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

Albert Benveniste\textsuperscript{1} Benoît Caillaud\textsuperscript{1}
Timothy Bourke\textsuperscript{1} Marc Pouzet\textsuperscript{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

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

Simulink
Ptolemy
...
Motivation

Hybrid Systems Modelers

- 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

- 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

Background

Hybrid Synchronous Language
  Semantics
  Compilation
  Execution
  Typing

Conclusion
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)

\[ \dot{y}(t) = f(t, y) \]

**instantaneous derivatives**

\[ y(0) = y_i \]

**initial values**

**(Causal) First-order ODEs**

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

\[ F = m \cdot a \]

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

\[ \frac{d^2 h(t)}{dt^2} = -\frac{g}{m} \]
Bouncing ball model

\[
F = m \cdot a
\]

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

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

\[
\dot{v} = -\frac{g}{m} \quad v(0) = v_0
\]

\[
\dot{h} = v \quad h(0) = h_0
\]
Bouncing ball model

\[ F = m \cdot a \]

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

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

\[ \dot{v} = -\frac{g}{m} \quad v(0) = v_0 \]

\[ \dot{h} = v \quad h(0) = h_0 \]

\[ v(t) = v_0 + \int_0^t (-\frac{g}{m}) \cdot d\tau \]

\[ h(t) = h_0 + \int_0^t v(\tau) \cdot d\tau \]
Bouncing ball
model

\[ F = m \cdot a \]

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

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

\[ \dot{v} = -\frac{g}{m} \]

\[ h = v \]

\[ v(0) = v_0 \]

\[ h(0) = h_0 \]

\[ v(t) = v_0 + \int_0^t (-\frac{g}{m}) \cdot d\tau \]

\[ h(t) = h_0 + \int_0^t v(\tau) \cdot d\tau \]
Bouncing ball model

\[ F = m \cdot a \]

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

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

\[ \dot{v} = -\frac{g}{m} \]

\[ \dot{h} = v \]

\[ v(0) = v_0 \]

\[ h(0) = h_0 \]

\[ v(t) = v_0 + \int_0^t (-\frac{g}{m}) \cdot d\tau \]

\[ h(t) = h_0 + \int_0^t v(\tau) \cdot d\tau \]
Solver execution

1. Approximation error too large

2. Expression crosses zero

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

\( t \) does not necessarily advance monotonically

Ok for continuous states (managed by solver)

Cannot change state within \( f \) or \( g \)
Solver execution

1. Approximation error too large

2. Expression crosses zero

▶ Bigger and bigger steps (bound by $h_{\text{min}}$ and $h_{\text{max}}$)

▶ $t$ does not necessarily advance monotonically

▶ Ok for continuous states (managed by solver)

▶ Cannot change state within $f$ or $g$
Solver execution

1. Approximation error too large
2. Expression crosses zero

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

1. Approximation error too large

2. Expression crosses zero

▶ 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 $g$
Solver execution

1. approximation error too large

2. expression crosses zero

- 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 $g$
1. Approximation error too large

2. Expression crosses zero

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

1. approximation error too large

- f f f f f
- g g g g g

- Bigger and bigger steps (bound by $h_{min}$ and $h_{max}$)
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 $h_{\text{min}}$ and $h_{\text{max}}$)
Solver execution

1. approximation error too large

2. expression crosses zero

- Bigger and bigger steps (bound by $h_{\text{min}}$ and $h_{\text{max}}$)
Solver execution

1. approximation error too large

\[ f f f \quad f \quad f \quad f \quad f \quad f \quad f \quad f \quad f \quad f \]

2. expression crosses zero

\[ g \quad g \quad g \quad g \quad g \quad g \quad g \quad g \quad g \quad g \]

- Bigger and bigger steps (bound by \( h_{min} \) and \( h_{max} \))
1. approximation error too large

2. expression crosses zero

- Bigger and bigger steps (bound by $h_{min}$ and $h_{max}$)
1. approximation error too large

2. expression crosses zero

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

1. approximation error too large

2. expression crosses zero

▷ Bigger and bigger steps (bound by $h_{\text{min}}$ and $h_{\text{max}}$)

▷ $t$ 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.

\[ \dot{x}(t) = f(t, x) \]

- instantaneous derivatives
- variables
- initial values

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

\[ \text{der}_x = e_{\text{init}}x_i \]

reset every \( e_z_1 \)...

...\( \text{default}_{\text{init}}e_i \)...

purely sync every event

Very simple: no clocks, no automata, no higher-order without default

with default
Basic Hybrid Language
Start with Lucid Synchrone (subset); add first-order ODEs with reset.

\[ \dot{x}(t) = f(t, x) \]
\[ x(0) = x_i \]

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

\[ \text{der } x = e_d \text{ init } x_i \]
Basic Hybrid Language

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

\[
\dot{x}(t) = f(t, x) \quad x(0) = x_i
\]

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

\[
\text{der } x = e_d \text{ init } x_i \text{ reset } e_1 \text{ every } \text{up}(e_{z_1})
\]
Basic Hybrid Language
Start with Lucid Synchrone (subset); add first-order ODEs with reset.

\[ \dot{x}(t) = f(t, x) \]

\[ x(0) = x_i \]

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

\[ \text{der } x = e_d \text{ init } x_i \text{ reset } e_1 \text{ every } \text{up}(e_{z_1}) \]
\[ \ldots \text{ init } x_i \text{ reset } e_n \text{ every } \text{up}(e_{z_n}) \]
Basic Hybrid Language

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

\[ \dot{x}(t) = f(t, x) \]

\[ x(0) = x_i \]

instantaneous derivatives

variables

initial values

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

\[ \text{der } x = e_d \quad \text{init } x_i \quad \text{reset } e_1 \quad \text{every } \text{up}(e_{z_1}) \]

\[ \cdots \]

\[ \text{init } e_i \quad \text{every } \text{up}(e_{z_n}) \]

\[ x = (\text{pre } h + 1) \quad \text{every } \text{up}(e) \]
Basic Hybrid Language

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

\[
\frac{dx}{dt} = f(t, x) \\
x(0) = x_i
\]

Rather than \( \frac{dx}{dt} = e_d \) and \( x(0) = x_i \), write

\[
\text{der} \ x = e_d \quad \text{init} \ x_i \quad \text{reset} \ e_1 \quad \text{every} \ \text{up}(e_{z_1}) \\
\quad \cdots \\
\quad \text{init} \ e_i \quad \text{every} \ \text{up}(e_{z_n})
\]

\[
x = (\text{pre } h + 1) \quad \text{every} \ \text{up}(e) \\
| \quad \text{purely sync} \quad \text{every} \ \text{event}
\]
Basic Hybrid Language

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

\[
\dot{x}(t) = f(t, x) \\
x(0) = x_i
\]

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

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

\[
x = (\text{pre } h + 1) \quad \text{every } \text{up}(e) \quad \text{default } e_c \quad \text{init } e_i \\
| \quad \text{purely sync } \text{every } \text{event}
\]
Basic Hybrid Language
Start with Lucid Synchrone (subset); add first-order ODEs with reset.

\[
\dot{x}(t) = f(t, x) \\
x(0) = x_i
\]

instantaneous derivatives

variables

initial values

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

\[
der x = e_d \text{ init } x_i \text{ reset } e_1 \text{ every } \text{up}(e_{z_1}) \\
\quad \ldots \quad | \quad e_n \text{ every } \text{up}(e_{z_n})
\]

\[
x = (\text{pre } h + 1) \text{ every } \text{up}(e) \text{ default } e_c \text{ init } e_i \\
\quad | \quad \text{purely sync} \text{ every } \text{event}
\]

Very simple: no clocks, no automata, no higher-order
Bouncing ball program

\[ \dot{v} = -\frac{g}{m} \quad v(0) = v_0 \]

\[ \dot{h} = v \quad h(0) = h_0 \]

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

\( \mathbb{R} \) + infinitesimals (\( \partial \))

non-standard reals

\( \star \mathbb{R} \)
Semantics

reals

\[ \mathbb{R} \]

non-standard reals

\[ \mathbb{R}^\ast \]

+ infinitesimals (\( \partial \))

\[ \cdots < t - 3\partial < t - 2\partial < t - \partial < t < t + \partial < t + 2\partial < t + 3\partial < \cdots \]
Semantics

deals

\[ \mathbb{R} \]

\[ \mathbb{R} \] + infinitesimals \((\partial)\)

\[ \mathbb{R} \star \]

\[ \cdots < t - 3\partial < t - 2\partial < t - \partial < t < t + \partial < t + 2\partial < t + 3\partial < \cdots \]

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

\[
\text{integ}_{\#}(T)(s)(s_0)(hs)(t) = s'(t) \quad \text{where}
\]

\[
s'(t) = s_0(t) \quad \text{if } t = \min(T)
\]

\[
s'(t) = s'(^* t) + \partial s(^* t) \quad \text{if } \text{handler}_{\#}(T)(hs)(t) = \text{NoEvent}
\]

\[
s'(t) = v \quad \text{if } \text{handler}_{\#}(T)(hs)(t) = \text{Xcrossing}(v)
\]

\[
\text{up}_{\#}(T)(s)(t) = \text{false} \quad \text{if } t = \min(T)
\]

\[
\text{up}_{\#}(T)(s)(t^*) = \text{true} \quad \text{if } (s(^* t) \leq 0) \land (s(t) > 0) \text{ and } (t \in T)
\]

\[
\text{up}_{\#}(T)(s)(t^*) = \text{false} \quad \text{otherwise}
\]

\[
\cdots \quad \cdots \quad \cdots
\]
Semantics

*real*ns non-standard real*ns

\[ R + \text{infinitesimals } (\partial) \quad \rightarrow \quad \star R \]

\[ \cdots < t - 3\partial < t - 2\partial < t - \partial < t < t + \partial < t + 2\partial < t + 3\partial < \cdots \]

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

\[
\begin{align*}
\text{integr}^{\#}(T)(s)(s_0)(hs)(t) &= s'(t) \quad \text{where} \\
&s'(t) = s_0(t) \quad \text{if } t = \text{min}(T) \\
&s'(t) = s'(^{\bullet}t) + \partial s(^{\bullet}t) \quad \text{if } \text{handler}^{\#}(T)(hs)(t) = \text{NoEvent} \\
&s'(t) = v \quad \text{if } \text{handler}^{\#}(T)(hs)(t) = \text{Xcrossing}(v)
\end{align*}
\]

\[
\begin{align*}
\text{up}^{\#}(T)(s)(t) &= \text{false} \quad \text{if } t = \text{min}(T) \\
\text{up}^{\#}(T)(s)(^{\bullet}t) &= \text{true} \quad \text{if } (s(^{\bullet}t) \leq 0) \land (s(t) > 0) \land (t \in T) \\
\text{up}^{\#}(T)(s)(t^{\bullet}) &= \text{false} \quad \text{otherwise}
\end{align*}
\]

...
Semantics

\[ \mathbb{R} + \text{infinitesimals} (\partial) \quad \text{to} \quad \star \mathbb{R} \]

\[ \cdots < t - 3\partial < t - 2\partial < t - \partial < t < t + \partial < t + 2\partial < t + 3\partial < \cdots \]

- dense and discrete
- base clock for both continuous and discrete behaviors
- \( \forall t. \) \( \ast t \) is the previous instant, \( t^* \) is the next instant

\[
\begin{align*}
\text{integr}^\#(T)(s)(s_0)(hs)(t) &= s'(t) & \text{where} \\
\hspace{1cm} s'(t) &= s_0(t) & \text{if} \ t = \text{min}(T) \\
\hspace{1cm} s'(t) &= s'(\ast t) + \partial s(\ast t) & \text{if} \ \text{handler}^\#(T)(hs)(t) = \text{NoEvent} \\
\hspace{1cm} s'(t) &= v & \text{if} \ \text{handler}^\#(T)(hs)(t) = \text{Xcrossing}(v) \\
\text{up}^\#(T)(s)(t) &= \text{false} & \text{if} \ t = \text{min}(T) \\
\text{up}^\#(T)(s)(t^*) &= \text{true} & \text{if} \ (s(\ast t) \leq 0) \land (s(t) > 0) \land (t \in T) \\
\text{up}^\#(T)(s)(t^*) &= \text{false} & \text{otherwise} \\
\end{align*}
\]
Semantics

**reals**

\[ \mathbb{R} \]

\[ + \text{ infinitesimals } (\partial) \]

\[ \star \mathbb{R} \]

\[ \cdots < t - 3\partial < t - 2\partial < t - \partial < t < t + \partial < t + 2\partial < t + 3\partial < \cdots \]

- dense and discrete
- base clock for both continuous and discrete behaviors
- \( \forall t. \) \( \ast t \) is the previous instant, \( t^* \) is the next instant

\[ \text{integr}^\# (T)(s)(s_0)(hs)(t) = s'(t) \]
where

\[ s'(t) = s_0(t) \]
if \( t = \min(T) \)

\[ s'(t) = s'(*t) + \partial s(*t) \]
if \( \text{handler}^\# (T)(hs)(t) = \text{NoEvent} \)

\[ s'(t) = v \]
if \( \text{handler}^\# (T)(hs)(t) = \text{Xcrossing}(v) \)

\[ \text{up}^\# (T)(s)(t) = \text{false} \]
if \( t = \min(T) \)

\[ \text{up}^\# (T)(s)(t^*) = \text{true} \]
if \( (s(*t) \leq 0) \land (s(t) > 0) \) and \( t \in T \)

\[ \text{up}^\# (T)(s)(t^*) = \text{false} \]
otherwise

\[ \cdots \]

\[ \cdots \]
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 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))
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))
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
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))
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

All continuous parts execute in 1st instant
  ▶ type system prevents C inside D
  ▶ no branching or activations

in ((v, h), upz1, (h, v), (dh, dv))
Execution (Simulation)

\[(upz, y, dy) = main_{\sigma}(z, y)\]

\[f(t, y) = \text{let } (_, _, dy) = main_{\sigma}(false, y) \text{ in } dy\]

\[g(t, y) = \text{let } (upz, _, _) = main_{\sigma}(false, y) \text{ in } upz\]

\[d(z, y) = \text{let } (upz, y, _) = main_{\sigma}(z, y) \text{ in } (upz, y)\]

▶ Only \(d\) may have side effects
▶ Neither \(f\), nor \(g\) may change the (internal) discrete state
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 $\mathbb{D}$ only
  - continuous evolutions in $\mathbb{C}$ only
- Reject unreasonable programs
  - where behavior depends ‘too much’ on simulation parameters (like the step size, or number of iterations)
Typing
Unreasonable programs

\[ \text{der } y = 1.0 \text{ init } 0.0 \quad \text{and} \quad x = (0.0 \rightarrow \text{ pre } x) + y \]

\[ x = 0.0 \rightarrow (\text{ pre } x + . 1.0) \quad \text{and} \quad \text{der } y = x \text{ init } 0.0 \]

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

The type language

\[
\begin{align*}
bt & ::= \text{float} \mid \text{int} \mid \text{bool} \mid \text{zero} \\
t & ::= bt \mid t \times t \mid \beta \\
\sigma & ::= \forall \beta_1, \ldots, \beta_n. t \xrightarrow{k} t \\
k & ::= D \mid C \mid A
\end{align*}
\]
Typing

The type language

\[
bt ::= \text{float} | \text{int} | \text{bool} | \text{zero} \\
t ::= bt | t \times t | \beta \\
\sigma ::= \forall \beta_1, ..., \beta_n. t \rightarrow^k t \\
k ::= D | C | A
\]

Initial conditions

\[
(+) : \text{int} \times \text{int} \xrightarrow{A} \text{int} \\
(=) : \forall \beta. \beta \times \beta \xrightarrow{A} \text{bool} \\
\text{if} : \forall \beta. \text{bool} \times \beta \times \beta \xrightarrow{A} \beta \\
\text{pre}(.) : \forall \beta. \beta \xrightarrow{D} \beta \\
\text{fby} . : \forall \beta. \beta \times \beta \xrightarrow{D} \beta \\
\text{up}(.) : \text{float} \rightarrow \text{zero}
\]
Typing

\[ G, H \vdash_{C} \text{der } y = 1.0 \text{ init } 0.0 \quad G, H \vdash_{D} x = (0.0 \text{ fby } (x + 1)) \]
Typing

\[ G, H \vdash_c \text{der } y = 1.0 \text{ init } 0.0 \quad G, H \vdash_D x = (0.0 \text{ fby } (x + 1)) \]

\[ G, H \vdash_? \text{der } y = \cdots \text{ and } x = \cdots \]
Typing

\[
G, H \vdash_c \text{der } y = 1.0 \text{ init } 0.0 \quad G, H \vdash_D x = (0.0 \text{ fby } (x + 1))
\]

\[
G, H \vdash ? \text{ der } y = \cdots \text{ and } x = \cdots \quad \times
\]
Typing

\[ G, H \vdash_C \text{der } y = 1.0 \text{ init } 0.0 \]
\[ G, H \vdash_D x = (0.0 \text{ fby } (x + 1)) \]
\[ G, H \vdash_C x' = (0.0 \text{ fby } (x + 1)) \]
\[ \text{every } \text{up(ez)} \text{ init } 0.0 \]

\[ G, H \vdash? \text{der } y = \cdots \text{ and } x = \cdots \]

\[ X \]
Typing

\( G, H \vdash_C \text{der } y = 1.0 \text{ init } 0.0 \)

\( G, H \vdash D x = (0.0 \text{ fby } (x + 1)) \)

\( G, H \vdash_C x' = (0.0 \text{ fby } (x + 1)) \)

\( G, H \vdash ? \text{ der } y = \cdots \text{ and } x = \cdots \)

\( G, H \vdash ? \text{ der } y = \cdots \text{ and } x' = \cdots \)

\( \text{every up(ez) init } 0.0 \)
Typing

\[ G, H \vdash_C \text{der } y = 1.0 \text{ init } 0.0 \]

\[ G, H \vdash_D x = (0.0 \text{ fby } (x + 1)) \]

\[ G, H \vdash_C x' = (0.0 \text{ fby } (x + 1)) \]

\[ \text{every up(ez) init } 0.0 \]

\[ G, H \vdash? \text{ der } y = \cdots \text{ and } x = \cdots \]

\[ G, H \vdash_C \text{ der } y = \cdots \text{ and } x' = \cdots \]
Typing

\( G, H \vdash C\> \text{der} \> y = 1.0 \text{ init} \> 0.0 \)

\( G, H \vdash D\> x = (0.0 \> \text{fby} \> (x + 1)) \)

\( G, H \vdash C\> x' = (0.0 \> \text{fby} \> (x + 1)) \)

every up(ez) init 0.0

\( G, H \vdash ?\> \text{der} \> y = \cdots \) and \( x = \cdots \) \( \times \)

\( G, H \vdash C\> \text{der} \> y = \cdots \) and \( x' = \cdots \) \( \checkmark \)

\( G, H \vdash ?\> x = \cdots \) and \( x = \cdots \)
Typing

\[ G, H \vdash_C \text{der } y = 1.0 \text{ init } 0.0 \quad G, H \vdash_D x = (0.0 \text{ fby } (x + 1)) \]

\[ G, H \vdash_C x' = (0.0 \text{ fby } (x + 1)) \quad \text{every up(ez) init } 0.0 \]

\[ G, H \vdash? \text{ der } y = \cdots \text{ and } x = \cdots \quad \text{x} \]

\[ G, H \vdash_C \text{ der } y = \cdots \text{ and } x' = \cdots \quad \text{✓} \]

\[ G, H \vdash_D x = \cdots \text{ and } x = \cdots \quad \text{✓} \]
Typing

\[ G, H \vdash_c \text{der } y = 1.0 \text{ init } 0.0 \]
\[ G, H \vdash_D x = (0.0 \ fby (x + 1)) \]
\[ G, H \vdash_C x' = (0.0 \ fby (x + 1)) \]
\[ \text{every up(ez) init } 0.0 \]

\[ G, H \vdash? \text{der } y = \cdots \text{ and } x = \cdots \]
\[ G, H \vdash_C \text{der } y = \cdots \text{ and } x' = \cdots \]
\[ G, H \vdash_D x = \cdots \text{ and } x = \cdots \]

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

\[ h : \text{float} \times \text{float} \rightarrow^k \text{float} \times \text{float} \]

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

\[ j : (\text{float}_D) \times (\text{float}_C) \rightarrow (\text{float}_D) \times (\text{float}_C) \]
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

Ocaml Sundials CVODE interface and compiler available
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