CS:5810 Formal Methods in Software Engineering

Reactive Systems and the Lustre Language\footnote{Copyright 2015-18, Adrien Champion and Cesare Tinelli, the University of Iowa. These notes are copyrighted materials and may not be used in other course settings outside of the University of Iowa in their current form or modified form without the express written permission of one of the copyright holders. During this course, students are prohibited from selling notes to or being paid for taking notes by any person or commercial firm without the express written permission of one of the copyright holder.}

Part 2
Lustre: a synchronous dataflow language

Design of reactive systems:

- run in an infinite loop, and
- produce an output every $n$ milliseconds
Lustre: a synchronous dataflow language

Design of **reactive** systems:

- run in an infinite loop, and
- produce an output every $n$ milliseconds
Design of **reactive** systems:

- run in an infinite loop, and
- produce an output every $n$ milliseconds
Exercises

Model a switch with two buttons, Set and Reset.

```
node Switch( Set, Reset, Init : bool ) returns ( State : bool );
```

such that:

- pressing Set turns the switch on;
- pressing Reset turns the switch off;
- the initial position of the switch is determined by a third signal Init
  if Set and Reset are initially both unpressed.
Exercises

Model a switch with two buttons, Set and Reset.

```plaintext
node Switch ( Set, Reset, Init : bool ) returns ( State : bool );
```

such that:

- pressing Set turns the switch on;
- pressing Reset turns the switch off;
- the initial position of the switch is determined by a third signal Init if Set and Reset are initially both unpressed.

```plaintext
node Switch ( Set, Reset, Init : bool ) returns ( X : bool );
let
    X = if Set then true
        else if Reset then false
        else (Init -> pre X);
```

Exercises

Model a switch with two buttons, Set and Reset.

\[
\text{node } \text{Switch}( \text{Set}, \text{Reset}, \text{Init} : \text{bool} ) \text{ returns (} \\
\hspace{1cm} \text{State} : \text{bool}) ;
\]

such that:

- pressing Set turns the switch on;
- pressing Reset turns the switch off;
- the initial position of the switch is determined by a third signal Init if Set and Reset are initially both unpressed.

Equivalently:

\[
\text{node } \text{Switch}( \text{Set}, \text{Reset}, \text{Init} : \text{bool} ) \\
\text{returns (} X : \text{bool}) ; \\
\text{let} \\
\hspace{1cm} X = \text{Set or } (\text{not Reset and (Init -> pre X)}) ; \\
\text{tel}
\]
node ??? (r, b: bool) returns (out: int);
let

    out = if r then 0
         else if b then (0 -> pre out) + 1
         else (0 -> pre out);

tel
node ??? (r, b: bool) returns (out: int);

let

    out = if r then 0
    else if b then (0 -> pre out) + 1
    else (0 -> pre out);

tel

\[
\begin{array}{ccc}
  r_0 & b_0 & \text{out at 0} \\
  0 & 1 \\
  \uparrow & \uparrow \\
  \text{out at 0} & \text{out at 1} & \text{out at 2} \\
  1 & 1 \\
  \uparrow & \uparrow \\
  \text{out at 1} & \text{out at 2} & \end{array}
\]
node ??? (r, b: bool) returns (out: int);

let

out = if r then 0
    else if b then (0 -> pre out) + 1
    else (0 -> pre out);

tel
Counter with reset:

```ml
node ??? (r, b : bool) returns (out : int);
let

out = if r then 0
else if b then (0 -> pre out) + 1
else (0 -> pre out);

tel
```

<table>
<thead>
<tr>
<th>r</th>
<th>b</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Counter with reset:

```plaintext
node cnt (r, b: bool) returns (out: int);
var pre_out: int;
let pre_out = 0 -> pre_out;
out = if r then 0
    else if b then pre_out + 1
    else pre_out;
```

test

<table>
<thead>
<tr>
<th>r</th>
<th>b</th>
<th>out at 0</th>
<th>r</th>
<th>b</th>
<th>out at 1</th>
<th>r</th>
<th>b</th>
<th>out at 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Exercises

Counter with reset:

```plaintext
node cnt (r, b: bool) returns (out: int);
var pre_out: int;
let pre_out = 0 -> pre out;
out = if r then 0
    else if b then pre_out + 1
    else pre_out;
tel
```

<table>
<thead>
<tr>
<th>r</th>
<th>b</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
Once defined, a node can be used as a basic operator.
Modularity

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

What does $A$ look like?

$X = \text{true} \rightarrow (\text{pre } A = 3)$

$A = \text{cnt}(X, \text{true})$;
Once defined, a node can be used as a basic operator

What does A look like?

\[ X = \text{true} \rightarrow (\text{pre } A = 3) \]
\[ A = \text{cnt}(X, \text{true}); \]
\[ A = 0, \]
Once defined, a node can be used as a basic operator

What does $A$ look like?

\[
X = \text{true} \rightarrow (\text{pre} \ A = 3) \\
A = \text{cnt}(X, \text{true}); \\
A = 0, 1,
\]
Once defined, a node can be used as a basic operator

What does $A$ look like?

$$X = \text{true} \rightarrow (\text{pre } A = 3)$$

$$A = \text{cnt}(X, \text{true});$$

$$A = 0, 1, 2,$$
Modularity

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

What does $A$ look like?

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

\[
A = \text{cnt}(X, \text{true});
\]

$A = 0, 1, 2, 3,$
Modularity

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

What does $A$ look like?

$X = \text{true} \rightarrow (\text{pre } A = 3)$

$A = \text{cnt}(X, \text{true});$

$A = 0, 1, 2, 3, 0,$
Once defined, a node can be used as a basic operator

What does $A$ look like?

$$X = \text{true} \rightarrow (\text{pre } A = 3)$$

$$A = \text{cnt}(X, \text{true});$$

$$A = 0, 1, 2, 3, 0, 1, 2, 3, 0, 1\ldots$$
A node can have several outputs:

```plaintext
node MinMax( X : real ) returns ( Min, Max : real );
let
  Min = X -> if (X < pre Min) then X else pre Min ;
  Max = X -> if (X > pre Max) then X else pre Max ;
tel

node minMaxAverage ( X: real ) returns ( Y: real ) ;
var Min, Max: real ;
let
  Min, Max = MinMax(X) ;
  Y = (Min + Max)/2.0 ;
tel
```
Complete example: specification

Stopwatch:

- one integer output: \textbf{time} “to display”;
- three input buttons:
  - \textbf{on\_off} starts and stops the stopwatch,
  - \textbf{reset} resets the stopwatch \textbf{if not running},
  - \textbf{freeze} freezes the displayed time \textbf{if running}, cancelled if stopped
Complete example: available nodes

-- Bistable switch
node switch (on, off: bool) returns (state: bool);
let
    state =
        if (false -> pre state) then not off else on;
tel

-- Counts steps if inc is true, can be reset
node counter (reset, inc: bool) returns (out: int);
let
    out =
        if reset then 0
        else if inc then (0 -> pre_out) + 1
        else (0 -> pre_out);
tel

-- Detects raising edges of a signal
node edge (in: bool) returns (out: bool);
let
    out = false -> in and (not pre in);
tel
Unsatisfactory solution not using edge:

```plaintext
node stopwatch (on_off, reset, freeze: bool)
returns (time: int);
var actual_time: int;
    running, frozen: bool;

let

    running = switch(on_off, on_off);
    frozen = switch(
        freeze and running, freeze or on_off
    );
    actual_time = counter(reset and not running, running);
    time = if frozen then (0 -> pre time) else actual_time;
```

Complete example: solution(s)

Satisfactory solution:

```plaintext
node stopwatch (on_off, reset, freeze: bool)
returns (time: int);
var actual_time: int;
    running, frozen,
    on_off_pressed, r_pressed, f_pressed: bool;
let
    on_off_pressed = edge(on_off);
    r_pressed = edge(reset);
    f_pressed = edge(freeze);
    running = switch(on_off_pressed, on_off_pressed);
    frozen = switch(
        f_pressed and running, f_pressed or on_off_pressed
    );
    actual_time = counter(r_pressed and not running, running);
    time = if frozen then (0 -> pre time) else actual_time;
tel
```
Part of these notes are based on the following lectures notes:

The Lustre Language — Synchronous Programming
by Pascal Raymond and Nicolas Halbwachs
Verimag-CNRS