The genesis of Lustre

Nicolas Halbwachs

Verimag/CNRS
Grenoble

1. Prehistory
2. Birth and childhood
3. The industrial adventure
4. Twenty years of research
5. The language at work

Prehistory. . .

- 1979-80 : Joint work on a contract with CROUZET (now Thales) on design methodology for real-time embedded software (case study: avionic anemometer)
- 1980-84 : Joint work on a formalism for specifying and reasoning about time behaviors of systems
  - event = increasing sequence of dates
    (continuous or discrete time)
  - variable = event + sequence of values
  - + many formal tools (counters, formal power series, . . .)

. . . in a peaceful athmosphere

. . . in a peaceful athmosphere

- No industrial partners

. . . in a peaceful athmosphere

- No industrial partners
- No scientific policy

. . . in a peaceful athmosphere

- No industrial partners
- No scientific policy
- No European projects

. . . in a peaceful athmosphere

- No industrial partners
- No scientific policy
- No European projects
- No money

. . . in a peaceful athmosphere

- No industrial partners
- No scientific policy
- No European projects
- No money
- No machines

1. Prehistory
2. **Birth and childhood**
3. The industrial adventure
4. Twenty years of research
5. The language at work

Birth and childhood

Paul’s basic idea

- Embedded software replaces previous technologies

  analog systems, switches networks, hardware...

Birth and childhood

Paul’s basic idea

- Embedded software replaces previous technologies

  analog systems, switches networks, hardware...

- Most embedded software is not developed by computer scientists, but rather by control engineers used with previous technologies (and this is still true!)

Birth and childhood

Paul’s basic idea

- Embedded software replaces previous technologies

  analog systems, switches networks, hardware...

- Most embedded software is not developed by computer scientists, but rather by control engineers used with previous technologies (and this is still true!)
- These people are used with specific formalisms:

  differential or finite-difference equations, analog diagrams, “block-diagrams”...

Birth and childhood

Paul's basic idea

- Embedded software replaces previous technologies

  analog systems, switches networks, hardware...
- Most embedded software is not developed by computer scientists, but rather by control engineers used with previous technologies (and this is still true!)
- These people are used with specific formalisms:

  differential or finite-difference equations, analog diagrams, “block-diagrams”...
- These data-flow formalisms enjoy some nice properties:

  simple formal (functional and temporal) semantics, implicit parallelism

Birth and childhood

Paul's basic idea

- Embedded software replaces previous technologies

  analog systems, switches networks, hardware...
- Most embedded software is not developed by computer scientists, but rather by control engineers used with previous technologies (and this is still true!)
- These people are used with specific formalisms:

  differential or finite-difference equations, analog diagrams, “block-diagrams”...
- These data-flow formalisms enjoy some nice properties:

  simple formal (functional and temporal) semantics, implicit parallelism

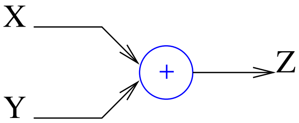
Specialize our formalism into a programming language

(discrete time, executable semantics)

Paul’s basic idea (cont.): Lustre in 3 slides

The dataflow paradigm:

$Z = X + Y$ or



means $\forall t, Z(t) = X(t) + Y(t)$

Variables are functions of time.

Discrete time: Variable = sequence of values + clock

x_n: value of X at “instant” n of its clock.

Paul's basic idea: Lustre in 3 slides (cont.)

Program = system of equations : x = exp

Expressions made of constants (constant sequences), variables, usual operators and temporal operators:

pre(x) = $(nil, x_0, x_1, \ldots, x_{n-1}, \ldots)$

x -> y = $(x_0, y_1, y_2, \ldots, y_n, \ldots)$

Paul’s basic idea: Lustre in 3 slides (cont.)

Program = system of equations : x = exp

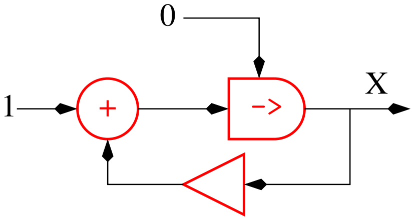
Expressions made of constants (constant sequences), variables, usual operators and temporal operators:

pre(x) = $(nil, x_0, x_1, \ldots, x_{n-1}, \ldots)$

x -> y = $(x_0, y_1, y_2, \ldots, y_n, \ldots)$

Example:

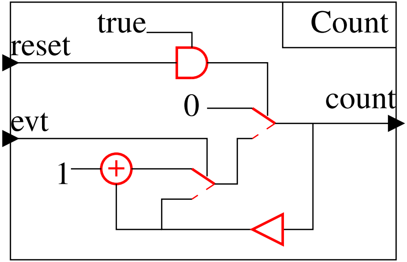
x = 0 ->(pre(x) + 1)



Paul’s basic idea: Lustre in 3 slides (end)

Program structure

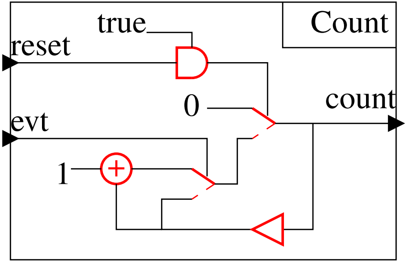
```
node Count(evt, reset: bool)
    returns (count: int);
let  count = if (true -> reset)
             then 0
             else if evt then
                 pre(count)+1
             else pre(count)
tel
```



Paul’s basic idea: Lustre in 3 slides (end)

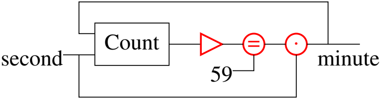
Program structure

```
node Count(evt, reset: bool)
    returns (count: int);
let  count = if (true -> reset)
             then 0
             else if evt then
                  pre(count)+1
             else pre(count)
tel
```



Functional call:

```
nb_sec = Count (second, minute);
minute = true ->
    second and (pre(nb_second)=59);
```



End of childhood

- First definition of the language: Jean-Louis Bergerand's thesis [1986]
- A very favourable context:
  - the synchronous community
  - the industrial demand

The community of synchronous languages

Competition/cooperation with Esterel and Signal teams

- 3 languages born in France, roughly at the same time
- inspired from [Milner81], [Harel83]
- all designed by teams merging competences in control theory and computer science

{ Jean-Paul Rigaud
Jean-Paul Marmorat } and Gerard Berry for Esterel

Albert Benveniste and Paul Le Guernic for Signal

Paul and me for Lustre

1 Prehistory

2 Birth and childhood

3 The industrial adventure

4 Twenty years of research

5 The language at work

The industrial adventure

Strong industrial challenges for embedded software in the eighties (especially in France):

- Control of nuclear power plants: the SPIN system (Nuclear Integrated Protection System)
- Avionics: A320, first full “fly-by-wire” aircraft
- Ground transportation: TGV, VAL,. . .

The industrial adventure (cont.)

- 1984-85 :
  - Schneider-Electric designs the SPIN
    → SAGA, an inhouse design environment based on Lustre:
    graphical editor, automatic code generation
    2 members of our team (Eric Pilaud and Jean-Louis Bergerand) move to Schneider
  - Aerospatiale designs the A320 → SAO

The industrial adventure (cont.)

- 1988-89 :
  Schneider-Electric, Aerospatiale and the Verilog company set up a consortium for the development of SCADE (“Safety Critical Applications Development Environment”), an extended commercial version of SAGA.

  1 member of our team (Daniel Pilaud) moves to Verilog.

The industrial adventure (cont.)

- 1992-96 : Verimag common laboratory with Verilog, for the design of Scade
- 1995 : SCADE qualified DO178-B for the A340/500-600

The industrial adventure (cont.)

- 1992-96 : Verimag common laboratory with Verilog, for the design of Scade
- 1995 : SCADE qualified DO178-B for the A340/500-600
- Industrial vicissitudes:

Verilog bought by CS,

The industrial adventure (cont.)

- 1992-96 : Verimag common laboratory with Verilog, for the design of Scade
- 1995 : SCADE qualified DO178-B for the A340/500-600
- Industrial vicissitudes:

  Verilog bought by CS,
  sold to Telelogic, which finally sells SCADE to . . .

The industrial adventure (cont.)

- 1992-96 : Verimag common laboratory with Verilog, for the design of Scade
- 1995 : SCADE qualified DO178-B for the A340/500-600
- Industrial vicissitudes:

Verilog bought by CS,
sold to Telelogic, which finally sells SCADE to . . .
Esterel-Technologies (2000).

Strong technical and commercial development.

1 Prehistory

2 Birth and childhood

3 The industrial adventure

4 Twenty years of research

5 The language at work

Twenty years of research (in the team)

Language and extensions

- Language definition and formal semantics
  [Caspi, Halbwachs, Bergerand 86, D. Pilaud]
- Mixed imperative/dataflow extensions
  [Maraninchi, Vachon-Jourdan 94, Rémond 01]
- Arrays
  [Rocheteau 92, Maraninchi, Morel 05]
- Higher-order extensions
  [Caspi, Pouzet, . . .]

Twenty years of research (cont.)

Compilation

- to sequential code
  [Halbwachs, Plaice 88, Raymond 91, Rocheteau 92]
- to distributed and non-sequential code
  [Caspi, Buggiani 89, Girault 94, Salem-Habermehl 01, Curic 04, Scaife, Sofronis 06]

Twenty years of research (cont.)

Verification/Validation

- Automatic verification (observers)

  [Halbwachs, Glory-Kerbrat 89, Ratel 91, Raymond, Lesens 97, Jeannet 00, Merchat 05, Gonnord 07]
- Program proof and derivation

  [Caspi, Dumas-Canovas 00, Mikac 05]
- Automatic testing

  [Raymond, Halbwachs, Jahier, Weber 98, Pace, Roux 04]

  and debugging

  [Maraninchi, Gaucher 03]

1. Prehistory
2. Birth and childhood
3. The industrial adventure
4. Twenty years of research
5. The language at work

What worked as expected

What worked as expected

. . . precisely, Paul’s initial idea!!

- synchronous data-flow
- abstract formal semantics
- automatic code generation (single loop)

What did not work as expected

What did not work as expected

- explicit automata (for the code)

What did not work as expected

- explicit automata (for the code)
- clocks (in Lustre) vs. activation conditions (in Scade)
  Restrictive use of the notion of clock, simpler to understand

What did not work as expected

- explicit automata (for the code)
- clocks (in Lustre) vs. activation conditions (in Scade)
  Restrictive use of the notion of clock, simpler to understand
- unexpected compilation constraints:

  node inlining often forbidden

  - code readability
  - separate compiling

What worked unexpectedly

What worked unexpectedly

- Unexpected “non problems”

  Generally, users don’t mind adding some “pre”

What worked unexpectedly

- Unexpected “non problems”

  Generally, users don’t mind adding some “pre”

  - causality (and conditional dependencies)
  - separate compiling: forbid instantaneous feedback of node ouputs to inputs

What worked unexpectedly

- Unexpected “non problems”

  Generally, users don’t mind adding some “pre”

  - causality (and conditional dependencies)
  - separate compiling: forbid instantaneous feedback of node ouputs to inputs

- Unexpected “qualities”

  - program structure, user-defined operators

    (opposed to libraries of predefined operators)
  - compiler efficiency
  - detection of instant loops

Mitigated results about formal verification

- observers well accepted
  (and adopted by Prover in Scade-verifier)
- what do you want to verify?
  Expressing desired properties often considered as a new task in the design process
- Our hope: Automated testing will pave the way to formal verification

Credits (contributors from the Lustre team)

J.-L. Bergerand
Ch. Bodennec
B. Bugiani
P. Caspi
A. Curic
Ch. Dubois
C. Dumas
F. Gaucher
A. Girault
N. Halbwachs
E. Jahier
B. Jeannet
M. Jourdan
A.-C. Kerbrat
F. Lagnier
D. Lesens
F. Maraninchi
Y. Mikac
L. Morel
X. Nicollin
F. Ouabdesselam
G. Pace
D. Pilaud
E. Pilaud
J. Plaice
M. Pouzet
Y. Raoul
Ch. Ratel
P. Raymond
Y. Remond
Y. Roux
F. Rocheteau
R. Salem
N. Scaife
Ch. Sofronis
S. Tripakis
D. Weber