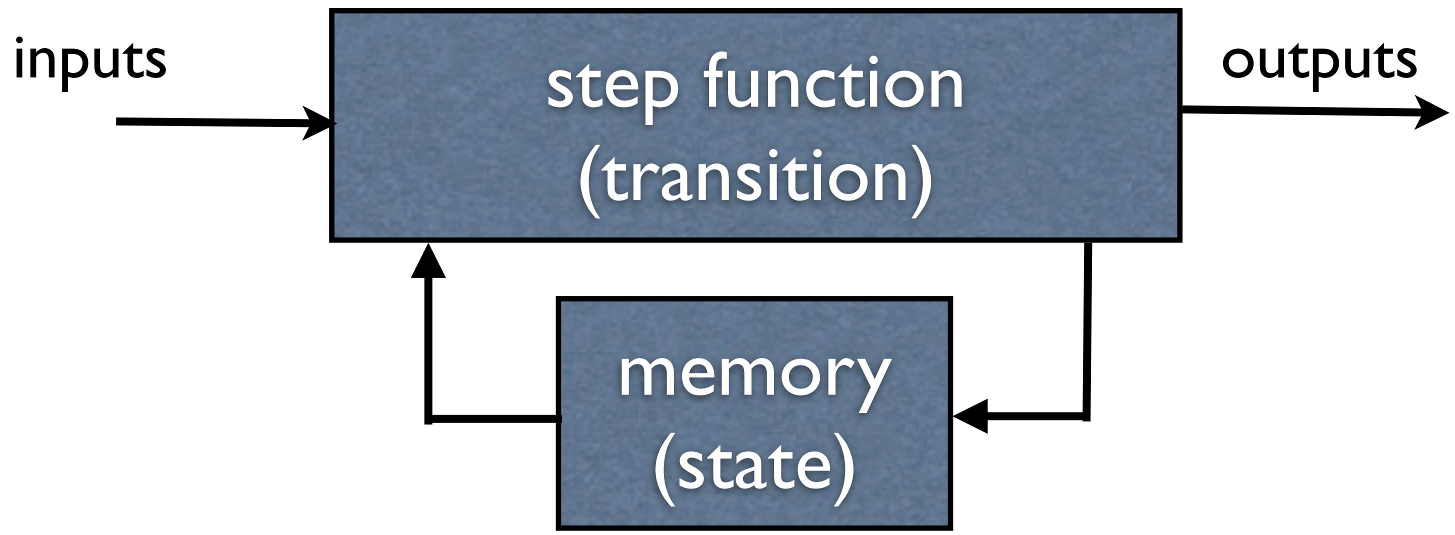
Lustre

- Declarative and deterministic specification language
- Lustre programs = systems of equational constraints between input and output streams
- A Lustre program models an I/O automaton



Implementing a Lustre program

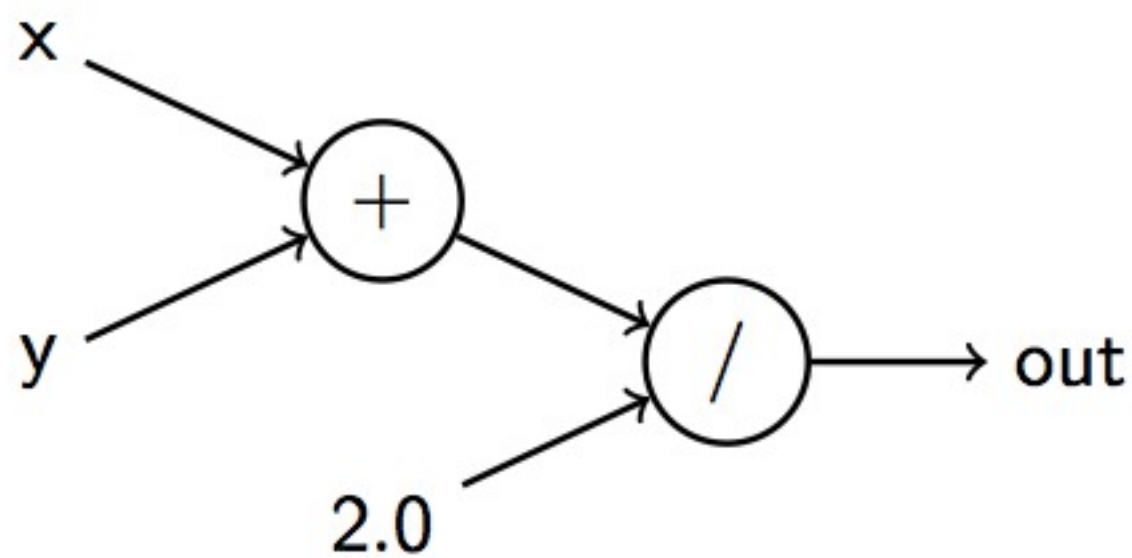
- Read inputs
- Compute next state and outputs
- Write outputs
- Update state

Repeat at every trigger (external event)

Lustre

```
node average (x, y: real) returns (out: real);
let
  out = (x + y) / 2.0;
tel
```

Circuit View



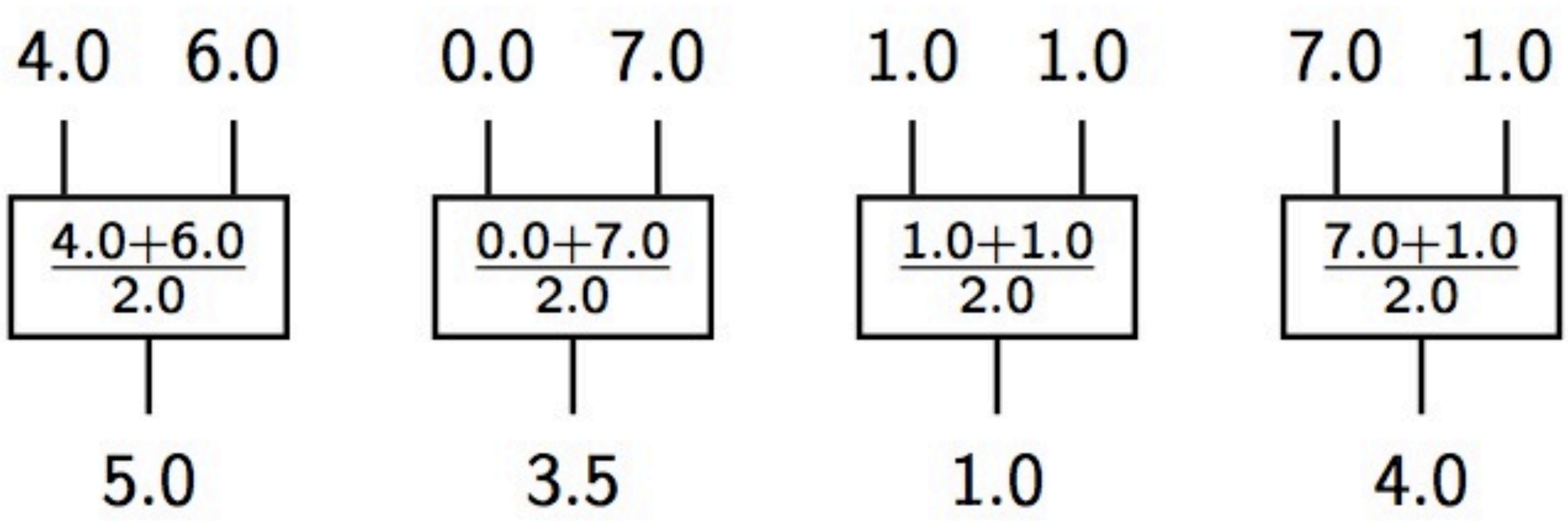
Mathematical View

$$\forall i \in \mathbb{N},\ out_i = \frac{x_i + y_i}{2}$$

Lustre

```
node average (x, y: real) returns (out: real);
let
   out = (x + y) / 2.0;
tel
```

Execution:



clock ticks 0 1 2 3 $\cdots$

Lustre

Memory:

Min / Max

```
node guess (n: int) returns (out1,out2: int);
let
  out1 = n -> if (n < pre out1) then n else pre out1;
  out2 = n -> if (n > pre out2) then n else pre out2;
tel
```

n	4	2	3	0	3	7	...
out1	4	2	2	0	0	0	...
out2	4	4	4	4	4	7	...

Lustre

Memory:

Min / Max

```
node guess (n: int) returns (out1,out2: int);
let
  out1 = n -> if (n < pre out1) then n else pre out1;
  out2 = n -> if (n > pre out2) then n else pre out2;
tel
```

n	4	2	3	0	3	7	...
out1	4	2	2	0	0	0	...
out2	4	4	4	4	4	7	...

Lustre

A Lustre program is a collection of nodes: $L = [N_0, N_1, \ldots, N_m]$

Lustre

A Lustre program is a collection of nodes: $L = [N_0, N_1, \ldots, N_m]$

$$N_i = (\mathcal{I}_i, \mathcal{O}_i, \mathcal{L}_i, Init_i, Trans_i)$$

- $\mathcal{I}_i, \mathcal{O}_i, \mathcal{L}_i$: set of input/output/local vars
- $Init_i, Trans_i$: set of formulas for the initial states and transition relation

Lustre

A Lustre program is a collection of nodes: $L = [N_0, N_1, \ldots, N_m]$

$$N_i = (\mathcal{I}_i, \mathcal{O}_i, \mathcal{L}_i, Init_i, Trans_i)$$

- $\mathcal{I}_i, \mathcal{O}_i, \mathcal{L}_i$: set of input/output/local vars
- $Init_i, Trans_i$: set of formulas for the initial states and transition relation

$$\bigwedge_{i \in \mathbb{N}} v_i = \rho(s_i)$$

- $v_i \in \mathcal{O}_i \cup \mathcal{L}_i$ and $Vars(si) \subseteq \mathcal{I}_i \cup \mathcal{O}_i \cup \mathcal{L}_i$
- s_i arbitrary Lustre expression including node calls $N_j(u_1, \ldots, u_n)$
- ρ function maps expression to expression

$$a \to b \text{ is projected as } \begin{cases} a \text{ in } Init_i \\ b \text{ in } Trans_i \end{cases}$$

Lustre

A Lustre program is a collection of nodes: $L = [N_0, N_1, \ldots, N_m]$

$$N_i = (\mathcal{I}_i, \mathcal{O}_i, \mathcal{L}_i, Init_i, Trans_i)$$

- $\mathcal{I}_i, \mathcal{O}_i, \mathcal{L}_i$: set of input/output/local vars
- $Init_i, Trans_i$: set of formulas for the initial states and transition relation

$$\bigwedge_{i \in \mathbb{N}} v_i = \rho(s_i)$$

- $v_i \in \mathcal{O}_i \cup \mathcal{L}_i$ and $Vars(si) \subseteq \mathcal{I}_i \cup \mathcal{O}_i \cup \mathcal{L}_i$
- s_i arbitrary Lustre expression including node calls $N_j(u_1, \ldots, u_n)$
- ρ function maps expression to expression

$$a \to b \text{ is projected as } \begin{cases} a \text{ in } Init_i \\ b \text{ in } Trans_i \end{cases}$$

- A safety property P is any Lustre expression over the main node N_0

From Lustre to CHC

```
node therm_control (actual: real; up, dn: bool )
        returns (heat, cool : bool)
 var desired, margin : real;
 let
    margin = 1.5;
    desired = 21.0 → if dn then (pre desired) − 1.0
                     else if up then (pre desired) + 1.0
                     else (pre desired);
    cool = (actual − desired) > margin;
    heat = (actual − desired) < −margin;
 tel
```

From Lustre to CHC

```
node therm_control (actual: real; up, dn: bool )
        returns (heat, cool : bool)
 var desired, margin : real;
 let
    margin = 1.5;
    desired = 21.0 → if dn then (pre desired) − 1.0
                      else if up then (pre desired) + 1.0
                      else (pre desired);
    cool = (actual − desired) > margin;
    heat = (actual − desired) < −margin;
 tel
```

$$
\begin{aligned}
&[\ margin = 1.5 \\
&\wedge\ desired = 21.0 \\
&\wedge\ cool = actual - desired > margin \\
&\wedge\ heat = ...] \Rightarrow TC_{init}(actual, up, dn, heat, cool, desired)
\end{aligned}
$$

Initial states

From Lustre to CHC

```
node therm_control (actual: real; up, dn: bool )
          returns (heat, cool : bool)
 var desired, margin : real;
 let
    margin = 1.5;
    desired = 21.0 → if dn then (pre desired) − 1.0
                     else if up then (pre desired) + 1.0
                     else (pre desired);
    cool = (actual − desired) > margin;
    heat = (actual − desired) < −margin;
 tel
```

$$\begin{aligned}&[\ margin = 1.5\\ &\wedge\ desired = 21.0\\ &\wedge\ cool = actual - desired > margin\\ &\wedge\ heat = ...] \Rightarrow TC_{init}(actual, up, dn, heat, cool, desired)\end{aligned}$$

Initial states

$$\begin{aligned}&[\ margin = 1.5\\ &\wedge\ desired' = ite(dn\ (desired - 1.0)\ (ite...))\\ &\wedge\ cool = actual - desired' > margin\\ &\wedge\ heat = ...] \Rightarrow TC_{trans}(actual, up, dn, heat, cool, desired, desired')\end{aligned}$$

Transition relation

From Lustre to CHC

```
node therm_control (actual: real; up, dn: bool )
         returns (heat, cool : bool)
 var desired, margin : real;
 let
    margin = 1.5;
    desired = 21.0 → if dn then (pre desired) − 1.0
                     else if up then (pre desired) + 1.0
                     else (pre desired);
    cool = (actual − desired) > margin;
    heat = (actual − desired) < −margin;
 tel
```

$$
\begin{aligned}
&[\ margin = 1.5 \\
&\wedge\ desired = 21.0 \\
&\wedge\ cool = actual - desired > margin \\
&\wedge\ heat = ...] \Rightarrow TC_{init}(actual, up, dn, heat, cool, desired)
\end{aligned}
$$

Initial states

$$
\begin{aligned}
&[\ margin = 1.5 \\
&\wedge\ desired' = ite(dn\ (desired - 1.0)\ (ite...)) \\
&\wedge\ cool = actual - desired' > margin \\
&\wedge\ heat = ...] \Rightarrow TC_{trans}(actual, up, dn, heat, cool, desired, desired')
\end{aligned}
$$

Transition relation

$$
TC_{init}(actual, up, dn, heat, cool, desired) \Rightarrow Loop(actual, up, dn, heat, cool, desired)
$$

$$
\begin{aligned}
& \quad Loop(actual', up', dn', heat', cool', desired) \\
\wedge & \quad TC_{trans}(actual, up, dn, heat, cool, desired, desired') \\
& \quad \Rightarrow Loop(actual, up, dn, heat, cool, desired')
\end{aligned}
$$

Loop

Example 2

```
node n1(x: int) returns (y: int; z: bool);
var t: int;
let
  t = 0 -> pre(x);
  y = 0 -> pre(x + t);
  z = y > 10;
tel;

node n2(a: int; r: bool) returns (b: int);
let
  b = if r then 0 else a + 1;
tel;

node main(y:bool) returns (prop2: bool);
var
    m_r: bool;
    m_y, m_x : int;
    out : int;
let
    m_y, m_r = n1(m_x);
    m_x = n2(m_y, m_r);
    out = m_x;
    prop2 = out <= 11;
    --!PROPERTY : prop2;
tel;
```

Example 2

```
node n1(x: int) returns (y: int; z: bool);
var t: int;
let
  t = 0 -> pre(x);
  y = 0 -> pre(x + t);
  z = y > 10;
tel;

node n2(a: int; r: bool) returns (b: int);
let
  b = if r then 0 else a + 1;
tel;

node main(y:bool) returns (prop2: bool);
var
    m_r: bool;
    m_y, m_x : int;
    out : int;
let
    m_y, m_r = n1(m_x);
    m_x = n2(m_y, m_r);
    out = m_x;
    prop2 = out <= 11;
    --!PROPERTY : prop2;
tel;
```

Example 2 (cont.)

node n1

$$[\ t = 0 \wedge \ \ y = 0 \wedge \ \ z = y > 10] \Rightarrow N1_{init}(x, y, z)$$

$$[\ t = x' \wedge y = x' + t' \wedge z = y > 10] \Rightarrow N1_{trans}(x, y, z, x', t')$$

node n2

$$[b = (\textbf{ite}\ r\ 0\ (a + 1))] \Rightarrow N2(a, r, b)$$

main

$$[N1_{init}(x, y, z) \wedge N2(y, r, x) \wedge prop = x \leq 11] \Rightarrow Main_{init}(v, prop)$$

$$[N1_{trans}(x, y, z, x', t') \wedge N2(y, r, x) \wedge prop = x \leq 11] \Rightarrow Main_{trans}(v, prop, x', t')$$

loop

$$Main_{init}(v, prop) \Rightarrow Loop(v, prop)$$

$$Loop(v, prop) \wedge Main_{trans}(v, prop') \Rightarrow Loop(v, prop')$$

safety property

$$(\textbf{not}\ prop) \wedge Loop(v, prop) \Rightarrow Error$$

Example 2 (cont.)

node n1

$$[\ t = 0 \wedge \ \ y = 0 \wedge \ \ z = y > 10] \Rightarrow N1_{init}(x, y, z)$$

$$[\ t = x' \wedge y = x' + t' \wedge z = y > 10] \Rightarrow N1_{trans}(x, y, z, x', t')$$

node n2

$$[b = (\textbf{ite}\ r\ 0\ (a + 1))] \Rightarrow N2(a, r, b)$$

main

$$[N1_{init}(x, y, z) \wedge N2(y, r, x) \wedge prop = x \leq 11] \Rightarrow Main_{init}(v, prop)$$

$$[N1_{trans}(x, y, z, x', t') \wedge N2(y, r, x) \wedge prop = x \leq 11] \Rightarrow Main_{trans}(v, prop, x', t')$$

loop

$$Main_{init}(v, prop) \Rightarrow Loop(v, prop)$$

$$Loop(v, prop) \wedge Main_{trans}(v, prop') \Rightarrow Loop(v, prop')$$

safety property

$$(\textbf{not}\ prop) \wedge Loop(v, prop) \Rightarrow Error$$

Check if $Error$ is reachable?

Exercises

Exercises (1)

Design a node

```
node switch (on,off: bool) returns (state: bool);
```

such that:

- `state` raises (false to true) if `on`;
- `state` falls (true to false) if `off`;
- everything behaves as if `state` was false at the origin;
- `switch` must work properly even if `on` and `off` are the same

Exercises (II)

Compute the sequence 1, 1, 2, 3, 5, 8, 13, 21 ...

Fibonacci sequence:

$u_0 = 1$

$u_1 = 1$

$u_n = u_{n-1} + u_{n-2} \quad \text{for } n \geq 2$

Exercises (III)

Stopwatch:

- one integer output: `time` "to display";
- three input buttons:

`on_off`	starts and stops the stopwatch,
`reset`	resets the stopwatch **if not running**,
`freeze`	freezes the displayed time **if running**, cancelled if stopped

Exercises (III)

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
```