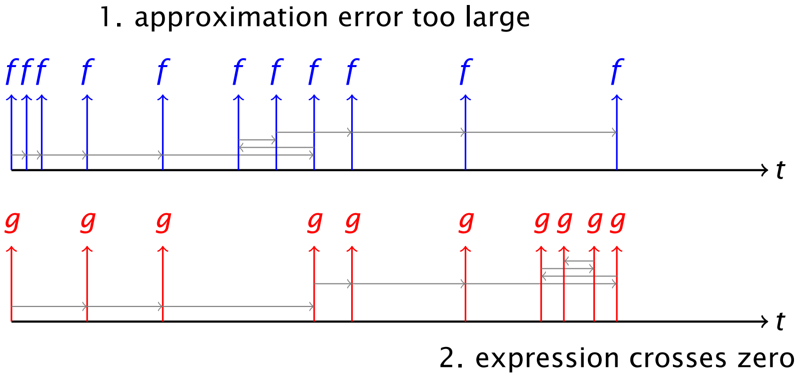
- Bigger and bigger steps (bound by h_{min} and h_{max})

Solver execution



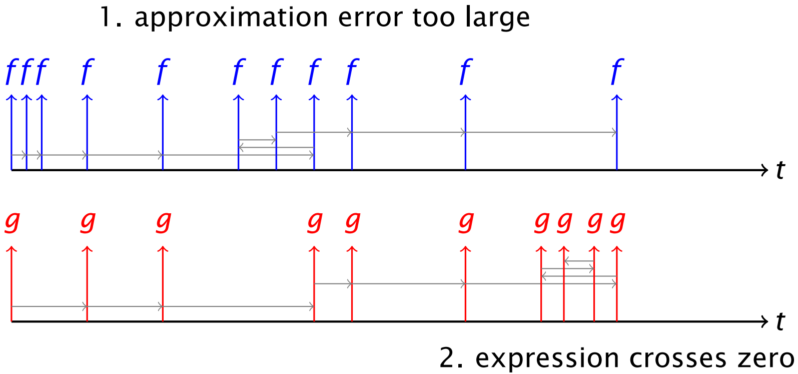
- Bigger and bigger steps (bound by h_{min} and h_{max})

Solver execution



- Bigger and bigger steps (bound by h_{min} and h_{max})

Solver execution



- Bigger and bigger steps (bound by h_{min} and h_{max})
- t does not necessarily advance monotonically
  - Ok for continuous states (managed by solver)
  - Cannot change state within f or g

Basic Hybrid Language

Start with Lucid Synchrone (subset); add first-order ODEs with reset.

$$\underbrace{\dot{\mathbf{x}}}_{\text{instantaneous derivatives}}(t) = f(t, \underbrace{\mathbf{x}}_{\text{variables}}) \qquad \underbrace{\mathbf{x}}_{\text{initial values}}(0) = \mathbf{x}_i$$

Basic Hybrid Language

Start with Lucid Synchrone (subset); add first-order ODEs with reset.

$$\dot{\mathbf{x}}(t) = f(t, \mathbf{x}) \qquad \mathbf{x}(0) = \mathbf{x}_i$$

instantaneous derivatives

variables

initial values

Rather than $\dot{x} = e_d$ and $x(0) = x_i$, write

der x = e_d **init** x_i

Basic Hybrid Language

Start with Lucid Synchrone (subset); add first-order ODEs with reset.

$$\dot{\mathbf{x}}(t) = f(t, \mathbf{x}) \qquad \mathbf{x}(0) = \mathbf{x}_i$$

instantaneous derivatives

variables

initial values

Rather than $\dot{x} = e_d$ and $x(0) = x_i$, write

der x = e_d **init** x_i **reset** e_1 **every** up(e_{z_1})

Basic Hybrid Language

Start with Lucid Synchrone (subset); add first-order ODEs with reset.

$$\dot{\mathbf{x}}(t) = f(t, \mathbf{x}) \qquad \mathbf{x}(0) = \mathbf{x}_i$$

instantaneous derivatives

variables

initial values

Rather than $\dot{x} = e_d$ and $x(0) = x_i$, write

$$\begin{array}{llllll} \textbf{der}\ x = e_d\ \textbf{init} & x_i & \textbf{reset} & e_1 & \textbf{every} & \mathrm{up}(e_{z_1}) \\ & \cdots & \mid & e_n & \textbf{every} & \mathrm{up}(e_{z_n}) \end{array}$$

Basic Hybrid Language

Start with Lucid Synchrone (subset); add first-order ODEs with reset.

$$\dot{\mathbf{x}}(t) = f(t, \mathbf{x}) \qquad \mathbf{x}(0) = \mathbf{x}_i$$

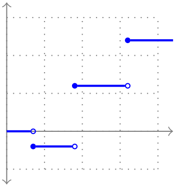
instantaneous derivatives

variables

initial values

Rather than $\dot{x} = e_d$ and $x(0) = x_i$, write

$$\begin{array}{llllllll} \textbf{der} & x = e_d & \textbf{init} & x_i & \textbf{reset} & e_1 & \textbf{every} & \mathrm{up}(e_{z_1}) \\ & & & \cdots & \mid & e_n & \textbf{every} & \mathrm{up}(e_{z_n}) \end{array}$$



$$x = (\text{pre h} + 1) \;\; \textbf{every} \;\; \mathrm{up}(e) \qquad\qquad \textbf{init} \;\; e_i$$

Basic Hybrid Language

Start with Lucid Synchrone (subset); add first-order ODEs with reset.

$$\dot{\mathbf{x}}(t) = f(t, \mathbf{x}) \qquad \mathbf{x}(0) = \mathbf{x}_i$$

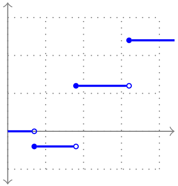
instantaneous derivatives

variables

initial values

Rather than $\dot{x} = e_d$ and $x(0) = x_i$, write

$$\begin{array}{llllll} \textbf{der}\ x = e_d\ \textbf{init}\ x_i & \textbf{reset} & e_1 & \textbf{every} & \text{up}(e_{z_1}) \\ \quad\cdots & \mid & e_n & \textbf{every} & \text{up}(e_{z_n}) \end{array}$$



$$\begin{array}{llll} x = & (\text{pre h} + 1) & \textbf{every} & \text{up}(e) \qquad \textbf{init}\ e_i \\ \quad\mid & \textit{purely sync} & \textbf{every} & \textit{event} \end{array}$$

Basic Hybrid Language

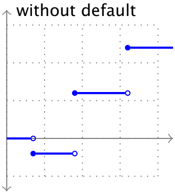
Start with Lucid Synchrone (subset); add first-order ODEs with reset.

$$\dot{\mathbf{x}}(t) = f(t, \mathbf{x}) \qquad \mathbf{x}(0) = \mathbf{x}_i$$

instantaneous derivatives

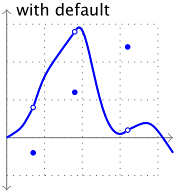
variables

initial values



Rather than $\dot{x} = e_d$ and $x(0) = x_i$, write

$$\textbf{der}\ x = e_d\ \textbf{init}\ x_i\ \textbf{reset}\ e_1\ \textbf{every}\ \text{up}(e_{z_1})\ \cdots\ |\ e_n\ \textbf{every}\ \text{up}(e_{z_n})$$



$$x = (\text{pre h} + 1)\ \textbf{every}\ \text{up}(e)\ \textbf{default}\ e_c\ \textbf{init}\ e_i$$

$$|\ \textit{purely sync}\ \textbf{every}\ \textit{event}$$

Basic Hybrid Language

Start with Lucid Synchrone (subset); add first-order ODEs with reset.

$$\dot{\mathbf{x}}(t) = f(t, \mathbf{x}) \qquad \mathbf{x}(0) = \mathbf{x}_i$$

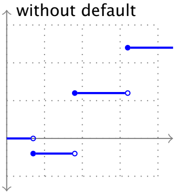
instantaneous derivatives

variables

initial values

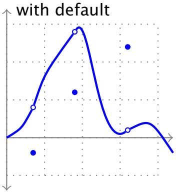
Rather than $\dot{x} = e_d$ and $x(0) = x_i$, write

$$\begin{array}{lllll} \textbf{der } x = e_d \textbf{ init} & x_i & \textbf{reset} & e_1 & \textbf{every} \ \text{up}(e_{z_1}) \\ & \cdots & | & e_n & \textbf{every} \ \text{up}(e_{z_n}) \end{array}$$



$$x = (\text{pre h} + 1) \ \textbf{every} \ \text{up}(e) \ \textbf{default} \ e_c \ \textbf{init} \ e_i$$

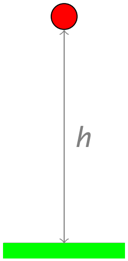
$$| \ \ \textit{purely sync} \ \textbf{every} \ \textit{event}$$



Very simple: no clocks, no automata, no higher-order

Bouncing ball

program



$$\dot{v} = -g/m \qquad v(0) = v_0$$

$$\dot{h} = v \qquad h(0) = h_0$$

reset v to $-0.8 \cdot v$ when h becomes 0

```
let hybrid ball () =
  let
  rec der v = (-. g / m) init v0
                 reset (-. 0.8 *. last v) every up(-. h)
  and der h =      v init h0
  in (v, h)
```

Semantics

reals		*non-standard reals*
$\mathbb{R}$	+ infinitesimals (∂)	${}^{\star}\mathbb{R}$

Semantics

reals $\mathbb{R}$ $\xrightarrow{\text{+ infinitesimals } (\partial)}$ *non-standard reals* ${}^{\star}\mathbb{R}$

$$\cdots < t - 3\partial < t - 2\partial < t - \partial < \mathrm{t} < t + \partial < t + 2\partial < t + 3\partial < \cdots$$

Semantics

reals $\mathbb{R}$ **+ infinitesimals (∂)** → *non-standard reals* ${}^{\star}\mathbb{R}$

$$\cdots < t-3\partial < t-2\partial < t-\partial < \mathrm{t} < t+\partial < t+2\partial < t+3\partial < \cdots$$

- dense and discrete
- base clock for both continuous and discrete behaviors
- $\forall t.\ {}^{\bullet}t$ is the previous instant, $t^{\bullet}$ is the next instant

$$
\begin{array}{lcll}
integr^{\#}(T)(s)(s_0)(hs)(t) & = & s'(t) & \text{where} \\
s'(t) & = & s_0(t) & \text{if } t = \min(T) \\
s'(t) & = & s'({}^{\bullet}t) + \partial s({}^{\bullet}t) & \text{if } handler^{\#}(T)(hs)(t) = NoEvent \\
s'(t) & = & v & \text{if } handler^{\#}(T)(hs)(t) = Xcrossing(v) \\
 & & & \\
up^{\#}(T)(s)(t) & = & \mathtt{false} & \text{if } t = \min(T) \\
up^{\#}(T)(s)(t^{\bullet}) & = & \mathtt{true} & \text{if } (s({}^{\bullet}t) \leq 0) \wedge (s(t) > 0) \text{ and } (t \in T) \\
up^{\#}(T)(s)(t^{\bullet}) & = & \mathtt{false} & \text{otherwise} \\
\ldots & & \ldots & \ldots
\end{array}
$$

Semantics

reals $\mathbb{R}$ + infinitesimals (∂) → *non-standard reals* ${}^\star\mathbb{R}$

$$\cdots < t - 3\partial < t - 2\partial < t - \partial < t < t + \partial < t + 2\partial < t + 3\partial < \cdots$$

- dense and discrete
- base clock for both continuous and discrete behaviors
- $\forall t.\ {}^\bullet t$ is the previous instant, $t^\bullet$ is the next instant

$$
\begin{array}{lcll}
\mathit{integr}^\#(T)(s)(s_0)(hs)(t) & = & s'(t) & \text{where} \\
s'(t) & = & s_0(t) & \text{if } t = \min(T) \\
s'(t) & = & s'({}^\bullet t) + \partial s({}^\bullet t) & \text{if } \mathit{handler}^\#(T)(hs)(t) = \mathit{NoEvent} \\
s'(t) & = & v & \text{if } \mathit{handler}^\#(T)(hs)(t) = \mathit{Xcrossing}(v) \\
 & & & \\
\mathit{up}^\#(T)(s)(t) & = & \mathtt{false} & \text{if } t = \min(T) \\
\mathit{up}^\#(T)(s)(t^\bullet) & = & \mathtt{true} & \text{if } (s({}^\bullet t) \leq 0) \wedge (s(t) > 0) \text{ and } (t \in T) \\
\mathit{up}^\#(T)(s)(t^\bullet) & = & \mathtt{false} & \text{otherwise} \\
\ldots & & \ldots & \ldots
\end{array}
$$

Semantics

reals $\quad\quad$ *non-standard reals*

$\mathbb{R}$ $\xrightarrow{+\text{ infinitesimals } (\partial)}$ $^\star\mathbb{R}$

$$\cdots < t - 3\partial < t - 2\partial < t - \partial < \mathrm{t} < t + \partial < t + 2\partial < t + 3\partial < \cdots$$

- dense and discrete
- base clock for both continuous and discrete behaviors
- $\forall t.\ ^\bullet t$ is the previous instant, $t^\bullet$ is the next instant

$$
\begin{array}{lcll}
integr^{\#}(T)(s)(s_0)(hs)(t) & = & s'(t) & \text{where} \\
s'(t) & = & s_0(t) & \text{if } t = \min(T) \\
s'(t) & = & s'(^\bullet t) + \partial s(^\bullet t) & \text{if } handler^{\#}(T)(hs)(t) = NoEvent \\
s'(t) & = & v & \text{if } handler^{\#}(T)(hs)(t) = Xcrossing(v) \\
 & & & \\
up^{\#}(T)(s)(t) & = & \mathtt{false} & \text{if } t = \min(T) \\
up^{\#}(T)(s)(t^\bullet) & = & \mathtt{true} & \text{if } (s(^\bullet t) \leq 0) \wedge (s(t) > 0) \text{ and } (t \in T) \\
up^{\#}(T)(s)(t^\bullet) & = & \mathtt{false} & \text{otherwise} \\
 & & & \\
\ldots & & \ldots & \ldots
\end{array}
$$

Semantics

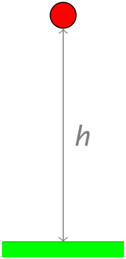
reals $\quad \mathbb{R} \xrightarrow{+\text{ infinitesimals } (\partial)} {}^{\star}\mathbb{R} \quad$ *non-standard reals*

$$\cdots < t-3\partial < t-2\partial < t-\partial < t < t+\partial < t+2\partial < t+3\partial < \cdots$$

- dense and discrete
- base clock for both continuous and discrete behaviors
- $\forall t.\ {}^{\bullet}t$ is the previous instant, $t^{\bullet}$ is the next instant

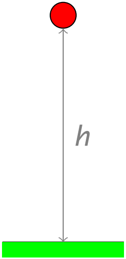
$$
\begin{array}{lcll}
\mathit{integr}^{\#}(T)(s)(s_0)(hs)(t) & = & s'(t) & \text{where} \\
s'(t) & = & s_0(t) & \text{if } t = \min(T) \\
s'(t) & = & s'({}^{\bullet}t) + \partial s({}^{\bullet}t) & \text{if } \mathit{handler}^{\#}(T)(hs)(t) = \mathit{NoEvent} \\
s'(t) & = & v & \text{if } \mathit{handler}^{\#}(T)(hs)(t) = \mathit{Xcrossing}(v) \\
 & & & \\
\mathit{up}^{\#}(T)(s)(t) & = & \mathtt{false} & \text{if } t = \min(T) \\
\mathit{up}^{\#}(T)(s)(t^{\bullet}) & = & \mathtt{true} & \text{if } (s({}^{\bullet}t) \leq 0) \wedge (s(t) > 0) \text{ and } (t \in T) \\
\mathit{up}^{\#}(T)(s)(t^{\bullet}) & = & \mathtt{false} & \text{otherwise} \\
\ldots & & \ldots & \ldots
\end{array}
$$

Compilation



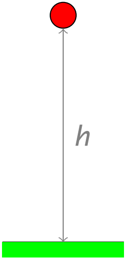

```
let hybrid ball () =
  let
  rec der v = (−. g / m) init v0
                  reset (−. 0.8 *. last v) every up(-. h)
  and der h =     v init h0
  in (v, h)
```

Compilation




```
let hybrid ball () =
  let
  rec der v = (-. g / m) init v0
              reset (-. 0.8 *. last v) every up(-. h)
  and der h =     v init h0
  in (v, h)

let node ball (z1, (lh, lv), ()) =
  let rec i = true fby false

  and dv = (-. g / m)
  and v = if i then v0
          else if z1 then -. 0.8 *. lv
          else lv

  and dh = v
  and h  = if i then h0 else lh

  and upz1 = -. h

  in ((v, h), upz1, (h, v), (dh, dv))
```

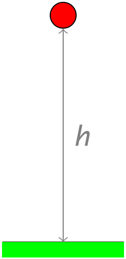
Compilation




```
let hybrid ball () =
  let
  rec der v = (-. g / m) init v0
                 reset (-. 0.8 *. last v) every up(-. h)
  and der h =     v init h0
  in (v, h)

let node ball (z1, (lh, lv), ()) =
  let rec i = true fby false

  and dv = (-. g / m)
  and v = if i then v0
          else if z1 then -. 0.8 *. lv
          else lv

  and dh = v
  and h  = if i then h0 else lh

  and upz1 = -. h

  in ((v, h), upz1, (h, v), (dh, dv))
```

transform into discrete subset

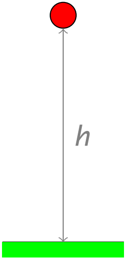
Compilation




```
let hybrid ball () =
  let
  rec der v = (-. g / m) init v0
                reset (-. 0.8 *. last v) every up(-. h)
  and der h =      v init h0
  in (v, h)

let node ball (z1, (lh, lv), ()) =
  let rec i = true fby false

  and dv = (-. g / m)
  and v = if i then v0
          else if z1 then -. 0.8 *. lv
          else lv

  and dh = v
  and h  = if i then h0 else lh

  and upz1 = -. h

  in ((v, h), upz1, (h, v), (dh, dv))
```

transform continuous variables

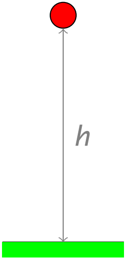
Compilation




```
let hybrid ball () =
  let
  rec der v = (-. g / m) init v0
                 reset (-. 0.8 *. last v) every up(-. h)
  and der h =      v init h0
  in (v, h)

let node ball (z1, (lh, lv), ()) =
  let rec i = true fby false

  and dv = (-. g / m)
  and v = if i then v0
          else if z1 then -. 0.8 *. lv
          else lv

  and dh = v
  and h  = if i then h0 else lh

  and upz1 = -. h

  in ((v, h), upz1, (h, v), (dh, dv))
```

transform zero-crossings

Compilation



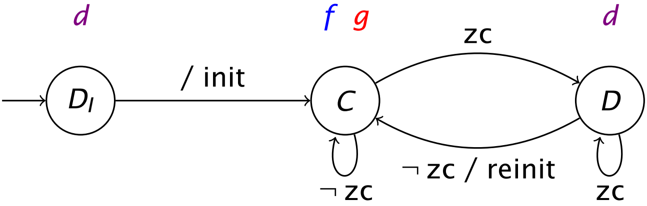

```
let hybrid ball () =
  let
  rec der v = (-. g / m) init v0
                  reset (-. 0.8 *. last v) every up(-. h)
  and der h =     v init h0
  in (v, h)

let node ball (z1, (lh, lv), ()) =
  let rec i = true fby false

  and dv = (-. g / m)
  and v = if i then v0
          else if z1 then -. 0.8 *. lv
          else lv

  and dh
  and h

  and up

  in ((v, h), upz1, (h, v), (dh, dv))
```

All continuous parts execute in 1st instant

- type system prevents C inside D
- no branching or activations

Execution (Simulation)

$$
\begin{array}{lcll}
(upz, y, dy) & = & main_\sigma(z, y) & \\
{\color{blue}f}\,(t, y) & = & \text{let } (\_, \_, dy) = main_\sigma(false, y) & \text{in } dy \\
{\color{red}g}\,(t, y) & = & \text{let } (upz, \_, \_) = main_\sigma(false, y) & \text{in } upz \\
{\color{purple}d}\,(z, y) & = & \text{let } (upz, y, \_) = main_\sigma(z, y) & \text{in } (upz, y)
\end{array}
$$



- Only d may have side effects
- Neither f, nor g may change the (internal) discrete state

Typing

Motivation

This compilation/execution scheme only works for some programs!

Typing
Motivation

This compilation/execution scheme only works for some programs!

We need a type system to:

- Reject programs that do not respect the invariant:
  - discrete computations in D only
  - continuous evolutions in C only

Typing
Motivation

This compilation/execution scheme only works for some programs!

We need a type system to:

- Reject programs that do not respect the invariant:
  - discrete computations in D only
  - continuous evolutions in C only
- Reject unreasonable programs
  - where behavior depends ‘too much’ on simulation parameters (like the step size, or number of iterations)

Typing

Unreasonable programs

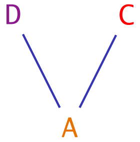
```
der y = 1.0 init 0.0        and    x = (0.0 → pre x) + y

x = 0.0 → (pre x +. 1.0)    and    der y = x init 0.0
```

- y is a variable that changes *continuously*
- x is *discrete* register
- The relationship between the two is ill-defined

Typing

The type language

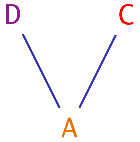
$$
\begin{array}{lcl}
bt & ::= & \texttt{float} \mid \texttt{int} \mid \texttt{bool} \mid \texttt{zero} \\
t & ::= & bt \mid t \times t \mid \beta \\
\sigma & ::= & \forall \beta_1, \ldots, \beta_n . t \xrightarrow{k} t \\
k & ::= & \texttt{D} \mid \texttt{C} \mid \texttt{A}
\end{array}
$$



Typing

The type language

$$
\begin{array}{lcl}
bt & ::= & \mathtt{float} \mid \mathtt{int} \mid \mathtt{bool} \mid \mathtt{zero} \\
t & ::= & bt \mid t \times t \mid \beta \\
\sigma & ::= & \forall \beta_1, \ldots, \beta_n . t \xrightarrow{k} t \\
k & ::= & \mathtt{D} \mid \mathtt{C} \mid \mathtt{A}
\end{array}
$$



Initial conditions

$$
\begin{array}{lcl}
(+) & : & \mathtt{int} \times \mathtt{int} \xrightarrow{\mathtt{A}} \mathtt{int} \\
(=) & : & \forall \beta . \beta \times \beta \xrightarrow{\mathtt{A}} \mathtt{bool} \\
\mathtt{if} & : & \forall \beta . \mathtt{bool} \times \beta \times \beta \xrightarrow{\mathtt{A}} \beta \\
\mathtt{pre}(.) & : & \forall \beta . \beta \xrightarrow{\mathtt{D}} \beta \\
\mathtt{.\,fby\,.} & : & \forall \beta . \beta \times \beta \xrightarrow{\mathtt{D}} \beta \\
\mathtt{up}(.) & : & \mathtt{float} \xrightarrow{\mathtt{C}} \mathtt{zero}
\end{array}
$$

Typing

$G, H \vdash_{\mathtt{C}}$ **der** y = 1.0 **init** 0.0

$G, H \vdash_{\mathtt{D}}$ x = (0.0 **fby** (x + 1))

Typing

$G, H \vdash_{\mathtt{C}}$ **der** y = 1.0 **init** 0.0 $\qquad$ $G, H \vdash_{\mathtt{D}}$ x = (0.0 **fby** (x + 1))

$G, H \vdash_{?}$ **der** y $= \cdots$ **and** x $= \cdots$

Typing

$G, H \vdash_{\mathtt{C}}$ **der** y = 1.0 **init** 0.0 $\qquad$ $G, H \vdash_{\mathtt{D}}$ x = (0.0 **fby** (x + 1))

$G, H \vdash_{?}$ **der** y $= \cdots$ **and** $x = \cdots$ ✗

Typing

$G, H \vdash_{\mathtt{C}}$ **der** y = 1.0 **init** 0.0

$G, H \vdash_{\mathtt{D}}$ x = (0.0 **fby** (x + 1))

$G, H \vdash_{\mathtt{C}}$ x' = (0.0 **fby** (x + 1)) **every** up(ez) **init** 0.0

$G, H \vdash_{?}$ **der** y $= \cdots$ **and** x $= \cdots$ ✗

Typing

$G, H \vdash_{\mathtt{C}}$ **der** y $=$ 1.0 **init** 0.0

$G, H \vdash_{\mathtt{D}}$ x $=$ (0.0 **fby** (x + 1))

$G, H \vdash_{\mathtt{C}}$ x' $=$ (0.0 **fby** (x + 1)) **every** up(ez) **init** 0.0

$G, H \vdash_{?}$ **der** y $= \cdots$ **and** $x = \cdots$ ✗

$G, H \vdash_{?}$ **der** y$= \cdots$ **and** $x' = \cdots$

Typing

$G, H \vdash_{\mathtt{C}}$ **der** y $=$ 1.0 **init** 0.0

$G, H \vdash_{\mathtt{D}} x \;=\;$ (0.0 **fby** (x + 1))

$G, H \vdash_{\mathtt{C}} x' \;=\;$ (0.0 **fby** (x + 1))
every up(ez) **init** 0.0

$G, H \vdash_{?}$ **der** y $= \cdots$ **and** $x = \cdots$ ✗

$G, H \vdash_{\mathtt{C}}$ **der** y$= \cdots$ **and** $x' = \cdots$ ✔

Typing

$G, H \vdash_{\mathtt{C}}$ **der** y $=$ 1.0 **init** 0.0

$G, H \vdash_{\mathtt{D}}$ x $=$ (0.0 **fby** (x + 1))

$G, H \vdash_{\mathtt{C}}$ x' $=$ (0.0 **fby** (x + 1)) **every** up(ez) **init** 0.0

$G, H \vdash_{?}$ **der** y $= \cdots$ **and** $x = \cdots$ ✗

$G, H \vdash_{\mathtt{C}}$ **der** y$= \cdots$ **and** $x' = \cdots$ ✔

$G, H \vdash_{?}$ $x = \cdots$ **and** $x = \cdots$

Typing

$G, H \vdash_{\mathtt{C}}$ **der** y $=$ 1.0 **init** 0.0

$G, H \vdash_{\mathtt{D}}$ x $=$ (0.0 **fby** (x + 1))

$G, H \vdash_{\mathtt{C}}$ x' $=$ (0.0 **fby** (x + 1))

every up(ez) **init** 0.0

$G, H \vdash_{?}$ **der** y $= \cdots$ **and** x $= \cdots$ ✗

$G, H \vdash_{\mathtt{C}}$ **der** y$= \cdots$ **and** x' $= \cdots$ ✓

$G, H \vdash_{\mathtt{D}}$ $x = \cdots$ **and** $x = \cdots$ ✓

Typing

$G, H \vdash_{\mathsf{C}}$ **der** y $=$ 1.0 **init** 0.0

$G, H \vdash_{\mathsf{D}}\ x = $ (0.0 **fby** (x + 1))

$G, H \vdash_{\mathsf{C}}\ x' = $ (0.0 **fby** (x + 1))
every up(ez) **init** 0.0

$G, H \vdash_{?}$ **der** y $= \cdots$ **and** $x = \cdots$ ✗

$G, H \vdash_{\mathsf{C}}$ **der** y$= \cdots$ **and** $x' = \cdots$ ✓

$G, H \vdash_{\mathsf{D}}\ x = \cdots$ **and** $x = \cdots$ ✓

Typing of function body gives its kind $k \in \{\mathsf{C}, \mathsf{D}, \mathsf{A}\}$:

$$h : \texttt{float} \times \texttt{float} \xrightarrow{k} \texttt{float} \times \texttt{float}$$

Less expressive but simpler than ‘per-wire’ kinds, e.g. Simulink

$$j : (\texttt{float}_{\mathsf{D}}) \times (\texttt{float}_{\mathsf{C}}) \longrightarrow (\texttt{float}_{\mathsf{D}}) \times (\texttt{float}_{\mathsf{C}})$$

Conclusion

- Simple extension of a synchronous data-flow language
  - Add first-order ODEs
  - and zero-crossing events
- Non-standard semantics
  - Gives a ‘continuous base clock’
  - Simplifies definitions, clarifies certain features
- Static block-based typing system
  - Divide system into continuous and discrete parts
- Compilation
  - Source-to-source transformation
  - Recycle existing compilers
- Execution
  - Simulate using Sundials CVODE solver

Conclusion

- Simple extension of a synchronous data-flow language
  - Add first-order ODEs
  - and zero-crossing events
- Non-standard semantics
  - Gives a ‘continuous base clock’
  - Simplifies definitions, clarifies certain features
- Static block-based typing system
  - Divide system into continuous and discrete parts
- Compilation
  - Source-to-source transformation
  - Recycle existing compilers
- Execution
  - Simulate using Sundials CVODE solver

Conclusion

- Simple extension of a synchronous data-flow language
  - Add first-order ODEs
  - and zero-crossing events
- Non-standard semantics
  - Gives a 'continuous base clock'
  - Simplifies definitions, clarifies certain features
- Static block-based typing system
  - Divide system into continuous and discrete parts
- Compilation
  - Source-to-source transformation
  - Recycle existing compilers
- Execution
  - Simulate using Sundials CVODE solver

Ocaml Sundials CVODE interface and compiler available