Zélus: a synchronous language with ODEs

Timothy Bourke[1,2] Marc Pouzet[2,1]

1. INRIA Paris-Rocquencourt

2. École normale supérieure (DI)

`http://www.di.ens.fr/ParkasTeam.html`

HSCC 2013, CPS Week, April 8–11, Philadelphia, USA

Hybrid Systems Modelers

Program complex discrete systems and
their physical environments in a single language

Many tools exist

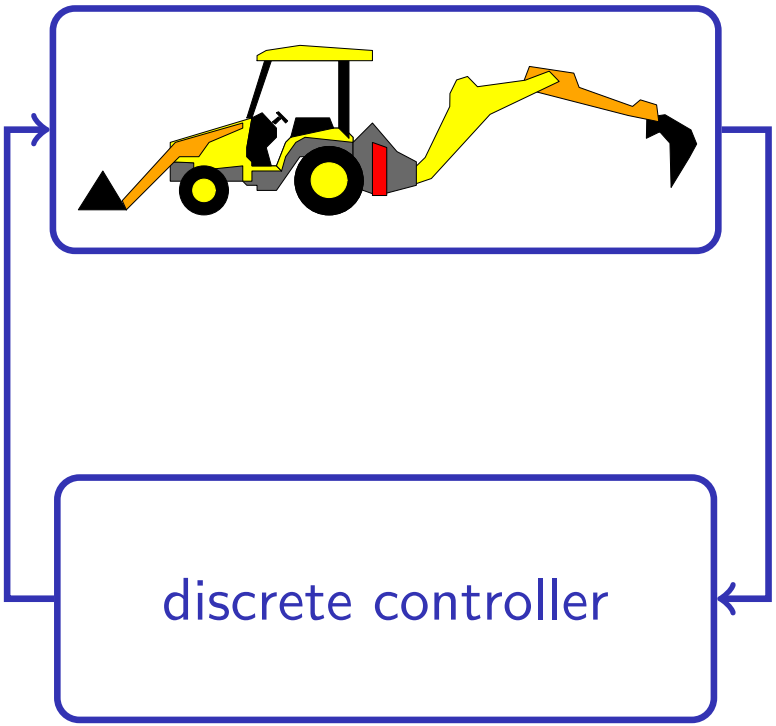
- Simulink/Stateflow, LabVIEW, Modelica, Ptolemy, ...

Focus on programming language issues to improve safety

Our proposal

- Build a hybrid modeler on top of a synchronous language
- Recycle existing techniques and tools
- Clarify underlying principles and guide language design/semantics

Typical system



Discrete controller

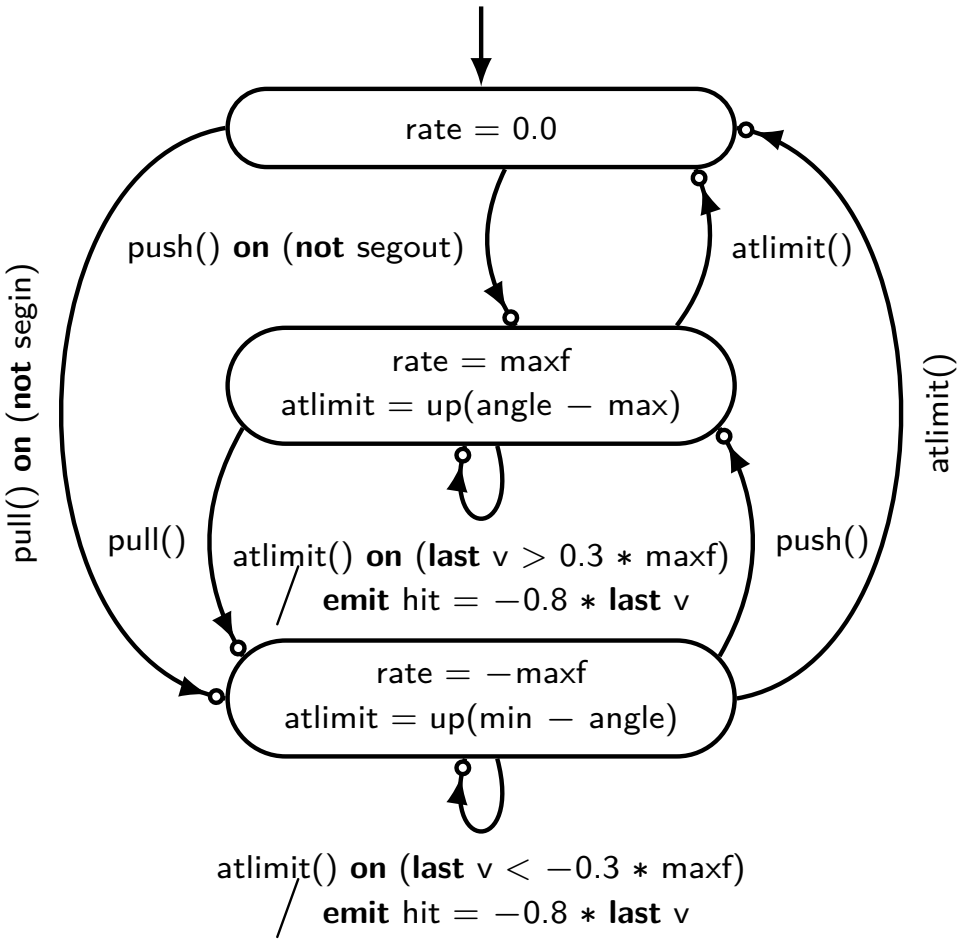
- Dataflow equations
- Hierarchical automata

Physical environment

- ODEs with reset

  der v = (0.7 /. maxf) ∗. error **init** 0.0 **reset** hit(v0) → v0

- Hierarchical hybrid automata



Reuse existing tools and techniques

Synchronous languages (SCADE/Lustre)

- Widely used for critical systems design and implementation
  - mathematically sound semantics
  - certified compilation (DO178C)
- Expressive language for both discrete controllers and mode changes

Off-the-shelf ODEs numeric solvers

- Sundials CVODE (LLNL) among others, treated as black boxes
- Exploit existing techniques and (variable step) solvers

A conservative extension:

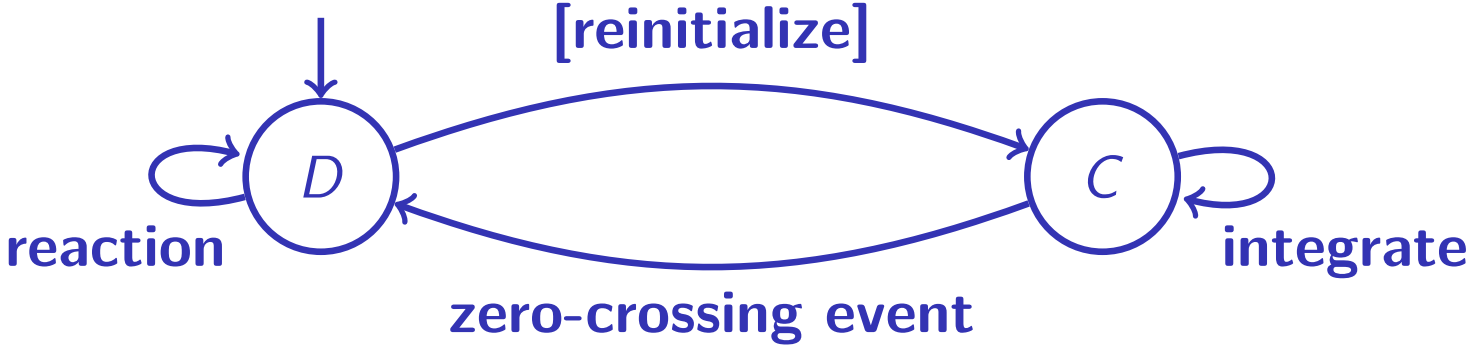
Any synchronous program must be compiled, optimized, and executed as per usual

Type systems to separate continuous from discrete

What is a discrete step?

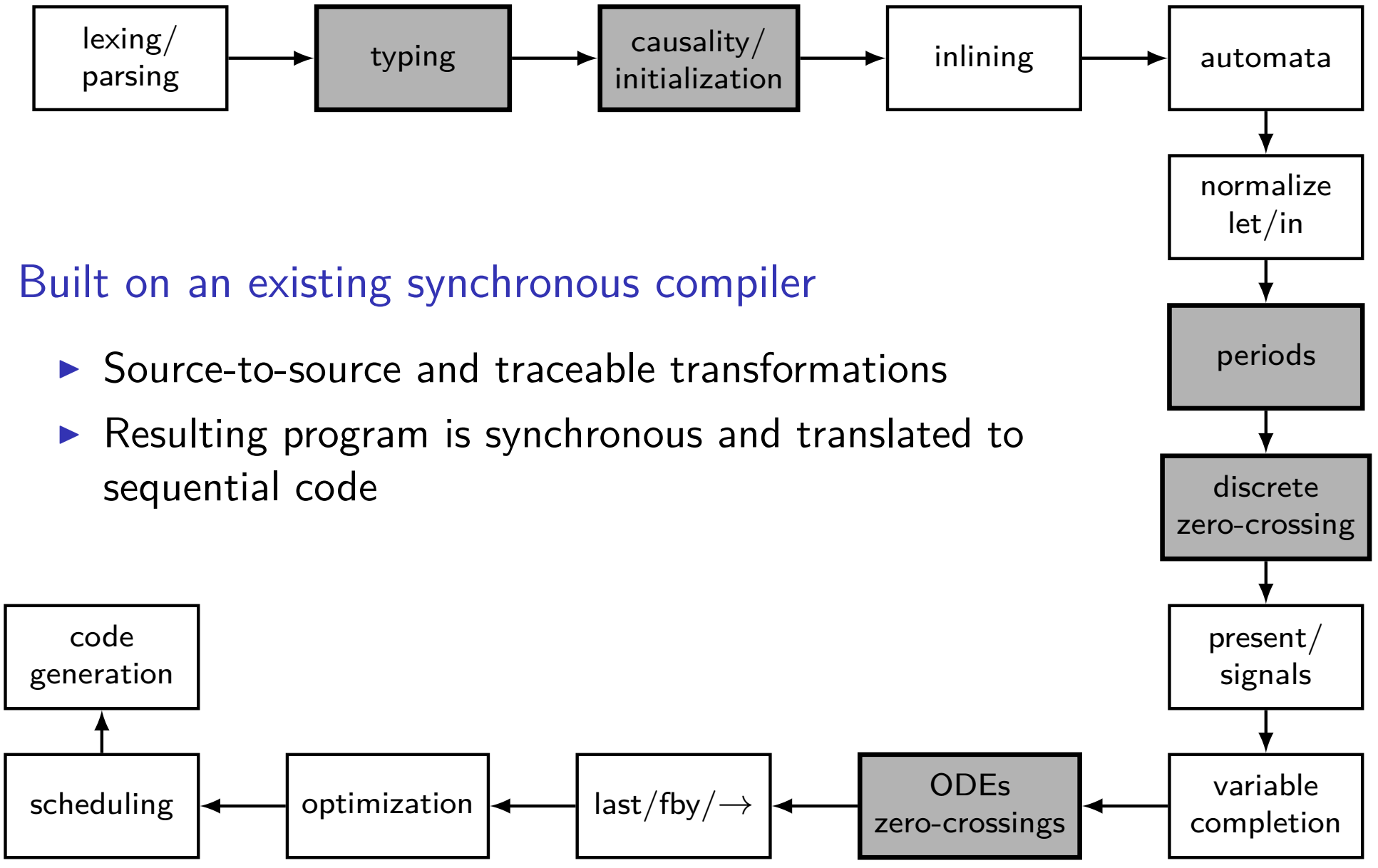
- Reject unreasonable parallel compositions
- Ensure by static typing that discrete changes occur on zero-crossings
- Statically detect causality loops, initialization issues

Simulation engine



$$\sigma' = d_\sigma(t, y) \qquad upz = g_\sigma(t, y) \qquad \dot{y} = f_\sigma(t, y)$$

Compiler architecture



Built on an existing synchronous compiler

- Source-to-source and traceable transformations
- Resulting program is synchronous and translated to sequential code

Comparison with existing tools

Simulink/Stateflow (Mathworks)

- Integrated treatment of automata *vs* two distinct languages
- More rigid separation of discrete and continuous behaviors

Modelica

- Do not handle DAEs
- Our proposal for automata will be integrated into new version 3.4

Ptolemy (E.A. Lee et al., Berkeley)

- A unique computational model: synchronous
- Everything is compiled to sequential code (not interpreted)

Zélus: A Synchronous Language with ODEs

Timothy Bourke **Marc Pouzet**

INRIA Team PARKAS, École normale supérieure (Paris, France)

http://www.di.ens.fr/ParkasTeam.html

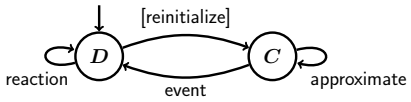
Compiler architecture: source-to-source and traceable transformations

lexing/ parsing → typing → causality/ initialization → inlining → automata → normalize let/in → periods → discrete zero-crossing → present/ signals → variable completion → ODEs zero-crossings → last/fby/→ → optimization → scheduling → code generation

Programming embedded systems and their environments in the same language

- A Lustre-like language with ODEs.
- Dedicated type systems to separate discrete time from continuous time behaviors.
- A compiler architecture based on checkable source-to-source transformations.
- Simulate with an off-the-shelf numeric solver.

Hybrid simulation run-time

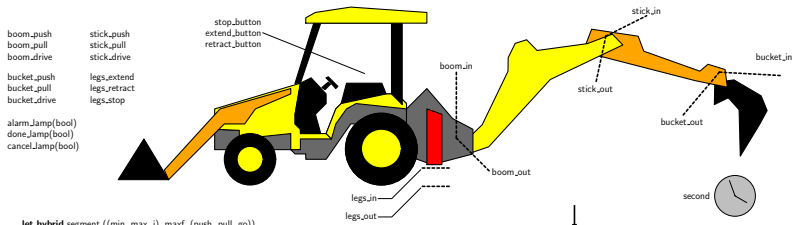


The Type system

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

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

Example system with (hierarchical) Hybrid Automaton



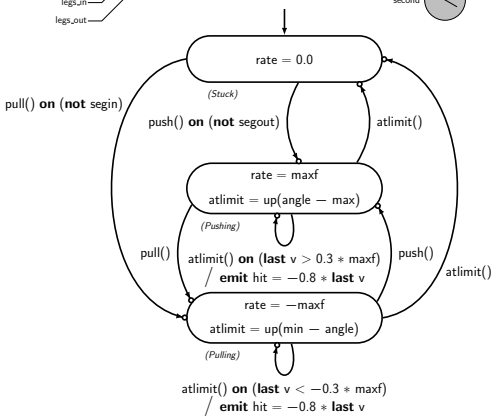

```
let hybrid segment ((min, max, i), maxf, (push, pull, go))
    = ((segin, segout), angle) where

  rec der angle = v init i
  and error = v_r − v
  and der v = (0.7 /. maxf) ∗. error +. 0.3 ∗. z init 0.0
            reset hit(v0) → v0
  and der z = error init 0.0 reset hit(_) → 0.0
  and v_r = if go then rate else 0.0
  and (segin, segout) = (angle <= min, angle >= max)

  and automaton
    | Stuck →
      do
        rate = 0.0
      until push() on (not segout) then Pushing
      else  pull() on (not segin) then Pulling

    | Pushing → local atlimit in
      do
        rate = maxf
        and atlimit = up(angle − max)
      until atlimit() on (last v > 0.3 ∗ maxf) then
            do emit hit = −0.8 ∗. last v in Pushing
      else atlimit() then Stuck
      else pull() then Pulling

    | Pulling → local atlimit in
      do
        rate = −. maxf
        and atlimit = up(min − angle)
      until atlimit() on (last v < −0.3 ∗. maxf) then
            do emit hit = −0.8 ∗. last v in Pulling
      else atlimit() then Stuck
      else push() then Pushing
```



Hybrid Systems: Computation and Control

9–11 April 2013

Philadelphia, USA