

Lecture 10
An introduction to Lustre

Wednesday Feb 16, 2011

Philipp Rümmer
Uppsala University
Philipp.Ruemmer@it.uu.se

1/25

2/25

- History of Lustre
- Overview of syntax + semantics
- Tutorial; Lustre by example

- Borrowed some material from slides by Pascal Raymond, Nicolas Halbwachs, Cesare Tinelli

History of Lustre

- Invented in 1980's at Verimag (Fr)
- Continuously developed further since then
- Currently:
 - Academic versions + compilers (Lustre V4, Lustre V6)
 - Commercial version (SCADE, Esterel Technologies)

3/25

Early applications

- 1979-1984: first versions of Lustre
- 1986: tool Saga (based on Lustre) to develop the monitoring and emergency stop system of a nuclear plant
- At the same time, a similar tool (SAO) was used to develop the fly-by-wire and flight control of the Airbus A320
- Both tools were later combined by company Verilog → SCADE

4/25

History of Lustre/SCADE

- Nowadays, one of the standard languages for safety-critical systems
 - Avionics, automotive, etc.
 - Certified tools

5/25

Ideas that led to Lustre

- Embedded software replaces previous technologies
 - analogue 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!)

6/25

Ideas that led to Lustre (2)

- These people are used to specific formalisms:
 - differential or finite-difference equations, analogue diagrams, "block-diagrams"...
- These data-flow formalisms enjoy some nice properties:
 - simple formal (functional and temporal) semantics, implicit parallelism

7/25

Ideas that led to Lustre (3)

- Idea: specialize our formalism into a programming language
 - (discrete time, executable semantics)→ **Lustre**
- First versions of Simulink were developed at the same time
→ similar concepts

8/25

Lustre paradigms

- **Dataflow** language
 - similar to Simulink, but textual + time-discrete
 - changes force propagation
- **Synchronous**
 - program can have concurrent tasks, but all tasks run on the same clock; static scheduling (similar to synchronous hardware circuits)
 - good for quick reactions to environment

9/25

Lustre paradigms (2)

- **Declarative**
 - similar to functional languages
 - definitions instead of assignments
- Simple + modular language

10/25

Synchronous language family

- **Lustre**
 - Synchronous + dataflow
- **Esterel**
 - Synchronous + imperative
- **Signal**
 - “Polychronous” → multiple top-level clocks possible

11/25

Tool chains

- Lustre programs can be compiled to different target languages
 - C
 - VHDL → hardware
 - ...
- Good V&V support
 - automatic testing
 - static verification, model checking

12/25

Main concepts

- **Nodes**
 - programs or sub-programs
 - collections of flow definitions
- **Flows/streams**
 - infinite sequence of values → e.g. stream of inputs or outputs
 - represented using variables
 - defined equationally (acyclic)

13/25

Node syntax

```
node name(parameters) returns(vals);
[var local_variable_list;]
let
  flow definition;
  flow definition;
  ...
tel
```

Order is not important!

14/25

Basic types

- **bool**
 - true, false, and, or, not, xor, =>
 - if ... then ... else ...
- **int, real**
 - machine integers, floating-point num.
 - +, -, *, /, div, mod, <>, <, <=, >, >=
- **Tuples**
 - Arbitrary combinations of bool, int, real, & tuple terms
 - Used to return multiple values

15/25

Variable declarations, comments

```
X : int;
A, B : bool;
C : bool; D : int;
```

-- Comments!

16/25

- Command line simulator & verifier
- Fragment of Lustre (v4) language
 - does not support arrays, const, assert, #, when, current, real
 - allows non-standard structures: nodes with no inputs; =, <> can be used on type bool
- Outputs simulations & counterexamples to javascript webpage

17/25

18/25

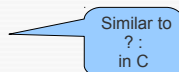
Warning: functional if-then-else

- **Never** write something like this:

```
node Abs (x : int) returns (y : int);
let
  if x >= 0 then y = x else y = -x;
tel
```

- Correct version:

```
node Abs (x : int) returns (y : int);
let
  y = if x >= 0 then x else -x;
tel
```



19/25

20/25

The pre operator

- Access values of variables in the previous cycle:

$$X = (X_0, X_1, X_2, X_3, \dots)$$

$$pre X = (nil, X_0, X_1, X_2, \dots)$$

The followed-by operator ->

- Choose the initial element of a flow:

$$X = (X_0, X_1, X_2, X_3, \dots)$$

$$Y = (Y_0, Y_1, Y_2, Y_3, \dots)$$

$$X \rightarrow Y = (X_0, Y_1, Y_2, Y_3, \dots)$$

- Typical use: `0 -> pre (...)`
- Be careful: `->` binds very weakly:
 - `X and false -> pre Y`
 - means
 - `(X and false) -> pre Y`

21/25

Luke usage

- **Simulation:**

```
luke --node top_node filename
```

- **Verification:**

```
luke --node top_node --verify filename
```

- returns either "Valid. All checks succeeded. Maximal depth was n" or "Falsified output 'X' in node 'Y' at depth n" along with a counterexample.

- *More on Monday*

22/25

Further Lustre features not supported in Luke

- Clocks
 - Used to delay sampling, execution
 - Operators: when, current
- assert, const, #
- Invocation of external functions
- Arrays, recursion, higher-order functions

23/25

SCADE features not supported in Luke

- case :: switching
- fby(x, n, i): n-fold followed-by + pre
 - Guarded delay
 - `i -> pre (i -> pre ...)`
- conduct
 - Guarded clock change

24/25

Further reading

- **A tutorial of Lustre:**

<http://www-verimag.imag.fr/~halbwach/PS/tutorial.ps>

- **Slides by Pascal Raymond, Nicolas Halbwachs:**

<http://www-verimag.imag.fr/~raymond/edu/eng/lustre-a.pdf>

<http://pop-art.inrialpes.fr/~girault/Synchron06/Slides/halbwachs>