Zélus, a Synchronous Language with ODEs for Hybrid Systems

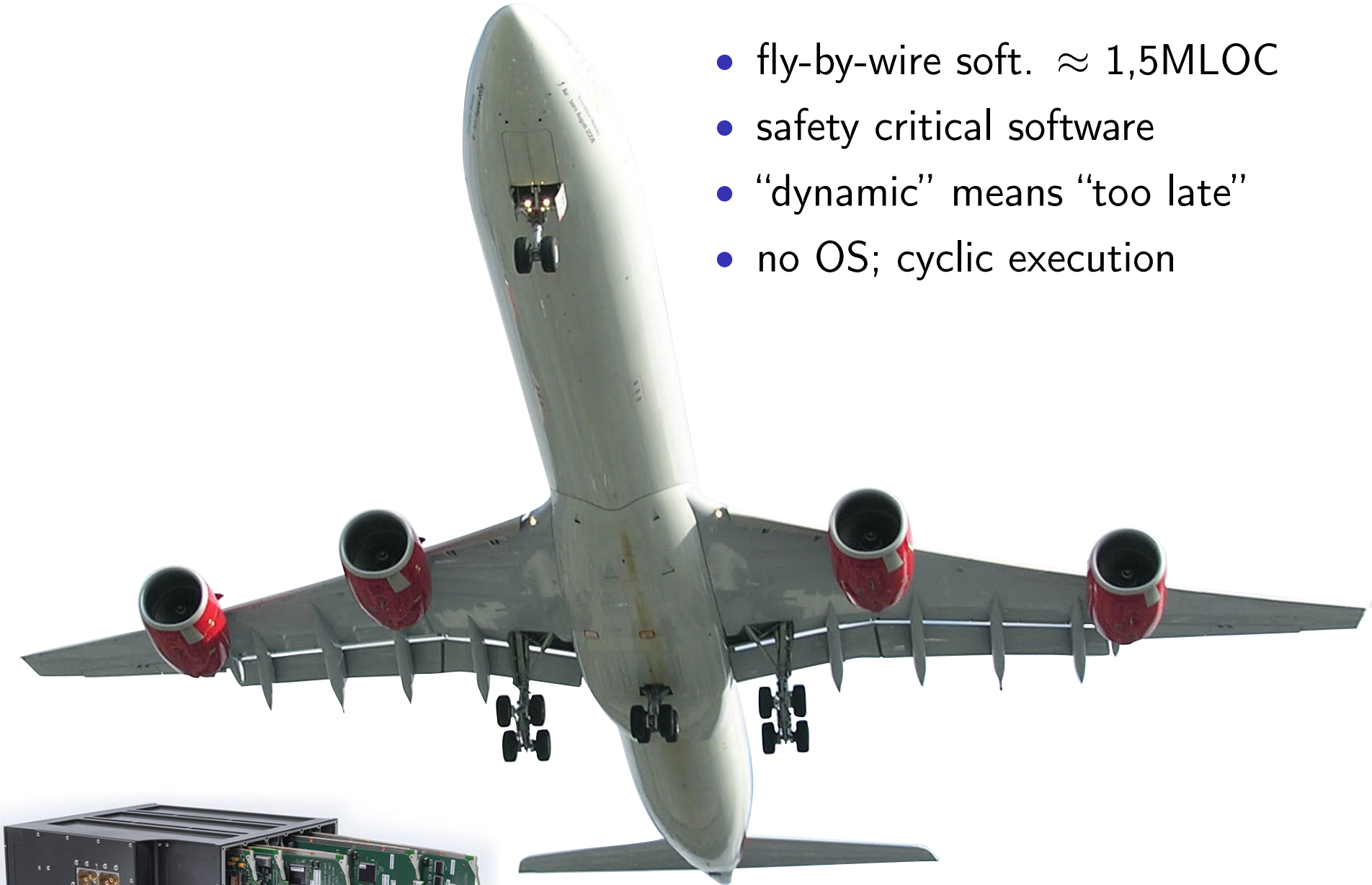
Marc Pouzet

UPMC/DI ENS/Inria Paris

Marc.Pouzet@ens.fr

IBM Watson, PL days
December 4, 2017

Program and control a reactive system?



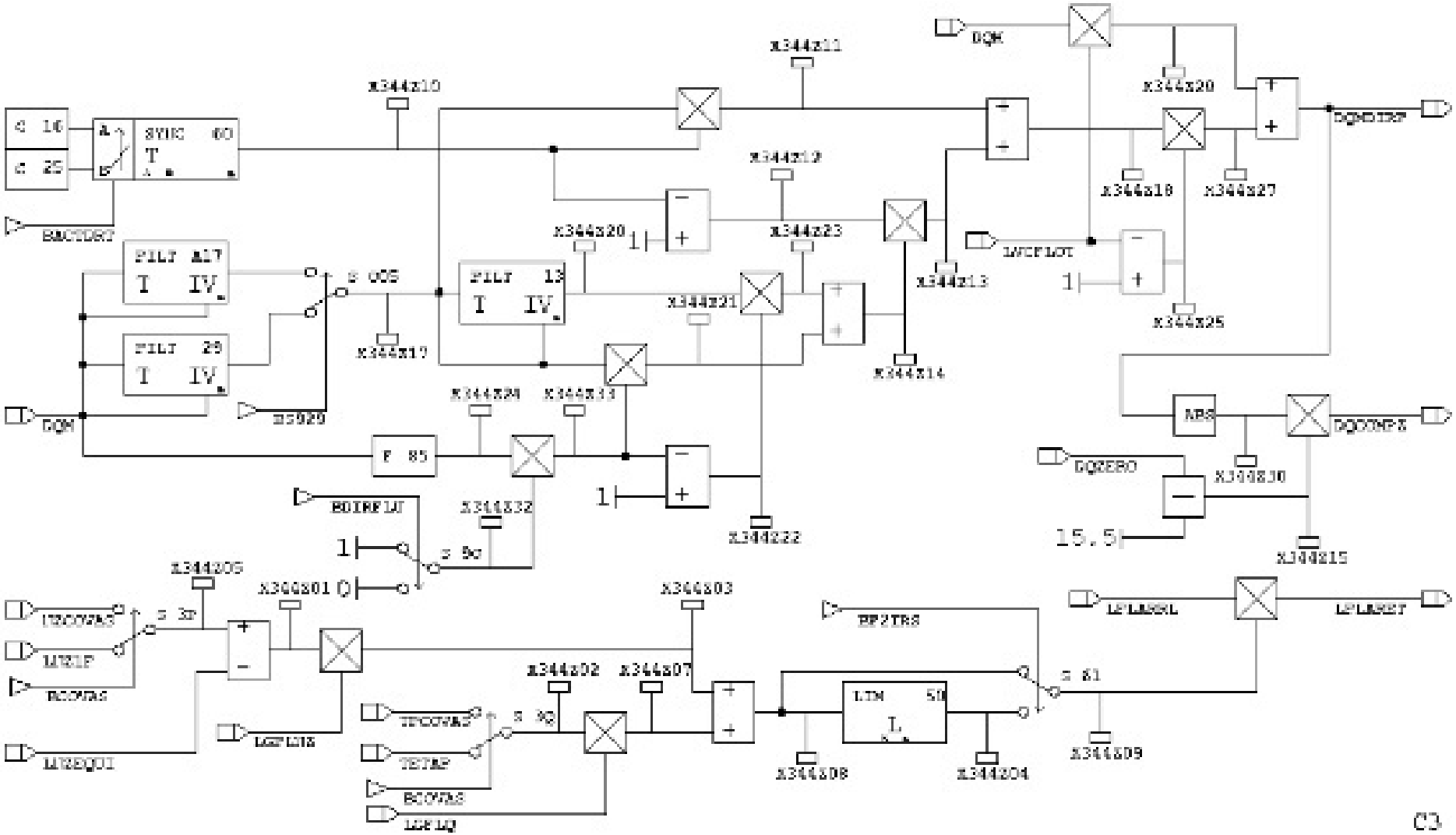
- fly-by-wire soft. $\approx$ 1,5MLOC
- safety critical software
- “dynamic” means “too late”
- no OS; cyclic execution

Write Assembly/C/C++/JavaScript/Python/OCaml/Coq code by hand?

And compare it with what?

What is the spec?

SAO (Spécification Assistée par Ordinateur) — Airbus 80's



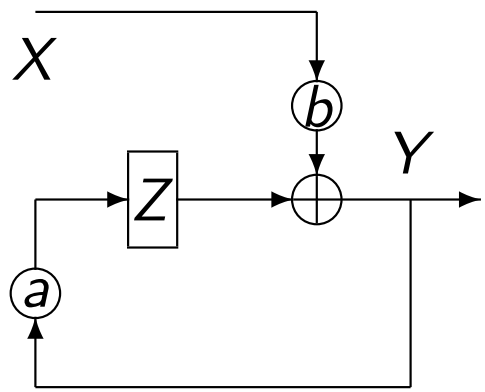
These drawings are very precise maths...

Stream and difference equations, z-transform.

Ordinary Differential equations, Laplace transform.

Example: a linear filter

$$Y_0 = bX_0 \ , \ \forall n \ Y_{n+1} = aY_n + bX_{n+1}$$



...but not executable! Write code and be convinced that it is correct?

How to make those maths executable?

The Synchronous Language Lustre (POPL'87)

A discrete-time system: a stream function; streams are synchronous.

X	1	2	1	4	5	6	...
Y	2	4	2	1	1	2	...
$X + Y$	3	6	3	5	6	8	...
pre X	*nil*	1	2	1	4	5	...
Y ->X	2	2	1	4	5	6	...

The equation $Z = X + Y$ means $\forall n. Z_n = X_n + Y_n$.

Time is logical: inputs X and Y arrive "at the same time"; the output Z is produced "at the same time"

Well... is-it real-time?

Think in worst case: check that the generated code produces its output before the arrival of a new input.

The beautiful idea of Lustre

Program by writing mathematical equations

Analyse/transform/simulate/test/verify them

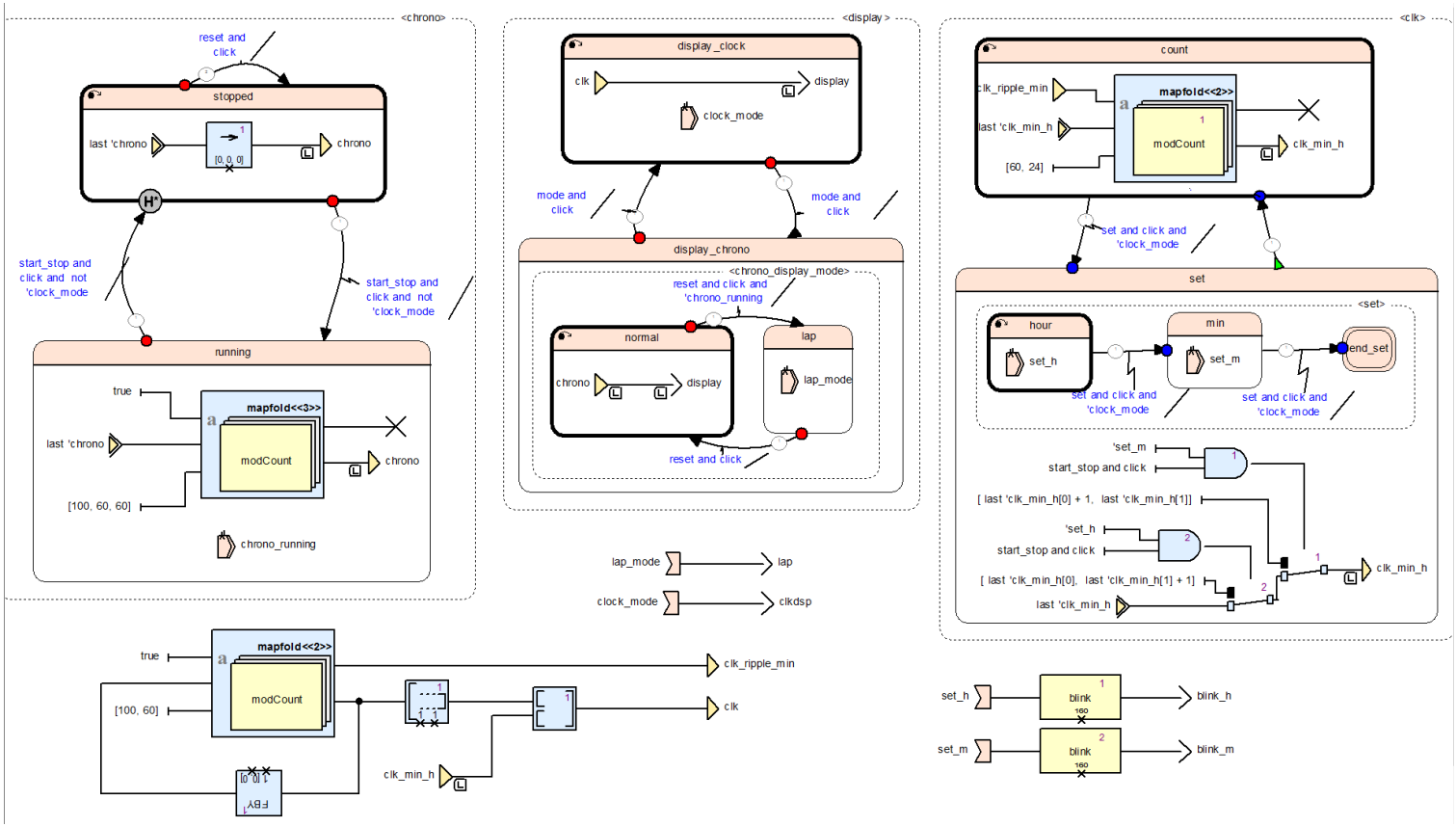
Translate automatically into executable code

Several PL extensions: data-flow + automata, type systems, compilation.

SCADE 6 (ANSYS/Esterel-Technologies)

New language/compiler, in 2008 (with many ideas from Lucid Synchrone)

Now, the *de facto* standard for critical software.



But what about hybrid systems?

A hybrid system = control software + physical model

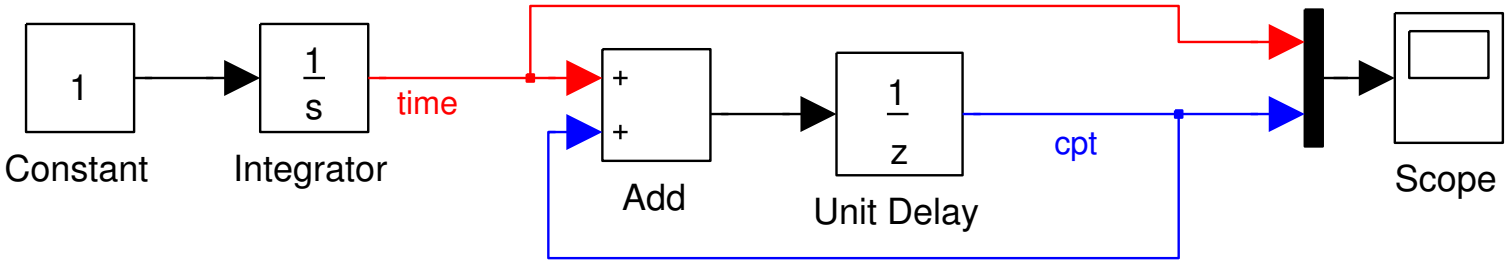
Mixed signals (discrete + continuous)

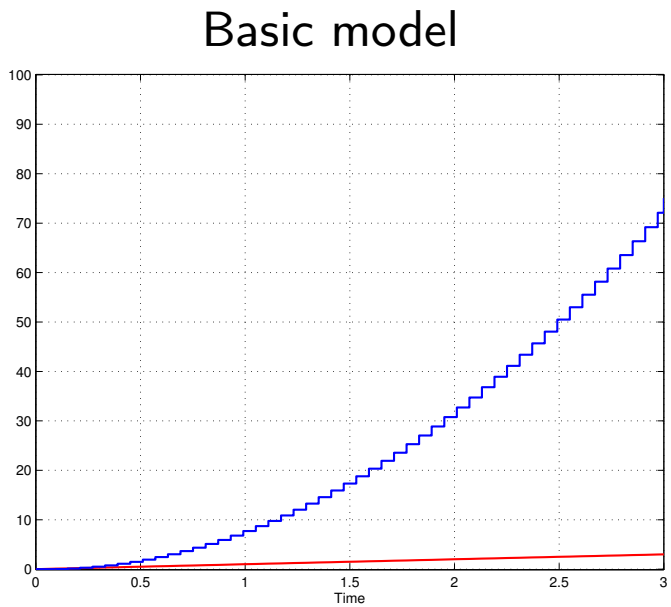
Some models mix discrete logical time and continuous time in an undisciplined manner

They are wrongly typed.

Typing Issues (with Simulink)

Dubious compositions of discrete and continuous time: statically reject?

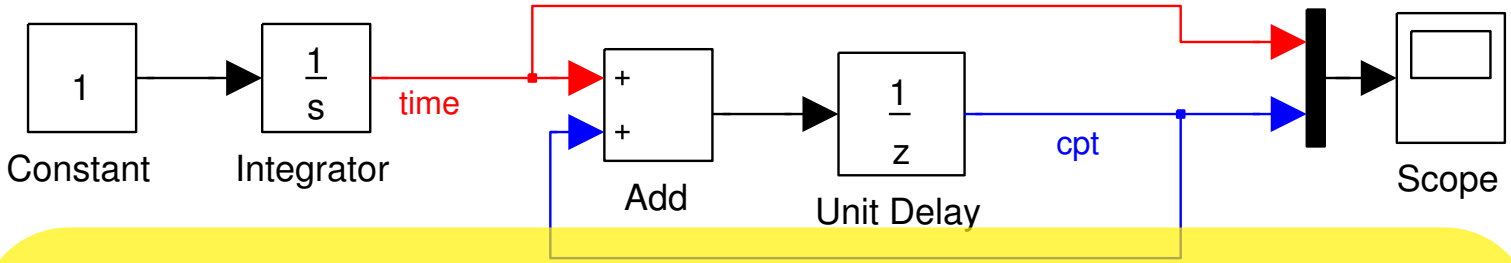




- The value of cpt depends on the steps chosen by the solver

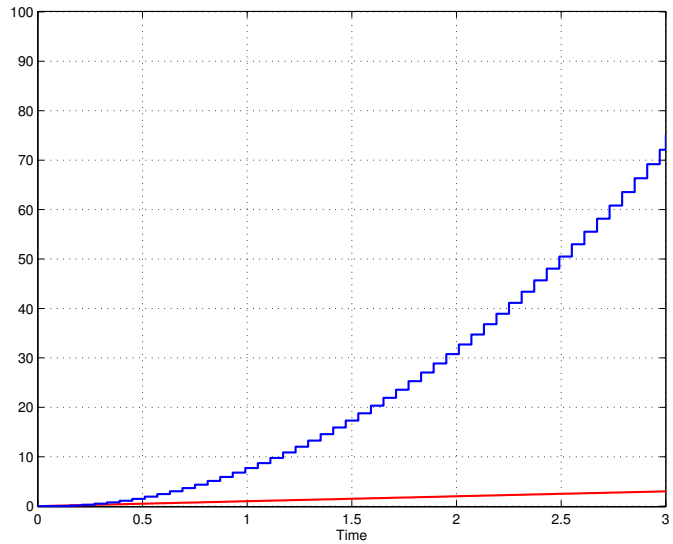
Typing Issues (with Simulink)

Dubious compositions of discrete and continuous time: statically reject?



discrete time is that of the simulation engine

Basic model



- The value of cpt depends on the steps chosen by the solver

Wrongly typed models

Design type systems to statically reject bizarre models.

Can we formally ensure a property like:

"Well typed programs cannot go wrong" (Robin Milner) ?

What is a wrong model/program?

To study those questions, define a minimalistic language,

consider only constructs for which the semantics is precisely defined,

together with typing constraints to ensure safety properties.

Build a Hybrid Modeler on Synch. Language Principles

Milestones

- An ideal semantics based on *non standard analysis* *[JCSS'12]*
- Lustre with ODEs *[LCTES'11]*
- Typing discrete/continuous *[LCTES'11]*
- Hierarchical automata, both discrete and hybrid *[EMSOFT'11]*
- Causality analysis *[HSCC'14]*
- Sequential code generation *[CC'15]*
- Higher-order, Standard Library (FIR, PID, etc.) *[EMSOFT'17]*

Implemented in Zélus *[HCSS'13]*

`http://zelus.di.ens.fr`

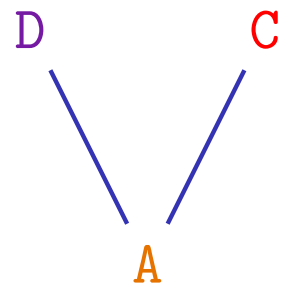
Simulate with an off-the-shelf solver: SUNDIALS CVODE from LLNL

Basic typing [LCTES'11]

A simple ML type system with effects.

The type language

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



Initial conditions

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

The Example in Zélus

```
let hybrid wrong() = (time, cpt) where
  rec
    der time = 1.0 init 0.0
  and
    cpt = 0.0 fby (time +. cpt)

  File ‘‘example.zls’, line 5, character 10-31:
  > cpt = 0.0 fby (time +. cpt)
  > ^^^^^^^^^^^^^^^^^^^^^
  >
  Type error: this is a discrete expression and is expected
  to be continuous.
```

The main theorem [HSCC'14, NAHS'17]

When programs are well typed and causally correct,

signals are continuous during integration

provided imported functions are.

Applications

SCADE Hybrid = SCADE + ODEs at ANSYS/Esterel-Tech.

based on KCG 6.4 (now 6.6), in 2015.

Make it a tool for multi-physics simulation + control software.

http://zelus.di.ens.fr

Zélus Download Examples Publications Contact

Compiler

Zélus is a synchronous language extended with Ordinary Differential Equations (ODEs) to model systems with complex interaction between discrete-time and continuous-time dynamics. It shares the basic principles of Lustre with features from Lucid Synchrone (type inference, hierarchical automata, and signals). The compiler is written

Research

Zélus is used to experiment with new techniques for building hybrid modelers like Simulink/Stateflow and Modelica on top of a synchronous language. The language exploits novel techniques for defining the semantics of hybrid modelers, it provides dedicated type systems to ensure the absence of discontinuities during integration and the