The Lustre language

Pascal Raymond
Verimag-CNRS

MOSIG - Embedded Systems

Data-flow approach

- A program = a network of operators connected by wires
- Rather classical (control theory, circuits)

\[
\text{node Average}(X, Y : \text{int})
\]
\[
\text{returns } (A : \text{int}); \quad \text{let}
\]
\[
A = (X + Y) / 2 ; \quad \text{tel}
\]

- Synchronous: discrete time = \(\mathbb{N}\)

\[
\forall t \in \mathbb{N} \quad A_t = (X_t + Y_t)/2
\]

- Full parallelism: nodes are running concurrently
Another version

node Average(X, Y : int)
returns (A : int);
var S : int;
let
    A = S / 2;
    S = X + Y;
tel

• declarative: set of equations (≠ sequence of assignments)
• a single equation for each output and local variable
• variables are infinite sequences of values

Lustre (textual) and Scade (graphical)
Combinational programs

- Basic types: bool, int, real

- Constants:
  \[ 2 \equiv 2, 2, 2, \ldots \]
  \[ \text{true} \equiv \text{true}, \text{true}, \text{true}, \ldots \]

- Pointwise operators:
  \[ X \equiv x_0, x_1, x_2, x_3, \ldots \]
  \[ Y \equiv y_0, y_1, y_2, y_3, \ldots \]
  \[ X + Y \equiv x_0 + y_0, x_1 + y_1, x_2 + y_2, x_3 + y_3, \ldots \]

- All classical operators are provided

Operator if-then-else

```plaintext
node Max(A, B: real) returns (M: real);
let
  M = if (A >= B) then A else B;
tel
```

Warning: functional “if then else” (≠ control statement)
Delay operator

- Previous operator: \texttt{pre}
  
  \[
  \begin{array}{cccccc}
  X & x_0 & x_1 & x_2 & x_3 & x_4 & \ldots \\
  \texttt{pre}X & \textit{nil} & x_0 & x_1 & x_2 & x_3 & \ldots \\
  \end{array}
  \]

  \(\rightarrow\) i.e. \((\texttt{pre}X)_0\) undefined and \(\forall i \neq 0 (\texttt{pre}X)_i = X_{i-1}\)

- Initialization: \(\rightarrow\)
  
  \[
  \begin{array}{cccccc}
  X & x_0 & x_1 & x_2 & x_3 & x_4 & \ldots \\
  Y & y_0 & y_1 & y_2 & y_3 & y_4 & \ldots \\
  X \rightarrow Y & x_0 & y_1 & y_2 & y_3 & y_4 & \ldots \\
  \end{array}
  \]

  \(\rightarrow\) i.e. \((X \rightarrow Y)_0 = X_0\) and \(\forall i \neq 0 (X \rightarrow Y)_i = Y_i\)


Nodes with memory

- Boolean example: raising edge

  \[
  \begin{array}{l}
  \text{node } \texttt{Edge} \ (X : \texttt{bool}) \ \text{return} \ (E : \texttt{bool}); \\
  \text{let} \\
  E = \texttt{false} \rightarrow X \ \text{and} \ \texttt{not} \ \textit{pre} \ X; \\
  \text{tel}
  \end{array}
  \]

- Numerical example: min and max of a sequence

  \[
  \begin{array}{l}
  \text{node } \texttt{MinMax}(X : \texttt{int}) \\
  \ \text{return} \ (\texttt{min}, \ \texttt{max} : \texttt{int}); \ -- \ \texttt{several} \ \texttt{outputs} \\
  \text{let} \\
  \ \texttt{min} = X \rightarrow \text{if} \ (X < \textit{pre} \ \texttt{min}) \ \text{then} \ X \ \text{else} \ \textit{pre} \ \texttt{min}; \\
  \ \texttt{max} = X \rightarrow \text{if} \ (X > \textit{pre} \ \texttt{max}) \ \text{then} \ X \ \text{else} \ \textit{pre} \ \texttt{max}; \\
  \text{tel}
  \end{array}
  \]
Recursive definition

Examples

• \( N = 0 \rightarrow \text{pre } N + 1 \)
  \[ N = 0, 1, 2, 3, \cdots \]

• \( A = \text{false} \rightarrow \text{not pre } A \)
  \[ A = \text{false}, \text{true}, \text{false}, \text{true}, \cdots \]

• Correct \( \Rightarrow \) the sequence can be computed step by step

Counter-example

• \( X = 1/ (2-X) \)
  • unique (integer) solution: “X=1”
  • but not computable step by step

Sufficient condition: forbid combinational loops

How to detect combinational loops?

Syntactic vs semantic loop

• Example:
  \[
  X = \text{if } C \text{ then } Y \text{ else } A; \\
  Y = \text{if } C \text{ then } B \text{ else } X;
  \]

• Syntactic loop

• But not semantic: \( X = Y = \text{if } C \text{ then } B \text{ else } A \)
  is the unique solution

Correct definitions in Lustre

• Choice: syntactic loops are rejected
  (even if they are “false” loops)
Exercices

• A flow $F = 1, 1, 2, 3, 5, 8, \ldots$ ?

• A node $\text{Switch(on, off: bool)}$ returns $(s: \text{bool})$; such that:
  $\rightarrow s$ raises (from $false$ to $true$) if $\text{on}$
  $\rightarrow s$ falls (from $true$ to $false$) if $\text{off}$
  $\rightarrow s$ is $false$ at the origin
  $\rightarrow$ must work properly even if $\text{off}$ and $\text{on}$ are both true

• A node $\text{Count(reset, x: bool)}$ returns $(c: \text{int})$; such that:
  $\rightarrow c$ is reset to 0 if $\text{reset}$,
  $\rightarrow$ otherwise it is incremented if $\text{x}$,
  $\rightarrow c$ is 0 at the origin

Recursive definition

Solutions

• Fibonacci:
  $f = 1 \rightarrow \text{pre}(f + (0 \rightarrow \text{pre} f))$;

• Bistable:
  $\text{node Switch(on, off: bool)}$ returns $(s: \text{bool})$;
  let
  $s = \text{if(false} \rightarrow \text{pre} s) \text{ then } \text{not} \text{ off else} \text{ on}$;
  tel

• Counter:
  $\text{node Count(reset, x: bool)}$ returns $(c: \text{int})$;
  let
  $c = \text{if reset then 0}$
  $\text{else if} \ x \ \text{then } (0\rightarrow\text{pre} \ c) + 1$
  $\text{else } (0\rightarrow\text{pre} \ c)$;
  tel

Recursive definition
Modularity

Reuse

• Once defined, a user node can be used as a basic operator

• Instantiation is functional-like

• Example (exercise: what is the value?)

\[ A = \text{Count}(\text{true} \rightarrow (\text{pre } A = 3), \text{true}) \]

• Several outputs:

\[
\text{node MinMaxAverage}(x: \text{int}) \text{ returns } (a: \text{int}); \nn\text{var min, max: int}; \n\text{let} \n\quad a = \text{Average}(\text{min}, \text{max}); \n\quad \text{min, max } = \text{MinMax}(x); \n\text{tel}
\]

A complete example: stopwatch

• 1 integer output: displayed time

• 3 input buttons: on off, reset, freeze

\[ \rightarrow \text{on off} \text{ starts and stops the stopwatch} \]

\[ \rightarrow \text{reset} \text{ resets the stopwatch (if not running)} \]

\[ \rightarrow \text{freeze} \text{ freezes the displayed time (if running)} \]

• Find local variables (and how they are computed):

\[ \rightarrow \text{running: bool, a Switch instance} \]

\[ \rightarrow \text{frozen: bool, a Switch instance} \]

\[ \rightarrow \text{cpt: int, a Count instance} \]
node Stopwatch(on_off, reset, freeze: bool) returns (time: int);
var running, frozen: bool; cpt: int;
let
    running = Switch(on_off, on_off);
    frozen = Switch(
        freeze and running,
        freeze or on_off);
    cpt = Count(reset and not running, running);
    time = if frozen then (0 -> pre time) else cpt;

tel

Clocks

Motivation

• Attempt to conciliate “control” with data-flow

• Express that some part of the program works less often

• ⇒ notion of data-flow clock (similar to clock-enabled in circuit)

Sampling: when operator

<table>
<thead>
<tr>
<th>X</th>
<th>4</th>
<th>1</th>
<th>-3</th>
<th>0</th>
<th>2</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
</tbody>
</table>

| X when C | 4 | 0 | 2 | 8 |

• whenever C is false, X when C does not exist
**Projection:** \textbf{current} operator

- One can operate only on flows with the same clock
- \textit{projection} on a common clock is (sometime) necessary

\begin{verbatim}
\begin{tabular}{c|ccccccc}
X & 4 & 1 & -3 & 0 & 2 & 7 & 8 \\
\hline
C & true & false & false & true & true & false & true \\
\end{tabular}
\end{verbatim}

Y = X when C 4 0 2 8
Z = current(Y) 4 4 4 0 2 2 8

**Nodes and clocks**

- Clock of a node instance = clock of its effective inputs
- Sampling inputs = enforce the whole node to run slower
- In particular, sampling inputs \(\neq\) sampling outputs

\begin{verbatim}
\begin{tabular}{c|ccccccc}
C & true & true & false & false & true & false & true \\
\hline
Count((r,true) when C) & 1 & 2 & 3 & 4 \\
Count(r,true) when C & 1 & 2 & 5 & 7 \\
\end{tabular}
\end{verbatim}
Example: stopwatch with clocks

```plaintext
node Stopwatch(on_off, reset, freeze: bool)
returns (time: int);
var running, frozen: bool;
    cpt_ena, tim_ena : bool;
    (cpt: int) when cpt_ena;
let
    running = Switch(on_off, on_off);
    frozen = Switch(
        freeze and running,
        freeze or on_off);
    cpt_ena = true -> reset or running;
    cpt = Count((not running, true) when cpt_ena);
    tim_ena = true -> not frozen;
    time = current(current(cpt) when tim_ena);
tel
```

Clock checking

- Similar to type checking
- Clocks must be named (clocks are equal iff they are the same var)
- The clock of each var must be declared (the default is the base clock)
- \( clk(exp \text{ when } C) = C \iff clk(exp) = clk(C) \)
- \( clk(current \ exp) = clk(clk(exp)) \)
- For any other op:
  \( clk(e1 \text{ op e2}) = C \iff clk(e1) = clk(e2) = C \)
Programming with clocks

- Clocks are the right semantic solution
- However, using clocks is quite tricky (cf. stopwatch)
- Main problem: initialisation
  \[ \text{current}(X \text{ when } C) \] exists, but is undefined until \( C \) becomes true for the first time
- Solution: activation condition
  \[ X = \text{CONDACT}(\text{OP, clk, args, dflt}) \]
  \[ \xrightarrow{\text{not an operator, rather a macro}} \]
  \[ X = \text{if clk then current}(\text{OP(args when clk)}) \]
  \[ \text{else (dflt }\rightarrow\text{ pre X}) \]

\[ \xrightarrow{\text{Provided by Scade (industrial)}} \]

Is that all there is? ________________________________

Dedicated vs general purpose languages

- Synchronous languages are dedicated to reactive kernel
- Not (really) convenient for complex data types manipulation
- Abstract types and functions are imported from the host language (typically C)
  However ...
  - Statically sized arrays are provided
  - Static recursion (Lustre V4, dedicated to circuit)
  - Modules and templates (Lustre V6, dedicated to software)