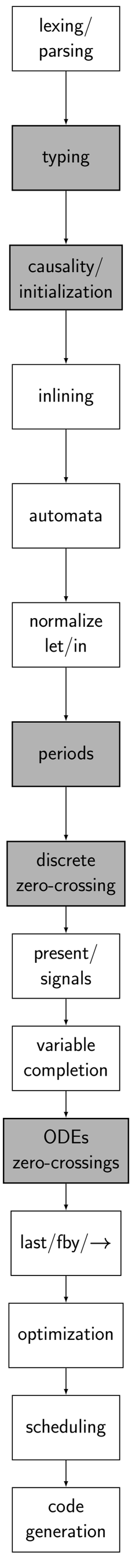
# Zélus: A Synchronous Language with ODEs

**Timothy Bourke  Marc Pouzet**

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

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

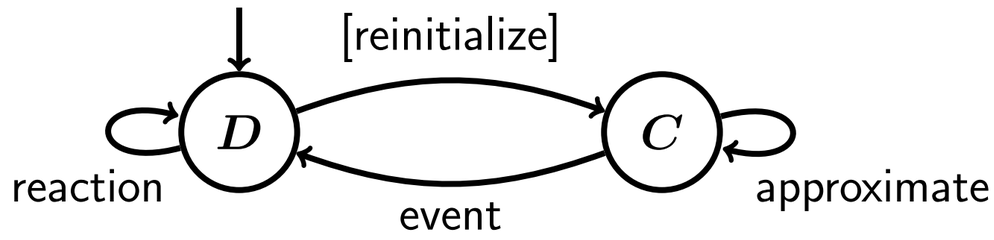



## Compiler architecture: source-to-source and traceable transformations

## 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

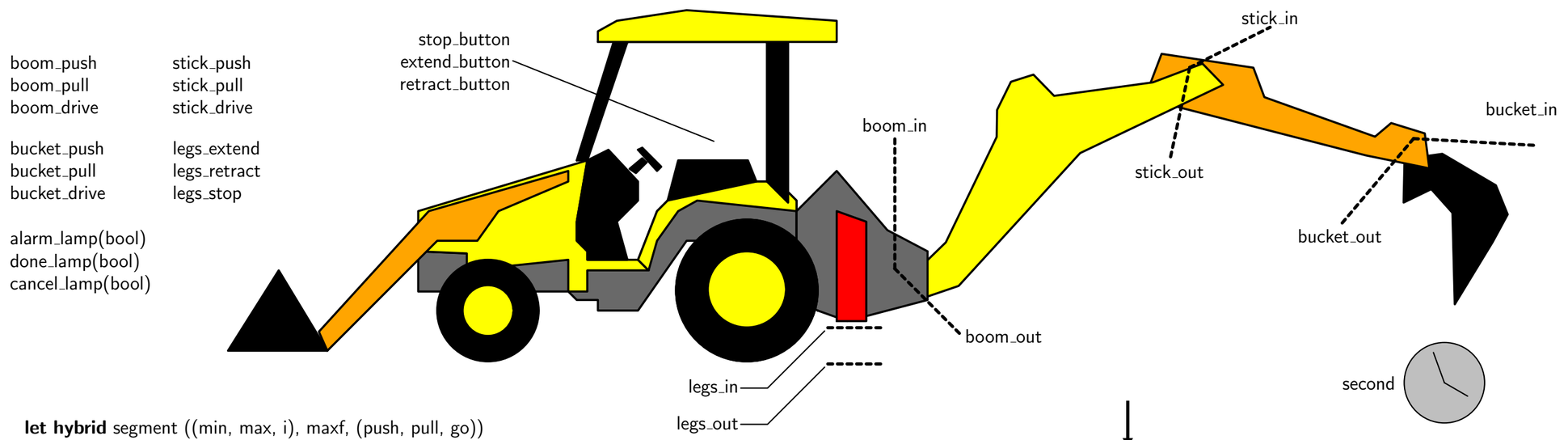



## The Type system

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

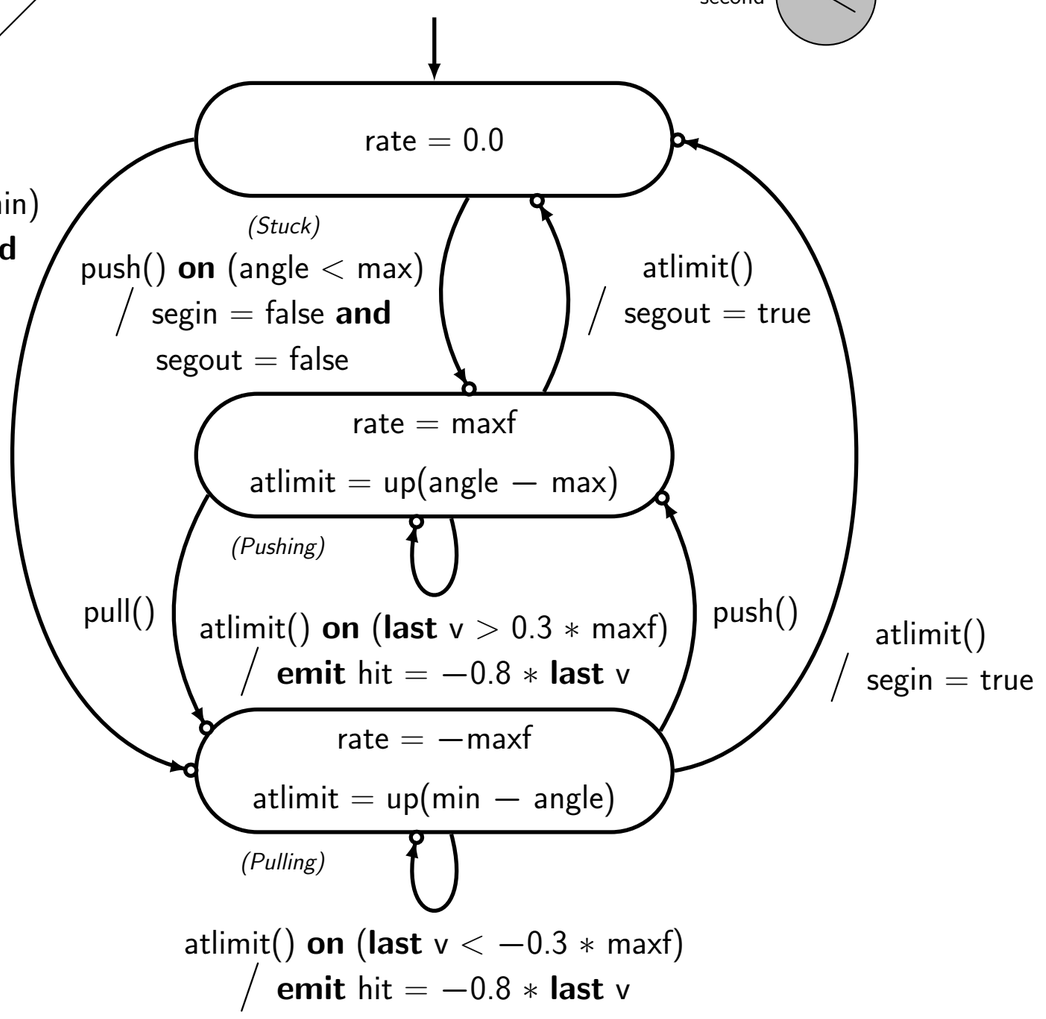

$$
\begin{array}{lll}
\boldsymbol{bt} & ::= & \mathtt{float} \mid \mathtt{int} \mid \mathtt{bool} \mid \mathtt{zero} \\
\boldsymbol{t} & ::= & \boldsymbol{bt} \mid \boldsymbol{t} \times \boldsymbol{t} \mid \beta \\
\boldsymbol{\sigma} & ::= & \forall \beta_1, ..., \beta_n.\boldsymbol{t} \xrightarrow{k}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

## 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 init (segin, segout) = (angle <= min, angle >= max)

 and automaton
   | Stuck → do
       rate = 0.0
     until push() on (angle < max) then
       do segin = false and segout = false in Pushing
     else  pull() on (angle > min) then
       do segin = false and segout = false in 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 do segout = true in 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 do segin = true in Stuck
     else  push() then Pushing
```

pull() **on** (angle > min)
/ segin = false **and**
segout = false




Hybrid Systems: Computation and Control

9–11 April 2013

Philadelphia, USA