Synchronous data-flow languages: extensions

Benoit Caillaud Julien Forget
benoit.caillaud@inria.fr
julien.forget@univ-lille.fr

École Temps Réel 2026

Outline

1 Automata

2 Arrays

3 Zelus, a synchronous language with ODEs

Mode automata

Many reactive applications consist of several **execution modes**

- For instance, airplane: takeoff, cruise, landing
- Computations depend on the current mode

$\Rightarrow$ Represent modes and mode changes using automata

A simple automaton

```
node f(i:int; c:bool)
  returns (o:int)
let
  automaton
    state S1
      do o=i-1
      until c then S2
    state S2
      do o=i+1
      until c then S1
  end
tel
```

c	true	false	true	false	...
i	4	3	7	5	...
state	S1	S2	S2	S1	...
o	3	4	8	4	...

Semantics

Each tick:

- Automaton is in some state S
- The equation block of S defines current flow values
- Transitions of S are evaluated to determine the next state

Transitions types

- Strong/weak transition:
  - **until** : evaluate equations *then* weak transitions; impacts *next* state
  - **unless**: evaluate strong transitions *then* equations; impacts *current* state
- State reset:
  - **then** : reset memory values
  - **continue** : no reset

Transition types: strong/weak

```
node f(i:int; c:bool)
returns (o:int)
let
  automaton
    state S1
      do o=i+1
      unless c then S2
    state S2
      do o=i-1
      until c then S1
  end
tel
```

c	true	true	false	...
i	6	4	8	...
state	S2	S2	S1	...
o	5	3	9	...

Transition types: memory reset

```
node f(c:bool)
returns (o:int)
let
  automaton
    state S1 do
      o=0 fby (o+1);
      until c continue S2
    state S2 do
      o=5 fby (o+1);
      until c then S1
  end
tel
```

c	false	true	false	...
state	S1	S1	S2	...
o	0	1	5	...

c	...	true	true	false	...
state	...	S2	S1	S2	...
o	...	6	0	7	...

Memory value at previous activation

```
node f(c:bool)
  returns (o:int)
let
  automaton
    state S1
      do o=7 fby (o+1)
      until c continue S2
    state S2
      do o=5 fby (o-1);
      until c continue S1
  end
tel
```

c	true	false	true	false	...
S	S1	S2	S2	S1	...
o	7	5	4	8	...

Memory value at previous instant

```
node f(c:bool)
  returns (o:int)
  var last x:int = 7;
let
  automaton
    state S1
      do x=last x + 1
      until c continue S2
    state S2
      do x=last x - 1
      until c continue S1
  end;
  o=x;
tel
```

c	true	false	true	false	...
S	S1	S2	S2	S1	...
x	7	6	5	6	...
o	7	6	5	6	...

A complete example: stopwatch

- Two buttons: Start/Stop and Reset
- Two outputs: Minutes and Seconds
- Time count toggled by Start/Stop
- If count is stopped, Reset puts time to 0
- Otherwise, Reset toggles freeze display

Outline

1 Automata

2 Arrays

3 Zelus, a synchronous language with ODEs

Operators

- Array type: int^10, int^3^2 (postfix notation$\neq$ C)
- Constructors:
  - [0,3,2]
  - true^3 = [true,true,true]
- Access:
  - Single value: A[i]
  - Slices:

$$A[i..j] = \begin{cases} [A[i], A[i+1], ..., A[j]] & \text{if } i \leq j \\ [A[i], A[i-1], ..., A[j]] & \text{if } j < i \end{cases}$$

- Concatenation: t1@t2

Iterator **map**

T2=**map**<<n>> (f)(T1);

Example

```
const size:int = 4

node incr(i:int) returns (o:int)
let
  o=i+1;
tel

node incrTab(ai:int^size) returns (ao:int^size)
let
  ao=map<<size>> incr(ai);
tel
```

Iterator **fold**

o=**fold**<<n>> (f)(T, init);

Semantics: f(t[n],(f(t[n–1], (..., (f(t[0], init))))))

Example

```
const size : int = 4
node sum(a,b:int) returns (c:int)
let
  c=a+b;
tel

node sumTab(a:int^size) returns (s:int)
let
  s=fold<<size>> sum(a, 0);
tel
```

Example: matrix-vector product

- Reminder:
  - Vector product:

$$PV(u,v) = \sum_{i=0}^{n} u_i.v_i$$

  - Product of matrix $M(m,n)$ by vector $v(n)$ is a vector z of size m, with:

$$\forall i, 1 \leq i \leq m, z_i = PV(M_i, v)$$

- We assume a node $transp(i,j,M)$ that transposes dimensions i and j of matrix M

Outline

1 Automata

2 Arrays

3 Zelus, a synchronous language with ODEs

Modelling an embedded system

- Complete model=controller + plant
- **Hybrid system**
  - Mix of discrete-time + continuous-time
  - Software model + model of the physics

Zelus in a nutshell

- Data-flow equations for the discrete part
- Ordinary differential equations for the continuous part
- Combine the two
- Typing system rejects bizarre models
- Compile to sequential code
- Simulated with an off-the-shelf solver

Continuous values

Example

```
let hybrid causal() = (x, y) where
rec
  der x = 1.0 init 0.0 reset z -> -3.0 *. last y
and
  der y = x init 0.0 reset z -> -4. *. last x
and
  z = up(last y -. 2.0)
```

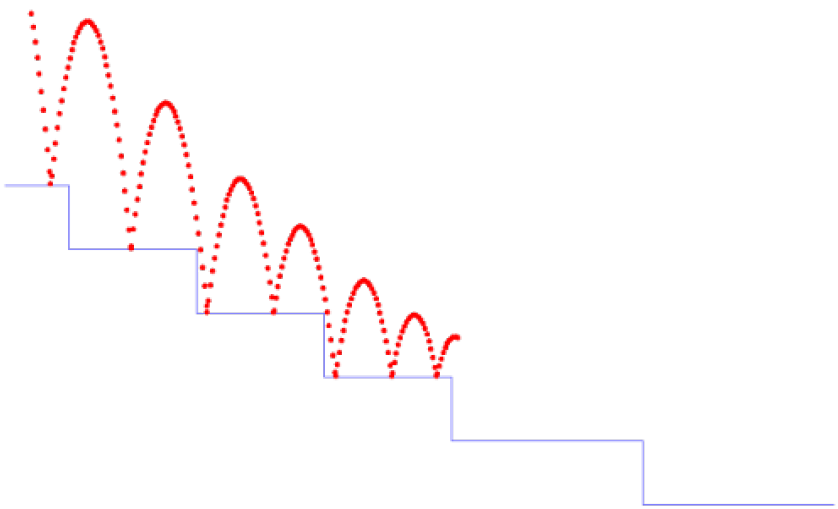
- The evolution of x and y is defined by their derivative
- up(c) is triggered when the continuous value c crosses from negative to positive (**zero-crossing**)
- reset c -> v denotes that the ODE is reset to v when c is true

A concrete example

Bouncing ball

```
let g = 9.81
let loose = 0.8

let hybrid bouncing(x0,y0,x'0,y'0) = (x,y) where
 rec der x = x' init x0
 and der x' = 0.0 init x'0
 and der y = y' init y0
 and der y' = -. g init y'0 reset up(-. y) -> -. loose *. last y'
```



My sources

- *A Conservative Extension of Synchronous Data-flow with State Machines*, EMSOFT'2005
- *Efficient Compilation of Array Iterators for Lustre*, ENTCS'2002
- *Divide and recycle: types and compilation for a hybrid synchronous language*, LCTES'2011
- *Zélus, A synchronous language with ODEs*, `zelus.di.ens.fr`
- *An experiment with Zélus*, MPRI course notes, Marc Pouzet