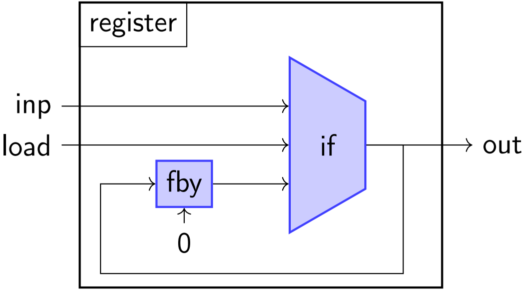
Towards Control Structures in Velus

Basile Pesin,
under the supervision of Timothy Bourke and Marc Pouzet

Inria - PARKAS

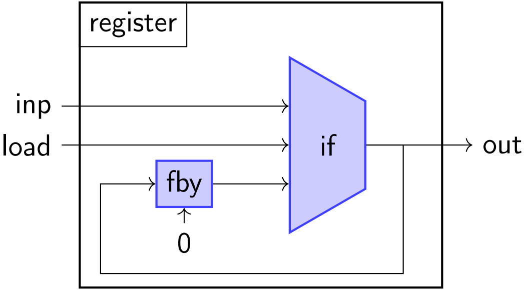
Synchron 2021 - 23 Nov

The Lustre Programming Language



inp	3	4	1	5	2	6	1	1	...
load	F	F	T	F	T	T	F	T	...
out	0	0	1	1	2	6	6	1	...

The Lustre Programming Language

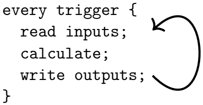


inp	3	4	1	5	2	6	1	1	...
load	F	F	T	F	T	T	F	T	...
out	0	0	1	1	2	6	6	1	...

```
node register(inp : int; load : bool);
returns out : int;
let out = if load then inp else (0 fby load);
tel;
```

Stream Semantics of Lustre

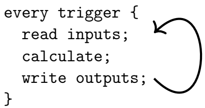
inp	3	4	1	5	2	6	1	1	...
load	F	F	T	F	T	T	F	T	...
out	0	0	1	1	2	6	6	1	...



```
every trigger {
  read inputs;
  calculate;
  write outputs;
}
```

Stream Semantics of Lustre

inp	3	4	1	5	2	6	1	1	...
load	F	F	T	F	T	T	F	T	...
out	0	0	1	1	2	6	6	1	...



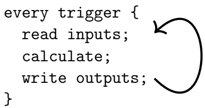
```
every trigger {
  read inputs;
  calculate;
  write outputs;
}
```

$$\frac{H(x) = vs}{G, H, bs \vdash x \Downarrow vs}$$

```
Inductive sem_exp:
| Svar: sem_var H x vs →
        sem_exp G H bs (Evar x ann) [vs]        [...]
```

Stream Semantics of Lustre

inp	3	4	1	5	2	6	1	1	...
load	F	F	T	F	T	T	F	T	...
out	0	0	1	1	2	6	6	1	...



```
every trigger {
  read inputs;
  calculate;
  write outputs;
}
```

$$\frac{H(x) = vs}{G, H, bs \vdash x \Downarrow vs}$$

$$\frac{G, H, bs \vdash \mathrm{es} \Downarrow H(\mathrm{xs})}{G, H, bs \vdash \mathrm{xs} = \mathrm{es}}$$

```
Inductive sem_exp:
| Svar: sem_var H x vs →
        sem_exp G H bs (Evar x ann) [vs]        [...]

with sem_equation:
| Seq: Forall2 (sem_exp H) es ss →
       Forall2 (sem_var H) xs (concat ss) →
       sem_equation G H bs (xs, es)
```

Stream Semantics of Lustre

inp	3	4	1	5	2	6	1	1	...
load	F	F	T	F	T	T	F	T	...
out	0	0	1	1	2	6	6	1	...

```
every trigger {
  read inputs;
  calculate;
  write outputs;
}
```

$$\frac{H(x) = vs}{G, H, bs \vdash x \Downarrow vs}$$

$$\frac{G, H, bs \vdash \mathrm{es} \Downarrow H(\mathrm{xs})}{G, H, bs \vdash \mathrm{xs} = \mathrm{es}}$$

```
Inductive sem_exp:
| Svar: sem_var H x vs →
        sem_exp G H bs (Evar x ann) [vs]        [...]

with sem_equation:
| Seq: Forall2 (sem_exp H) es ss →
       Forall2 (sem_var H) xs (concat ss) →
       sem_equation G H bs (xs, es)
```

$$\frac{\begin{array}{c}\mathtt{node}(G, f) \doteq n \qquad H(n.\mathbf{in}) = \mathrm{xs} \qquad H(n.\mathbf{out}) = \mathrm{ys} \\ \forall eq \in n.\mathbf{eqs},\ G, H, (\mathtt{base\text{-}of}\ \mathrm{xs}) \vdash eq\end{array}}{G \vdash f(\mathrm{xs}) \Downarrow \mathrm{ys}}$$

Coinductive semantics of the if operator

$$\begin{array}{c} \mathsf{ite}\ cs\ ts\ fs \doteq vs \\ \hline\hline \mathsf{ite}\ (\langle \mathrm{T} \rangle \cdot cs)\ (\langle t \rangle \cdot ts)\ (\langle f \rangle \cdot fs) \doteq \langle t \rangle \cdot vs \end{array} \qquad \begin{array}{c} \mathsf{ite}\ cs\ ts\ fs \doteq vs \\ \hline\hline \mathsf{ite}\ (\langle \mathrm{F} \rangle \cdot cs)\ (\langle t \rangle \cdot ts)\ (\langle f \rangle \cdot fs) \doteq \langle f \rangle \cdot vs \end{array}$$

Coinductive semantics of the if operator

$$\begin{array}{c} \text{ite}\ \mathit{cs}\ \mathit{ts}\ \mathit{fs} \doteq \mathit{vs} \\ \hline\hline \text{ite}\ (\langle \mathrm{T} \rangle \cdot \mathit{cs})\ (\langle t \rangle \cdot \mathit{ts})\ (\langle f \rangle \cdot \mathit{fs}) \doteq \langle t \rangle \cdot \mathit{vs} \end{array} \qquad \begin{array}{c} \text{ite}\ \mathit{cs}\ \mathit{ts}\ \mathit{fs} \doteq \mathit{vs} \\ \hline\hline \text{ite}\ (\langle \mathrm{F} \rangle \cdot \mathit{cs})\ (\langle t \rangle \cdot \mathit{ts})\ (\langle f \rangle \cdot \mathit{fs}) \doteq \langle f \rangle \cdot \mathit{vs} \end{array}$$

$$\begin{array}{c} \text{ite}\ \mathit{cs}\ \mathit{ts}\ \mathit{fs} \doteq \mathit{vs} \\ \hline\hline \text{ite}\ (\langle \rangle \cdot \mathit{cs})\ (\langle \rangle \cdot \mathit{ts})\ (\langle \rangle \cdot \mathit{fs}) \doteq \langle \rangle \cdot \mathit{vs} \end{array}$$

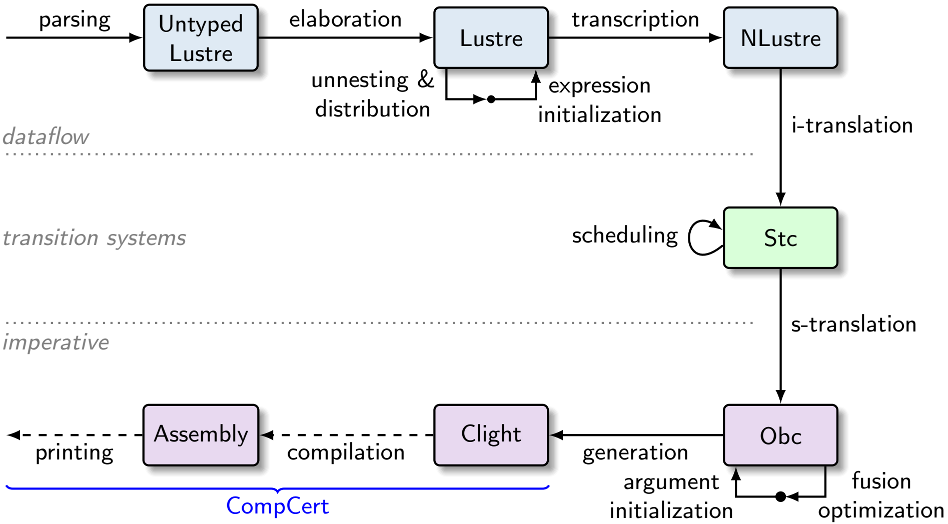
Coinductive semantics of the if operator

$$\begin{array}{c} \mathrm{ite}\ \mathit{cs}\ \mathit{ts}\ \mathit{fs} \doteq \mathit{vs} \\ \hline\hline \mathrm{ite}\ (\langle \mathrm{T} \rangle \cdot \mathit{cs})\ (\langle t \rangle \cdot \mathit{ts})\ (\langle f \rangle \cdot \mathit{fs}) \doteq \langle t \rangle \cdot \mathit{vs} \end{array} \qquad \begin{array}{c} \mathrm{ite}\ \mathit{cs}\ \mathit{ts}\ \mathit{fs} \doteq \mathit{vs} \\ \hline\hline \mathrm{ite}\ (\langle \mathrm{F} \rangle \cdot \mathit{cs})\ (\langle t \rangle \cdot \mathit{ts})\ (\langle f \rangle \cdot \mathit{fs}) \doteq \langle f \rangle \cdot \mathit{vs} \end{array}$$

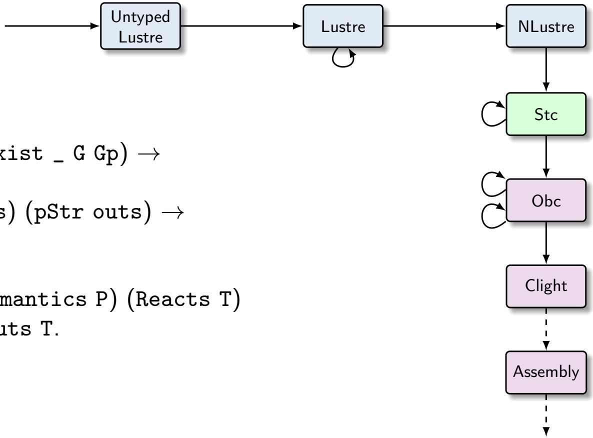
$$\begin{array}{c} \mathrm{ite}\ \mathit{cs}\ \mathit{ts}\ \mathit{fs} \doteq \mathit{vs} \\ \hline\hline \mathrm{ite}\ (\langle\rangle \cdot \mathit{cs})\ (\langle\rangle \cdot \mathit{ts})\ (\langle\rangle \cdot \mathit{fs}) \doteq \langle\rangle \cdot \mathit{vs} \end{array}$$

$$\frac{G, H, \mathit{bs} \vdash e \Downarrow [s] \qquad G, H, \mathit{bs} \vdash \boldsymbol{e_t} \Downarrow \mathrm{ts} \qquad G, H, \mathit{bs} \vdash \boldsymbol{e_f} \Downarrow \mathrm{fs} \qquad \mathrm{ite}\ s\ \mathrm{ts}\ \mathrm{fs} \doteq \mathrm{vs}}{G, H, \mathit{bs} \vdash \texttt{if}\ e\ \texttt{then}\ \boldsymbol{e_t}\ \texttt{else}\ \boldsymbol{e_f} \Downarrow \mathrm{vs}}$$

The Velus Compiler

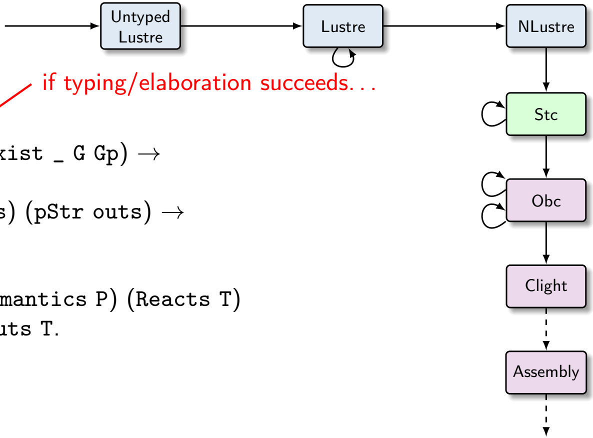


Main theorem



```
Theorem behavior_asm:
  ∀ D G Gp P main ins outs,
    elab_declarations D = OK (exist _ G Gp) →
    compile D main = OK P →
    Sem.sem_node G main (pStr ins) (pStr outs) →
    wt_ins G main ins →
    wc_ins G main ins →
    ∃ T, program_behaves (Asm.semantics P) (Reacts T)
          ∧ bisim_IO G main ins outs T.
```

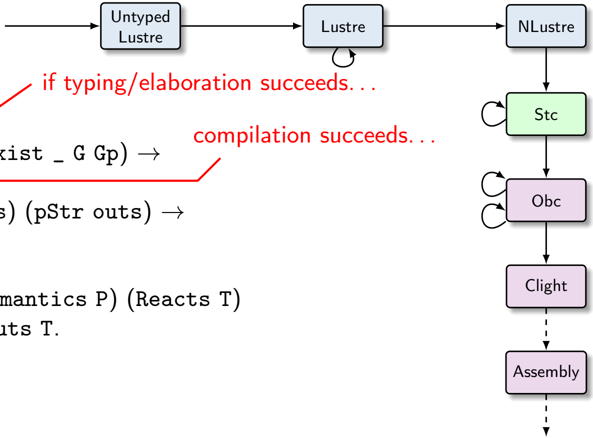
Main theorem



if typing/elaboration succeeds. . .

```
Theorem behavior_asm:
  ∀ D G Gp P main ins outs,
    elab_declarations D = OK (exist _ G Gp) →
    compile D main = OK P →
    Sem.sem_node G main (pStr ins) (pStr outs) →
    wt_ins G main ins →
    wc_ins G main ins →
    ∃ T, program_behaves (Asm.semantics P) (Reacts T)
         ∧ bisim_IO G main ins outs T.
```

Main theorem



if typing/elaboration succeeds. . .

compilation succeeds. . .

```
Theorem behavior_asm:
  ∀ D G Gp P main ins outs,
    elab_declarations D = OK (exist _ G Gp) →
    compile D main = OK P →
    Sem.sem_node G main (pStr ins) (pStr outs) →
    wt_ins G main ins →
    wc_ins G main ins →
    ∃ T, program_behaves (Asm.semantics P) (Reacts T)
         ∧ bisim_IO G main ins outs T.
```

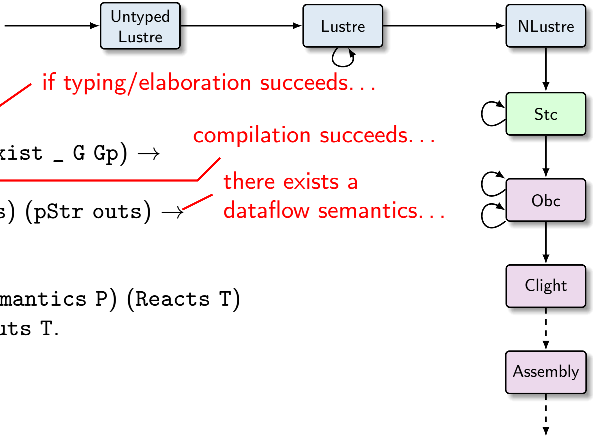
Main theorem

```
Theorem behavior_asm:
  ∀ D G Gp P main ins outs,
    elab_declarations D = OK (exist _ G Gp) →
    compile D main = OK P →
    Sem.sem_node G main (pStr ins) (pStr outs) →
    wt_ins G main ins →
    wc_ins G main ins →
    ∃ T, program_behaves (Asm.semantics P) (Reacts T)
          ∧ bisim_IO G main ins outs T.
```

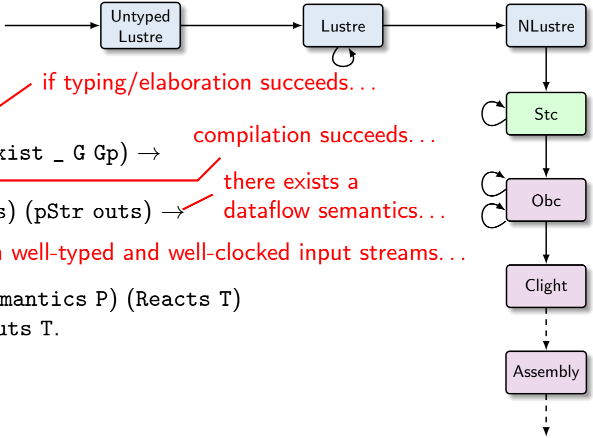
if typing/elaboration succeeds. . .

compilation succeeds. . .

there exists a
dataflow semantics. . .



Main theorem



```
Theorem behavior_asm:
  ∀ D G Gp P main ins outs,
    elab_declarations D = OK (exist _ G Gp) →
    compile D main = OK P →
    Sem.sem_node G main (pStr ins) (pStr outs) →
    wt_ins G main ins →
    wc_ins G main ins →
    ∃ T, program_behaves (Asm.semantics P) (Reacts T)
         ∧ bisim_IO G main ins outs T.
```

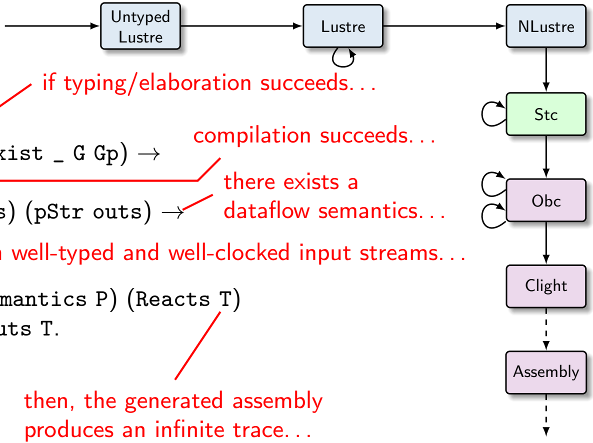
if typing/elaboration succeeds. . .

compilation succeeds. . .

there exists a dataflow semantics. . .

with well-typed and well-clocked input streams. . .

Main theorem



```
Theorem behavior_asm:
  ∀ D G Gp P main ins outs,
    elab_declarations D = OK (exist _ G Gp) →
    compile D main = OK P →
    Sem.sem_node G main (pStr ins) (pStr outs) →
    wt_ins G main ins →
    wc_ins G main ins →
    ∃ T, program_behaves (Asm.semantics P) (Reacts T)
          ∧ bisim_IO G main ins outs T.
```

if typing/elaboration succeeds. . .

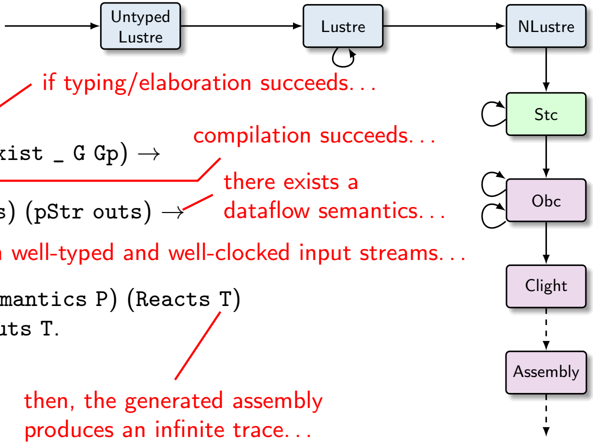
compilation succeeds. . .

there exists a dataflow semantics. . .

with well-typed and well-clocked input streams. . .

then, the generated assembly produces an infinite trace. . .

Main theorem



```
Theorem behavior_asm:
  ∀ D G Gp P main ins outs,
    elab_declarations D = OK (exist _ G Gp) →
    compile D main = OK P →
    Sem.sem_node G main (pStr ins) (pStr outs) →
    wt_ins G main ins →
    wc_ins G main ins →
    ∃ T, program_behaves (Asm.semantics P) (Reacts T)
         ∧ bisim_IO G main ins outs T.
```

if typing/elaboration succeeds. . .

compilation succeeds. . .

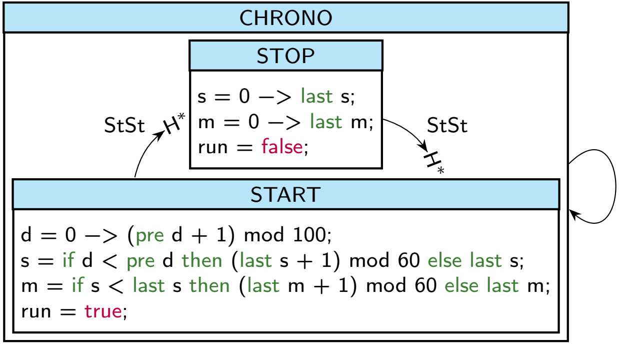
there exists a dataflow semantics. . .

with well-typed and well-clocked input streams. . .

then, the generated assembly produces an infinite trace. . .

. . . that corresponds to the dataflow model.

Extending Velus with control structures



CHRONO

STOP

```
s = 0 -> last s;
m = 0 -> last m;
run = false;
```

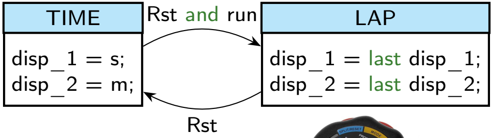
StSt

StSt

START

```
d = 0 -> (pre d + 1) mod 100;
s = if d < pre d then (last s + 1) mod 60 else last s;
m = if s < last s then (last m + 1) mod 60 else last m;
run = true;
```

Rst and not run



TIME

```
disp_1 = s;
disp_2 = m;
```

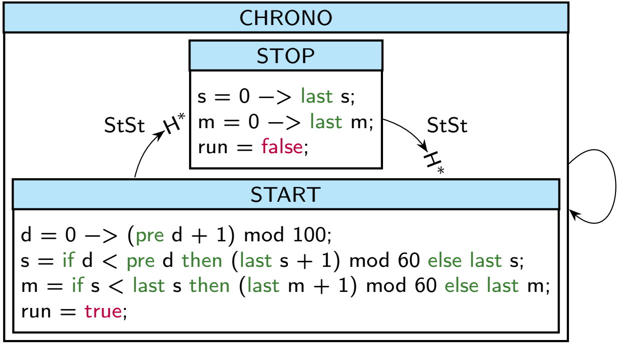
Rst and run

LAP

```
disp_1 = last disp_1;
disp_2 = last disp_2;
```

Rst

Extending Velus with control structures



CHRONO

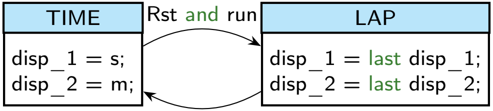
STOP

```
s = 0 −> last s;
m = 0 −> last m;
run = false;
```

START

```
d = 0 −> (pre d + 1) mod 100;
s = if d < pre d then (last s + 1) mod 60 else last s;
m = if s < last s then (last m + 1) mod 60 else last m;
run = true;
```

Rst and not run



TIME

```
disp_1 = s;
disp_2 = m;
```

LAP

```
disp_1 = last disp_1;
disp_2 = last disp_2;
```

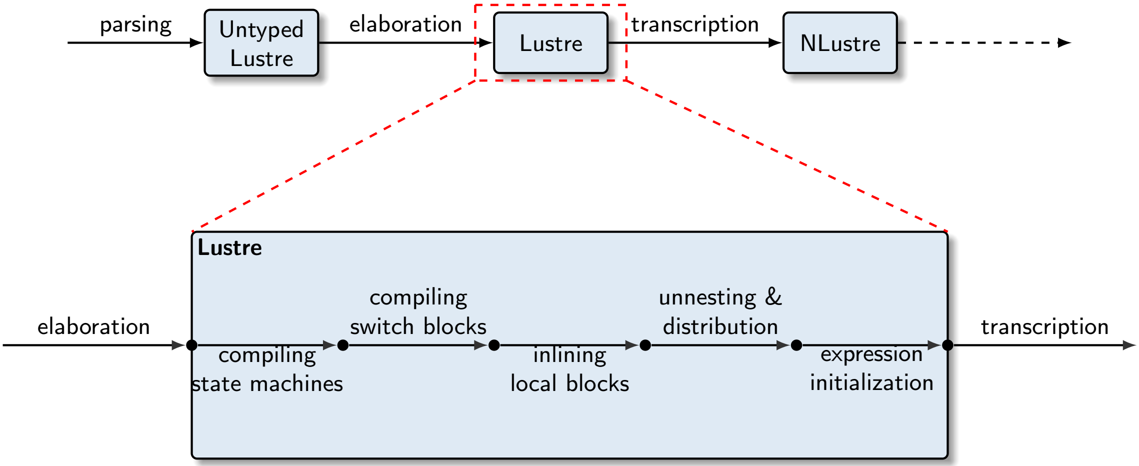
Rst and run

Rst

Intermediate structures used to compile state machines:

- Switch blocks
- Reset blocks
- Local blocks (useful for compiling other constructs)

Extending the Velus compiler



Expressing block semantics

How to express the semantics of blocks ?

- Solution 1 : blocks are functions; $G \vdash B(\mathsf{xs}) \Downarrow \mathsf{ys}$
  - inputs are the free variables of the block
  - outputs are the variables defined by the block

  Pros:

  - Definition of node semantics is direct

$$\frac{\texttt{node}(G, f) \doteq B \qquad G \vdash B(\mathsf{xs}) \Downarrow \mathsf{ys}}{G \vdash f(\mathsf{xs}) \Downarrow \mathsf{ys}}$$

  - Input / Output of blocks can be manipulated

Expressing block semantics

How to express the semantics of blocks ?

- Solution 1 : blocks are functions; $G \vdash B(\text{xs}) \Downarrow \text{ys}$
  - » inputs are the free variables of the block
  - » outputs are the variables defined by the block

  Pros:

  - » Definition of node semantics is direct

$$\frac{\texttt{node}(G, f) \doteq B \qquad G \vdash B(\text{xs}) \Downarrow \text{ys}}{G \vdash f(\text{xs}) \Downarrow \text{ys}}$$

  - » Input / Output of blocks can be manipulated

  Cons:

  - » Free / Defined variables have to be encoded in the semantics
  - » Composition cumbersome : inputs and outputs have to be constrained in both the inside and outside history of the block

Expressing block semantics

How to express the semantics of blocks ?

- Solution 1 : blocks are functions; $G \vdash B(\mathrm{xs}) \Downarrow \mathrm{ys}$
  - inputs are the free variables of the block
  - outputs are the variables defined by the block

  Pros:

  - Definition of node semantics is direct

$$\frac{\mathtt{node}(G, f) \doteq B \qquad G \vdash B(\mathrm{xs}) \Downarrow \mathrm{ys}}{G \vdash f(\mathrm{xs}) \Downarrow \mathrm{ys}}$$

  - Input / Output of blocks can be manipulated

  Cons:

  - Free / Defined variables have to be encoded in the semantics
  - Composition cumbersome : inputs and outputs have to be constrained in both the inside and outside history of the block
- Solution 2 : blocks are constraints; $G, H, bs \vdash B$

Reset - Example and Intuition

```
node expect(a : bool)
returns (o : bool)
let
  o = a or (false fby o);
tel

node abro(a, b, r : bool)
returns (o : bool)
let
  reset
    o = expect(a) and expect(b);
  every r
tel
```

a	F	F	T	F	T	F	...
o	F	F	T	T	T	T	...

Reset - Example and Intuition

```
node expect(a : bool)
returns (o : bool)
let
  o = a or (false fby o);
tel
```

a	F	F	T	F	T	F	...
o	F	F	T	T	T	T	...

```
node abro(a, b, r : bool)
returns (o : bool)
let
  reset
    o = expect(a) and expect(b);
  every r
tel
```

a	F	T	F	F	F	...
b	F	F	F	T	F	...
r	F	F	F	F	F	...
o	F	F	F	T	T	...

Reset - Example and Intuition

```
node expect(a : bool)
returns (o : bool)
let
  o = a or (false fby o);
tel
```

a	F	F	T	F	T	F	...
o	F	F	T	T	T	T	...

```
node abro(a, b, r : bool)
returns (o : bool)
let
  reset
    o = expect(a) and expect(b);
  every r
tel
```

a						F	F				...
b						T	F				...
r						T	F				...
o						F	F				...

Reset - Example and Intuition

```
node expect(a : bool)
returns (o : bool)
let
  o = a or (false fby o);
tel
```

a	F	F	T	F	T	F	...
o	F	F	T	T	T	T	...

```
node abro(a, b, r : bool)
returns (o : bool)
let
  reset
    o = expect(a) and expect(b);
  every r
tel
```

a								F	F	T	...
b								F	F	T	...
r								T	F	F	...
o								F	F	T	...

Reset - Example and Intuition

```
node expect(a : bool)
returns (o : bool)
let
  o = a or (false fby o);
tel
```

a	F	F	T	F	T	F	...
o	F	F	T	T	T	T	...

```
node abro(a, b, r : bool)
returns (o : bool)
let
  reset
    o = expect(a) and expect(b);
  every r
tel
```

a	F	T	F	F	F	F	F	F	F	T	...
b	F	F	F	T	F	T	F	F	F	T	...
r	F	F	F	F	F	T	F	T	F	F	...
o	F	F	F	T	T	F	F	F	F	T	...

Reset - stream semantics

$$
\begin{aligned}
\mathsf{mask}^{0}_{\mathrm{T}\cdot rs}(x \cdot xs) &\equiv \text{always-absent}\\
\mathsf{mask}^{0}_{\mathrm{F}\cdot rs}(x \cdot xs) &\equiv x \cdot \mathit{mask}^{0}_{rs}\ xs\\
\mathsf{mask}^{1}_{\mathrm{T}\cdot rs}(x \cdot xs) &\equiv x \cdot \mathit{mask}^{0}_{rs}\ xs\\
\mathsf{mask}^{k+1}_{\mathrm{T}\cdot rs}(x \cdot xs) &\equiv \langle\rangle \cdot \mathit{mask}^{k}_{rs}\ xs\\
\mathsf{mask}^{k+1}_{\mathrm{F}\cdot rs}(x \cdot xs) &\equiv \langle\rangle \cdot \mathit{mask}^{k+1}_{rs}\ xs
\end{aligned}
$$

Reset - stream semantics

$$
\begin{aligned}
\mathrm{mask}^{0}_{\mathrm{T}\cdot rs}(x \cdot xs) &\equiv \text{always-absent}\\
\mathrm{mask}^{0}_{\mathrm{F}\cdot rs}(x \cdot xs) &\equiv x \cdot \mathit{mask}^{0}_{rs}\ xs\\
\mathrm{mask}^{1}_{\mathrm{T}\cdot rs}(x \cdot xs) &\equiv x \cdot \mathit{mask}^{0}_{rs}\ xs\\
\mathrm{mask}^{k+1}_{\mathrm{T}\cdot rs}(x \cdot xs) &\equiv \langle\rangle \cdot \mathit{mask}^{k}_{rs}\ xs\\
\mathrm{mask}^{k+1}_{\mathrm{F}\cdot rs}(x \cdot xs) &\equiv \langle\rangle \cdot \mathit{mask}^{k+1}_{rs}\ xs
\end{aligned}
$$

rs	F	T	F	T	F	F	T	F	...
xs	1	2	3	4	5	6	7	8	...
$\mathrm{mask}^{2}_{rs}\ xs$	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	4	5	6	$\langle\rangle$	$\langle\rangle$	...

Reset - stream semantics

$$
\begin{aligned}
\mathrm{mask}^{0}_{\mathrm{T}\cdot rs}(x \cdot xs) &\equiv \text{always-absent}\\
\mathrm{mask}^{0}_{\mathrm{F}\cdot rs}(x \cdot xs) &\equiv x \cdot \mathit{mask}^{0}_{rs}\ xs\\
\mathrm{mask}^{1}_{\mathrm{T}\cdot rs}(x \cdot xs) &\equiv x \cdot \mathit{mask}^{0}_{rs}\ xs\\
\mathrm{mask}^{k+1}_{\mathrm{T}\cdot rs}(x \cdot xs) &\equiv \langle\rangle \cdot \mathit{mask}^{k}_{rs}\ xs\\
\mathrm{mask}^{k+1}_{\mathrm{F}\cdot rs}(x \cdot xs) &\equiv \langle\rangle \cdot \mathit{mask}^{k+1}_{rs}\ xs
\end{aligned}
$$

rs	F	T	F	T	F	F	T	F	...
xs	1	2	3	4	5	6	7	8	...
$\mathrm{mask}^{2}_{rs}\ xs$	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	4	5	6	$\langle\rangle$	$\langle\rangle$	...

$$
\frac{G, H, bs \vdash e_r \Downarrow [s] \qquad \text{bools-of } s \doteq r \qquad G, H, bs \vdash \boldsymbol{es} \Downarrow \mathrm{xs} \qquad \forall k,\ \ G \vdash f(\mathrm{mask}^{k}_{r}\ \mathrm{xs}) \Downarrow \mathrm{mask}^{k}_{r}\ \mathrm{ys}}{G, H, bs \vdash (\texttt{restart}\ f\ \texttt{every}\ e_r)(\boldsymbol{es}) \Downarrow \mathrm{ys}}
$$

Reset - stream semantics

$$\mathrm{mask}^{0}_{\mathrm{T}\cdot rs}(x \cdot xs) \equiv \text{always-absent}$$

$$\mathrm{mask}^{0}_{\mathrm{F}\cdot rs}(x \cdot xs) \equiv x \cdot \mathit{mask}^{0}_{rs}\ xs$$

$$\mathrm{mask}^{1}_{\mathrm{T}\cdot rs}(x \cdot xs) \equiv x \cdot \mathit{mask}^{0}_{rs}\ xs$$

$$\mathrm{mask}^{k+1}_{\mathrm{T}\cdot rs}(x \cdot xs) \equiv \langle\rangle \cdot \mathit{mask}^{k}_{rs}\ xs$$

$$\mathrm{mask}^{k+1}_{\mathrm{F}\cdot rs}(x \cdot xs) \equiv \langle\rangle \cdot \mathit{mask}^{k+1}_{rs}\ xs$$

rs	F	T	F	T	F	F	T	F	...
xs	1	2	3	4	5	6	7	8	...
$\mathrm{mask}^{2}_{rs}\ xs$	$\langle\rangle$	$\langle\rangle$	$\langle\rangle$	4	5	6	$\langle\rangle$	$\langle\rangle$	...

$$\frac{\begin{array}{c} G, H, bs \vdash e_r \Downarrow [s] \qquad \text{bools-of } s \doteq r \\ G, H, bs \vdash \boldsymbol{es} \Downarrow \mathrm{xs} \\ \forall k,\ \ G \vdash f(\mathrm{mask}^{k}_{r}\ \mathrm{xs}) \Downarrow \mathrm{mask}^{k}_{r}\ \mathrm{ys} \end{array}}{G, H, bs \vdash (\texttt{restart}\ f\ \texttt{every}\ e_r)(\boldsymbol{es}) \Downarrow \mathrm{ys}}$$

$$\frac{\begin{array}{c} G, H, bs \vdash e_r \Downarrow [s] \qquad \text{bools-of } s \doteq r \\ \forall k,\ \ G, \mathrm{mask}^{k}_{r}(H, bs) \vdash blks \end{array}}{G, H, bs \vdash \texttt{reset}\ blks\ \texttt{every}\ e_r}$$

Lustre fby operator semantics

$$\begin{array}{c} \mathrm{fby}\ \mathit{xs}\ \mathit{ys} \doteq \mathit{vs} \\ \hline\hline \mathrm{fby}\ (\langle\rangle \cdot \mathit{xs})\ (\langle\rangle \cdot \mathit{ys}) \doteq \langle\rangle \cdot \mathit{vs} \end{array} \qquad \begin{array}{c} \mathrm{fby}_1\ y\ \mathit{xs}\ \mathit{ys} \doteq \mathit{vs} \\ \hline\hline \mathrm{fby}\ (\langle x\rangle \cdot \mathit{xs})\ (\langle y\rangle \cdot \mathit{ys}) \doteq \langle x\rangle \cdot \mathit{vs} \end{array}$$

$$\begin{array}{c} \mathrm{fby}_1\ v\ \mathit{xs}\ \mathit{ys} \doteq \mathit{vs} \\ \hline\hline \mathrm{fby}_1\ v\ (\langle\rangle \cdot \mathit{xs})\ (\langle\rangle \cdot \mathit{ys}) \doteq \langle\rangle \cdot \mathit{vs} \end{array} \qquad \begin{array}{c} \mathrm{fby}_1\ y\ \mathit{xs}\ \mathit{ys} \doteq \mathit{vs} \\ \hline\hline \mathrm{fby}_1\ v\ (\langle x\rangle \cdot \mathit{xs})\ (\langle y\rangle \cdot \mathit{ys}) \doteq \langle v\rangle \cdot \mathit{vs} \end{array}$$

$$\frac{G, H \vdash \boldsymbol{e}_0 \Downarrow \mathrm{xs} \qquad G, H \vdash \boldsymbol{e}_1 \Downarrow \mathrm{ys} \qquad \mathrm{fby\ xs\ ys} \doteq \mathrm{vs}}{G, H \vdash \boldsymbol{e}_0\ \texttt{fby}\ \boldsymbol{e}_1 \Downarrow \mathrm{vs}}$$

Lustre fby operator semantics - With reset signal

$$\begin{array}{c} \text{fby}\ v\ \mathit{xs}\ \mathit{ys}\ \mathit{rs} \doteq \mathit{vs} \\ \hline\hline \text{fby}\ v\ (\langle\rangle \cdot \mathit{xs})\ (\langle\rangle \cdot \mathit{ys})\ (\mathrm{F} \cdot \mathit{rs}) \doteq \langle\rangle \cdot \mathit{vs} \end{array}$$

$$\begin{array}{c} \text{fby}\ \langle y\rangle\ \mathit{xs}\ \mathit{ys}\ \mathit{rs} \doteq \mathit{vs} \\ \hline\hline \text{fby}\ \langle\rangle\ (\langle x\rangle \cdot \mathit{xs})\ (\langle y\rangle \cdot \mathit{ys})\ (\mathrm{F} \cdot \mathit{rs}) \doteq \langle x\rangle \cdot \mathit{vs} \end{array} \qquad \begin{array}{c} \text{fby}\ \langle y\rangle\ \mathit{xs}\ \mathit{ys}\ \mathit{rs} \doteq \mathit{vs} \\ \hline\hline \text{fby}\ \langle v\rangle\ (\langle x\rangle \cdot \mathit{xs})\ (\langle y\rangle \cdot \mathit{ys})\ (\mathrm{F} \cdot \mathit{rs}) \doteq \langle v\rangle \cdot \mathit{vs} \end{array}$$

$$\begin{array}{c} \text{fby}\ \langle\rangle\ \mathit{xs}\ \mathit{ys}\ \mathit{rs} \doteq \mathit{vs} \\ \hline\hline \text{fby}\ v\ (\langle\rangle \cdot \mathit{xs})\ (\langle\rangle \cdot \mathit{ys})\ (\mathrm{T} \cdot \mathit{rs}) \doteq \langle\rangle \cdot \mathit{vs} \end{array} \qquad \begin{array}{c} \text{fby}\ \langle y\rangle\ \mathit{xs}\ \mathit{ys}\ \mathit{rs} \doteq \mathit{vs} \\ \hline\hline \text{fby}\ (\langle x\rangle \cdot \mathit{xs})\ (\langle y\rangle \cdot \mathit{ys})\ (\mathrm{T} \cdot \mathit{rs}) \doteq \langle x\rangle \cdot \mathit{vs} \end{array}$$

$$\dfrac{\begin{array}{c} G, H \vdash \boldsymbol{e}_0 \Downarrow \mathrm{xs} \qquad G, H \vdash \boldsymbol{e}_1 \Downarrow \mathrm{ys} \qquad G, H \vdash e_r \Downarrow [s] \qquad \text{bools-of}\ s \doteq r \\ \text{fby}\ \langle\rangle\ \mathrm{xs}\ \mathrm{ys}\ r\ \doteq \mathrm{vs} \end{array}}{G, H \vdash (\texttt{reset}\ \boldsymbol{e}_0\ \texttt{fby}\ \boldsymbol{e}_1\ \texttt{every}\ e_r) \Downarrow \mathrm{vs}}$$

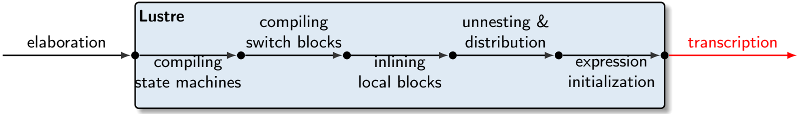
NLustre fby operator semantics

```
CoFixpoint sfby v xs :=
    match str with
    | ⟨v'⟩ · xs' ⇒ ⟨v⟩ · (sfby v' xs')
    | ⟨⟩ · xs' ⇒ ⟨⟩ · (sfby v xs')
    end.
```

```
CoFixpoint reset1 v0 xs rs doreset :=
    match xs, rs, doreset with
    | ⟨⟩ · xs, false · rs, false ⇒ ⟨⟩ · (reset1 v0 xs rs false)
    | ⟨⟩ · xs, true · rs, _
    | ⟨⟩ · xs, _ · rs, true ⇒ ⟨⟩ · (reset1 v0 xs rs true)
    | ⟨x⟩ · xs, false · rs, false ⇒ ⟨x⟩ · (reset1 v0 xs rs false)
    | ⟨x⟩ · xs, true · rs, _
    | ⟨x⟩ · xs, _ · rs, true ⇒ ⟨v0⟩ · (reset1 v0 xs rs false)
    end.
Definition reset v0 xs rs := reset1 v0 xs rs false.
```

$$\frac{\begin{array}{c} G, H, bs \vdash e_1 \Downarrow xs \qquad G, H, bs \vdash e_r \Downarrow [s] \qquad \text{bools-of } s \doteq r \\ H(x) = \text{reset } c_0 \; (\text{sfby } c0 \; xs) \; r \end{array}}{G, H, bs \vdash x = (\texttt{reset } c_0 \texttt{ fby } e_1 \texttt{ every } e_r)}$$

Reset - Compilation



Lustre

elaboration

compiling state machines

compiling switch blocks

inlining local blocks

unnesting & distribution

expression initialization

transcription

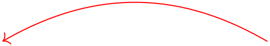
```
node abro(a, b, r : bool) returns (o : bool)
var ea, eb, peb : bool;
let
  reset
    ea = expect(a);
    peb = false fby eb;
    eb = b or peb;
    o = ea and eb;
  every r
tel
```

```
node abro (a, b, r : bool) returns (o : bool)
var ea, eb, peb : bool;
let
  ea = (restart expect every r)(a);
  peb = reset (false fby eb) every r;
  eb = b or peb;
  o = ea and eb;
tel
```

Local Blocks - Shadowing rules

```
node f(b : bool; x : int when b) returns (z : int)
let
    var b : bool;
    let
        z = merge b (true -> x) (false -> 0);
        b = true fby false;
    tel
tel
```

Local Blocks - Shadowing rules

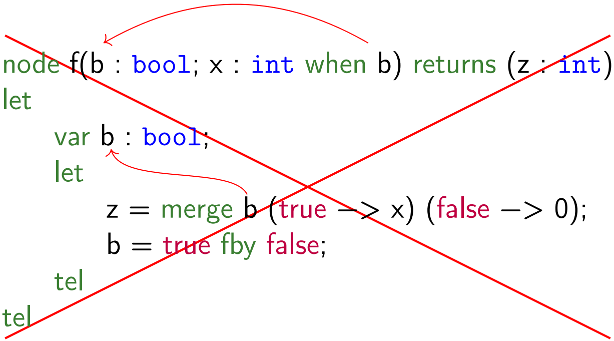


```
node f(b : bool; x : int when b) returns (z : int)
let
    var b : bool;
    let
        z = merge b (true -> x) (false -> 0);
        b = true fby false;
    tel
tel
```

Local Blocks - Shadowing rules

```
node f(b : bool; x : int when b) returns (z : int)
let
    var b : bool;
    let
        z = merge b (true −> x) (false −> 0);
        b = true fby false;
    tel
tel
```

Local Blocks - Shadowing rules



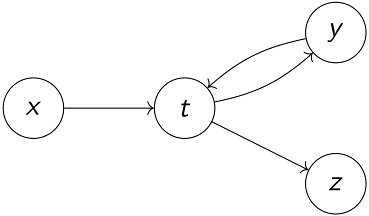
```
node f(b : bool; x : int when b) returns (z : int)
let
    var b : bool;
    let
        z = merge b (true -> x) (false -> 0);
        b = true fby false;
    tel
tel
```

$$\frac{\begin{array}{c}\text{NoDup}\ xs \qquad \forall x, x \in xs \Rightarrow x \notin \Gamma \\ (\Gamma \cup xs) \vdash_{NDL} B\end{array}}{\Gamma \vdash_{NDL} \texttt{var}\ xs\ \texttt{let}\ B\ \texttt{tel}}$$

$$\frac{\begin{array}{c}\text{NoDup}\ (n.\mathbf{in} \cup n.\mathbf{out}) \\ (n.\mathbf{in} \cup n.\mathbf{out}) \vdash_{NDL} n.\mathbf{blk}\end{array}}{\vdash_{NDL} n}$$

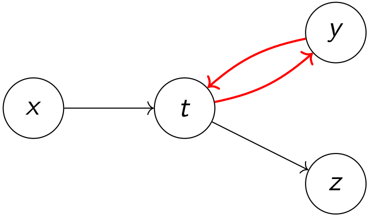
Local Blocks - Causality analysis

```
node f(x : int) returns (z : bool)
var y : int;
let
  var t : int;
  let t = x fby (t + 1);
      y = t;
  tel;
  var t : int;
  let t = y + 1;
      z = t > 0;
  tel
tel
```



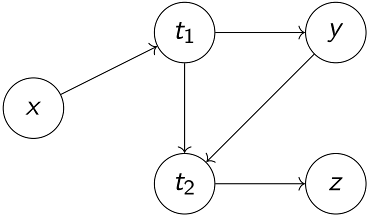
Local Blocks - Causality analysis

```
node f(x : int) returns (z : bool)
var y : int;
let
  var t : int;
  let t = x fby (t + 1);
      y = t;
  tel;
  var t : int;
  let t = y + 1;
      z = t > 0;
  tel
tel
```



Local Blocks - Causality analysis

```
node f(x(x1) : int) returns (z(z1) : bool)
var y(y1) : int;
let
  var t(t1) : int;
  let t = x fby (t + 1);
      y = t;
  tel;
  var t(t2) : int;
  let t = y + 1;
      z = t > 0;
  tel
tel
```



Local Blocks - Semantic rules

$$\frac{G, H', bs \vdash B \qquad {\color{red} R?\ H\ H'}}{G, H, bs \vdash \texttt{var}\ xs\ \texttt{let}\ B\ \texttt{tel}}$$

Local Blocks - Semantic rules

$$\frac{G, H', bs \vdash B \qquad {\color{red} H \subseteq H'}}{G, H, bs \vdash \texttt{var}\ xs\ \texttt{let}\ B\ \texttt{tel}}$$

Local Blocks - Semantic rules

$$\frac{G, H', bs \vdash B \qquad \forall x\ vs, H(x) = vs \Rightarrow H'(x) = vs}{G, H, bs \vdash \texttt{var}\ xs\ \texttt{let}\ B\ \texttt{tel}}$$

Local Blocks - Semantic rules

$$\frac{G, H', bs \vdash B \qquad \forall x\ vs, H(x) = vs \Rightarrow H'(x) = vs}{G, H, bs \vdash \texttt{var}\ xs\ \texttt{let}\ B\ \texttt{tel}}$$

```
node f(i : int) returns (o : int)
var z : int;
let
    var x : int;
    let
        x = 1;
        z = x;
    tel
    var t : int;
    let
        t = z;
        o = t;
    tel
tel
```

$z \notin H$

$H_1(x) = 1 \cdot 1 \cdot 1 \cdot \ldots$

$H_1(z) = 1 \cdot 1 \cdot 1 \cdot \ldots$

$H_2(z) = ?$

Local Blocks - Semantic rules

$$\frac{G, H', bs \vdash B \qquad \forall x\ vs, H'(x) = vs \Rightarrow H(x) = vs}{G, H, bs \vdash \texttt{var}\ xs\ \texttt{let}\ B\ \texttt{tel}}$$

```
node f(i : int) returns (o : int)
var z : int;
let
    var x : int;
    let
        x = 1;
        z = x;
    tel
    var t : int;
    let
        t = z;
        o = t;
    tel
tel
```

$H(z) = 1 \cdot 1 \cdot 1 \cdot \ldots$

$H_1(x) = 1 \cdot 1 \cdot 1 \cdot \ldots$
$H_1(z) = 1 \cdot 1 \cdot 1 \cdot \ldots$

$z \in H_2$, therefore
$H_2(z) = 1 \cdot 1 \cdot 1 \cdot \ldots$

Local Blocks - Semantic rules

$$\frac{\begin{array}{c} G, H', bs \vdash B \\ \forall x\ vs,\ H'(x) = vs \Rightarrow H(x) = vs \end{array}}{G, H, bs \vdash \texttt{var}\ xs\ \texttt{let}\ B\ \texttt{tel}}$$

$H(x) = ?$

```
node f(i : int) returns (o : int)
var z : int;
let
    var x : int;
    let
        x = 1;
        z = x;
    tel
    var x : int;
    let
        x = 2;
        o = x;
    tel
tel
```

$H_1(x) = 1 \cdot 1 \cdot 1 \cdot \ldots$

$H_1(z) = 1 \cdot 1 \cdot 1 \cdot \ldots$

$H_2(x) = 2 \cdot 2 \cdot 2 \cdot \ldots$

Local Blocks - Semantic rules

$$\frac{G, H', bs \vdash B \qquad \forall x\ vs, x \notin xs \Rightarrow H'(x) = vs \Rightarrow H(x) = vs}{G, H, bs \vdash \texttt{var}\ xs\ \texttt{let}\ B\ \texttt{tel}}$$

```
node f(i : int) returns (o : int)
var z : int;
let
    var x : int;
    let
        x = 1;
        z = x;
    tel
    var x : int;
    let
        x = 2;
        o = x;
    tel
tel
```

$x \notin H$

$H_1(x) = 1 \cdot 1 \cdot 1 \cdot \ldots$

$H_1(z) = 1 \cdot 1 \cdot 1 \cdot \ldots$

$H_2(x) = 2 \cdot 2 \cdot 2 \cdot \ldots$

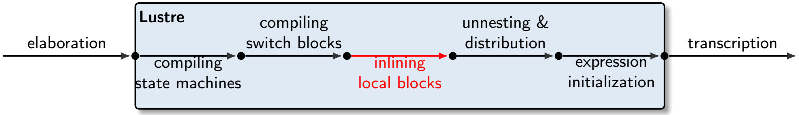
Local Blocks - Semantic rules

$$\frac{G, H', bs \vdash B \qquad \forall x\ vs, x \notin xs \Rightarrow H'(x) = vs \Rightarrow H(x) = vs}{G, H, bs \vdash \texttt{var}\ xs\ \texttt{let}\ B\ \texttt{tel}}$$

$$\frac{\texttt{node}(G, f) \doteq n \qquad H(n.\mathbf{in}) = \mathrm{xs} \qquad H(n.\mathbf{out}) = \mathrm{ys} \qquad G, H, (\texttt{base-of}\ \mathrm{xs}) \vdash n.\mathbf{blk}}{G \vdash f(\mathrm{xs}) \Downarrow \mathrm{ys}}$$

```
node f(i : int) returns (o : int)
var z : int;
let
    var x : int;
    let
        x = 1;
        z = x;
    tel
    var x : int;
    let
        x = 2;
        o = x;
    tel
tel
```

Local Blocks - Compilation



```
node f(x : int) returns (z : bool)
var y : int;
let
  var t : int;
  let t = x fby (t + 1);
      y = t;
  tel;
  var t : int;
  let t = y + 1;
      z = t > 0;
  tel
tel
```

```
node f (x : int) returns (z : bool)
var y : int; local$t$2 : int; local$t$1 : int;
let
  local$t$1 = x fby (local$t$1 + 1);
  y = local$t$1;
  local$t$2 = y + 1;
  z = local$t$2 > 0
tel
```

Switch - Stateful example

```
type modes = Up | Down

node two(m : modes) returns (o : int)
let
    switch m
    | Up -> o = 1 fby (o + 1)
    | Down -> o = 0
    end
tel
```

base	. . .
m	. . .
o	. . .

Switch - Stateful example

```
type modes = Up | Down

node two(m : modes) returns (o : int)
let
    switch m
    | Up -> o = 1 fby (o + 1)
    | Down -> o = 0
    end
tel
```

x	U	U	U	U	U	U	...
y	1	2	3	4	5	6	...

base	T	T	T			T	T			T	...
m	U	U	U			U	U			U	...
o	1	2	3			4	5			6	...

Switch - Stateful example

```
type modes = Up | Down

node two(m : modes) returns (o : int)
let
    switch m
    | Up -> o = 1 fby (o + 1)
    | Down -> o = 0
    end
tel
```

x	U	U	U	U	U	U	. . .
y	1	2	3	4	5	6	. . .

m	D	D	D	D	. . .
o	0	0	0	0	. . .

base			T	T			T	T	. . .
m			D	D			D	D	. . .
o			0	0			0	0	. . .

Switch - Stateful example

```
type modes = Up | Down

node two(m : modes) returns (o : int)
let
    switch m
    | Up -> o = 1 fby (o + 1)
    | Down -> o = 0
    end
tel
```

x	U	U	U	U	U	U	. . .
y	1	2	3	4	5	6	. . .

m	D	D	D	D	. . .
o	0	0	0	0	. . .

base	T	T	T	T	T	T	T	T	T	T	. . .
m	U	U	U	D	D	U	U	D	D	U	. . .
o	1	2	3	0	0	4	5	0	0	6	. . .

Switch - Semantic rules

$$\begin{aligned}
\mathsf{filter}^{C}_{\langle C \rangle \cdot cs}\ (v \cdot vs) &\equiv v \cdot \mathsf{filter}^{C}_{cs}\ vs \\
\mathsf{filter}^{C}_{\langle \rangle \cdot cs}\ (v \cdot vs) &\equiv \langle \rangle \cdot \mathsf{filter}^{C}_{cs}\ vs \\
\mathsf{filter}^{C}_{\langle C' \rangle \cdot cs}\ (v \cdot vs) &\equiv \langle \rangle \cdot \mathsf{filter}^{C}_{cs}\ vs\ \textbf{if}\ C' \neq C
\end{aligned}$$

$$\frac{G, H, bs \vdash e \Downarrow [cs] \qquad G, \mathsf{filter}^{C_i}_{cs}(H, bs) \vdash B_i}{G, H, bs \vdash \texttt{switch}\ e\ (C_1 \rightarrow B_1) \ldots (C_n \rightarrow B_n)}$$

Switch - Semantic rules

$$\text{filter}^{C}_{\langle C\rangle\cdot cs}\ (v \cdot vs) \equiv v \cdot \text{filter}^{C}_{cs}\ vs$$

$$\text{filter}^{C}_{\langle\rangle\cdot cs}\ (v \cdot vs) \equiv \langle\rangle \cdot \text{filter}^{C}_{cs}\ vs$$

$$\text{filter}^{C}_{\langle C'\rangle\cdot cs}\ (v \cdot vs) \equiv \langle\rangle \cdot \text{filter}^{C}_{cs}\ vs \ \textbf{if}\ C' \neq C$$

```
node f(b : bool; c : bool when b)
returns (z : int when b)
let switch c
      | true -> z = 1
      | false -> z = 0
      end
tel
```

b	T	T	F	T	...
c	T	F	⟨⟩	F	...
z	1	0	?	0	...

$$\frac{G, H, bs \vdash e \Downarrow [cs] \qquad G, \text{filter}^{C_i}_{cs}(H, bs) \vdash B_i}{G, H, bs \vdash \texttt{switch}\ e\ (C_1 \to B_1) \ldots (C_n \to B_n)}$$

Switch - Semantic rules

$$\text{filter}^{C}_{\langle C\rangle\cdot cs}\ (v\cdot vs)\equiv v\cdot\text{filter}^{C}_{cs}\ vs$$

$$\text{filter}^{C}_{\langle\rangle\cdot cs}\ (v\cdot vs)\equiv \langle\rangle\cdot\text{filter}^{C}_{cs}\ vs$$

$$\text{filter}^{C}_{\langle C'\rangle\cdot cs}\ (v\cdot vs)\equiv \langle\rangle\cdot\text{filter}^{C}_{cs}\ vs\ \textbf{if}\ C'\neq C$$

$$\frac{\text{slower}\ xs\ bs}{\text{slower}\ (\langle\rangle\cdot xs)\ (\mathrm{F}\cdot bs)}\qquad\frac{\text{slower}\ xs\ bs}{\text{slower}\ (v\cdot xs)\ (\mathrm{T}\cdot bs)}$$

```
node f(b : bool; c : bool when b)
returns (z : int when b)
let switch c
      | true -> z = 1
      | false -> z = 0
      end
tel
```

b	T	T	F	T	...
c	T	F	⟨⟩	F	...
z	1	0	⟨⟩	0	...

$$\frac{G,H,bs\vdash e\Downarrow[cs]\qquad G,\text{filter}^{C_i}_{cs}(H,bs)\vdash B_i}{G,H,bs\vdash\texttt{switch}\ e\ (C_1\rightarrow B_1)\ldots(C_n\rightarrow B_n)}$$

Switch - Semantic rules

$$\text{filter}^{C}_{\langle C\rangle\cdot cs}\ (v \cdot vs) \equiv v \cdot \text{filter}^{C}_{cs}\ vs$$

$$\text{filter}^{C}_{\langle\rangle\cdot cs}\ (v \cdot vs) \equiv \langle\rangle \cdot \text{filter}^{C}_{cs}\ vs$$

$$\text{filter}^{C}_{\langle C'\rangle\cdot cs}\ (v \cdot vs) \equiv \langle\rangle \cdot \text{filter}^{C}_{cs}\ vs \ \textbf{if}\ C' \neq C$$

$$\frac{\text{slower}\ xs\ bs}{\text{slower}\ (\langle\rangle \cdot xs)\ (\mathrm{F} \cdot bs)} \qquad \frac{\text{slower}\ xs\ bs}{\text{slower}\ (v \cdot xs)\ (\mathrm{T} \cdot bs)}$$

```
node f(b : bool; c : bool when b)
returns (z : int when b)
let switch c
      | true -> z = 1
      | false -> z = 0
      end
tel
```

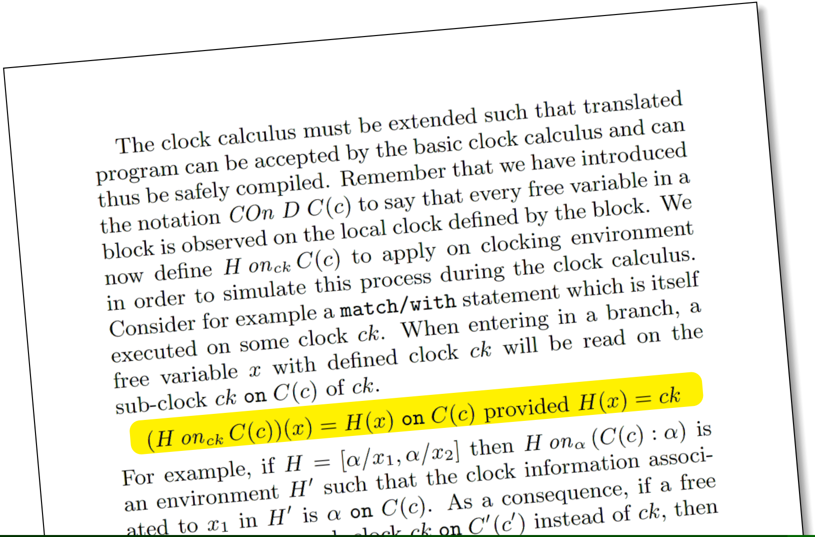
b	T	T	F	T	...
c	T	F	$\langle\rangle$	F	...
z	1	0		0	...

$$\frac{G, H, bs \vdash e \Downarrow [cs] \qquad G, \text{filter}^{C_i}_{cs}(H, bs) \vdash B_i \\ \forall x, x \in VD(\texttt{switch}\ e\ (C_1 \to B_1) \ldots (C_n \to B_n)) \Rightarrow \text{slower}\ H(x)\ (\text{abstract\_clock}\ cs)}{G, H, bs \vdash \texttt{switch}\ e\ (C_1 \to B_1) \ldots (C_n \to B_n)}$$

Switch - Clocking rules

[Colaço, Pagano, and Pouzet (2005): A Conservative Extension of Synchronous Data-flow with State Machines]

Switch - Clocking rules



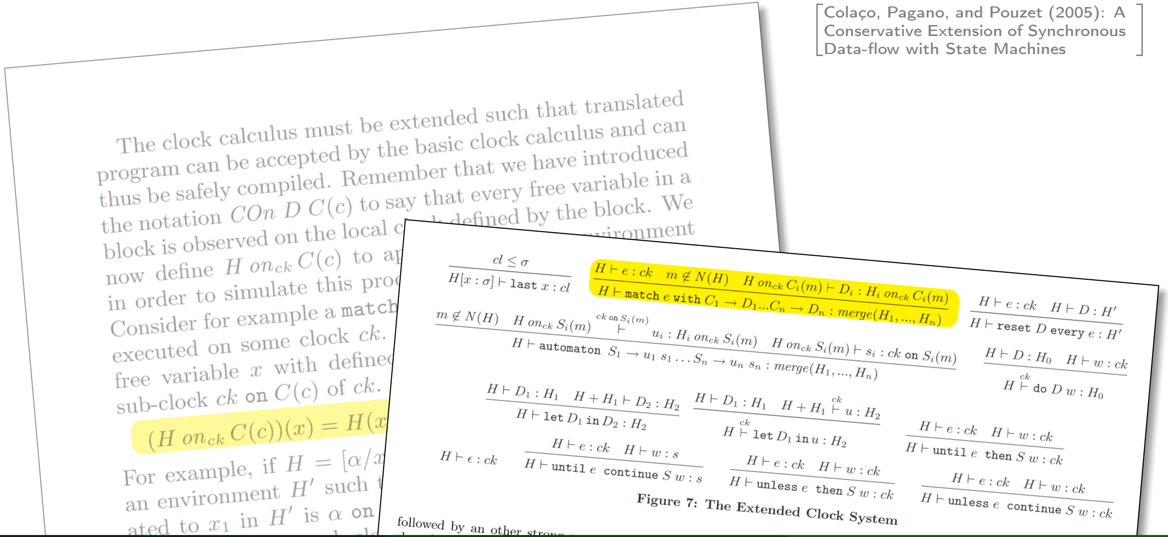
The clock calculus must be extended such that translated program can be accepted by the basic clock calculus and can thus be safely compiled. Remember that we have introduced the notation $COn\ D\ C(c)$ to say that every free variable in a block is observed on the local clock defined by the block. We now define $H\ on_{ck}\ C(c)$ to apply on clocking environment in order to simulate this process during the clock calculus. Consider for example a `match/with` statement which is itself executed on some clock ck. When entering in a branch, a free variable x with defined clock ck will be read on the sub-clock ck **on** $C(c)$ of ck.

$$(H\ on_{ck}\ C(c))(x) = H(x)\ \textbf{on}\ C(c) \text{ provided } H(x) = ck$$

For example, if $H = [\alpha/x_1, \alpha/x_2]$ then $H\ on_{\alpha}\ (C(c) : \alpha)$ is an environment H' such that the clock information associated to x_1 in H' is α **on** $C(c)$. As a consequence, if a free clock ck **on** $C'(c')$ instead of ck, then

[Colaço, Pagano, and Pouzet (2005): A Conservative Extension of Synchronous Data-flow with State Machines]

Switch - Clocking rules



The clock calculus must be extended such that translated
program can be accepted by the basic clock calculus and can
thus be safely compiled. Remember that we have introduced
the notation $COn\ D\ C(c)$ to say that every free variable in a
block is observed on the local c… k defined by the block. We
now define $H\ on_{ck}\ C(c)$ to ap… nvironment
in order to simulate this pro…
Consider for example a match…
executed on some clock ck.
free variable x with define…
sub-clock ck on $C(c)$ of ck.

$(H\ on_{ck}\ C(c))(x) = H(x$

For example, if $H = [\alpha/x$
an environment H' such t
ated to x_1 in H' is α on

$$\frac{cl \leq \sigma}{H[x:\sigma] \vdash \texttt{last}\ x : cl} \qquad \frac{H \vdash e : ck \quad m \notin N(H) \quad H\ on_{ck}\ C_i(m) \vdash D_i : H_i\ on_{ck}\ C_i(m)}{H \vdash \texttt{match}\ e\ \texttt{with}\ C_1 \rightarrow D_1 ... C_n \rightarrow D_n : merge(H_1, ..., H_n)} \qquad \frac{H \vdash e : ck \quad H \vdash D : H'}{H \vdash \texttt{reset}\ D\ \texttt{every}\ e : H'}$$

$$\frac{m \notin N(H) \quad H\ on_{ck}\ S_i(m) \overset{ck\ \mathtt{on}\ S_i(m)}{\vdash} u_i : H_i\ on_{ck}\ S_i(m) \quad H\ on_{ck}\ S_i(m) \vdash s_i : ck\ \mathtt{on}\ S_i(m)}{H \vdash \texttt{automaton}\ S_1 \rightarrow u_1\ s_1 \ldots S_n \rightarrow u_n\ s_n : merge(H_1, ..., H_n)} \qquad \frac{H \vdash D : H_0 \quad H \vdash w : ck}{H \overset{ck}{\vdash} \mathtt{do}\ D\ w : H_0}$$

$$\frac{H \vdash D_1 : H_1 \quad H + H_1 \vdash D_2 : H_2}{H \vdash \texttt{let}\ D_1\ \texttt{in}\ D_2 : H_2} \qquad \frac{H \vdash D_1 : H_1 \quad H + H_1 \overset{ck}{\vdash} u : H_2}{H \overset{ck}{\vdash} \texttt{let}\ D_1\ \texttt{in}\ u : H_2} \qquad \frac{H \vdash e : ck \quad H \vdash w : ck}{H \vdash \texttt{until}\ e\ \ \texttt{then}\ S\ w : ck}$$

$$H \vdash \epsilon : ck \qquad \frac{H \vdash e : ck \quad H \vdash w : s}{H \vdash \texttt{until}\ e\ \ \texttt{continue}\ S\ w : s} \qquad \frac{H \vdash e : ck \quad H \vdash w : ck}{H \vdash \texttt{unless}\ e\ \ \texttt{then}\ S\ w : ck} \qquad \frac{H \vdash e : ck \quad H \vdash w : ck}{H \vdash \texttt{unless}\ e\ \ \texttt{continue}\ S\ w : ck}$$

Figure 7: The Extended Clock System

followed by an other strong…

[Colaço, Pagano, and Pouzet (2005): A Conservative Extension of Synchronous Data-flow with State Machines]

Switch - Clocking rules

$$(H\ on_{ck}\ C(c))(x) = H(x)\ on\ C(c)\ \textbf{provided}\ H(x) = ck$$

[Colaço, Pagano, and Pouzet (2005): A Conservative Extension of Synchronous Data-flow with State Machines]

$$\frac{H \vdash e_c : ck \qquad m \notin N(H) \qquad H\ on_{ck}\ C_i(m) \vdash B_i}{H \vdash \texttt{switch}\ e_c\ (C_1 \rightarrow B_1) \ldots (C_n \rightarrow B_n)}$$

$$\frac{H \vdash e_c : ck \qquad H' \vdash B_i \qquad \forall x, H'(x) = .\ \textbf{provided}\ H(x) = ck}{H \vdash \texttt{switch}\ e_c\ (C_1 \rightarrow B_1) \ldots (C_n \rightarrow B_n)}$$

Switch - Clocking rules

$$(H\ on_{ck}\ C(c))(x) = H(x)\ on\ C(c)\ \textbf{provided}\ H(x) = ck$$

[Colaço, Pagano, and Pouzet (2005): A Conservative Extension of Synchronous Data-flow with State Machines]

introduces a skolem variable

$$\frac{H \vdash e_c : ck \qquad m \notin N(H) \qquad H\ on_{ck}\ C_i(m) \vdash B_i}{H \vdash \texttt{switch}\ e_c\ (C_1 \rightarrow B_1) \ldots (C_n \rightarrow B_n)}$$

$$\frac{H \vdash e_c : ck \qquad H' \vdash B_i \qquad \forall x, H'(x) = .\ \textbf{provided}\ H(x) = ck}{H \vdash \texttt{switch}\ e_c\ (C_1 \rightarrow B_1) \ldots (C_n \rightarrow B_n)}$$

Switch - Clocking rules

$$(H\ on_{ck}\ C(c))(x) = H(x)\ on\ C(c)\ \textbf{provided}\ H(x) = ck$$

[Colaço, Pagano, and Pouzet (2005): A Conservative Extension of Synchronous Data-flow with State Machines]

introduces a skolem variable

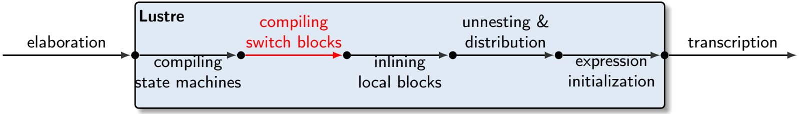
$$\frac{H \vdash e_c : ck \qquad m \notin N(H) \qquad H\ on_{ck}\ C_i(m) \vdash B_i}{H \vdash \texttt{switch}\ e_c\ (C_1 \rightarrow B_1)\ldots(C_n \rightarrow B_n)}$$

---

only one base clock

$$\frac{H \vdash e_c : ck \qquad H' \vdash B_i \qquad \forall x, H'(x) = .\ \textbf{provided}\ H(x) = ck}{H \vdash \texttt{switch}\ e_c\ (C_1 \rightarrow B_1)\ldots(C_n \rightarrow B_n)}$$

Switch - Compilation



elaboration

Lustre

compiling state machines

compiling switch blocks

inlining local blocks

unnesting & distribution

expression initialization

transcription

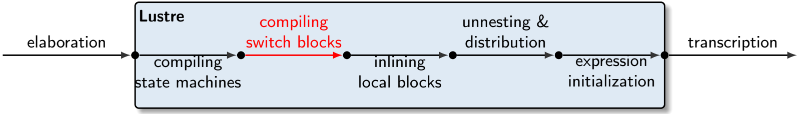
```
type modes = Up | Down

node two(m : modes) returns (o : int)
let
    switch m
    | Up -> o = 1 fby (o + 1)
    | Down -> o = 0
    end
tel
```

```
type modes = Up | Down

node two (m : modes) returns (o : int)
var swi$m$1 : modes when (m=Up); swi$o$2 : int when (m=Up);
    swi$m$3 : modes when (m=Up); swi$o$4 : int when (m=Down);
let
  let
    o = merge m (Up -> swi$o$2) (Down -> swi$o$4);
    swi$o$2 = (1 when (m=Up)) fby (swi$o$2 + (1 when (m=Down)));
    swi$m$1 = m when (m=Up);
    swi$o$4 = 0 when (m=Down);
    swi$m$3 = m when (m=Up);
  tel
tel
```

Switch - Compilation



```
type modes = Up | Down

node two(m : modes) returns (o : int)
let
    switch m
    | Up -> o = 1 fby (o + 1)
    | Down -> o = 0
    end
tel
```

```
type modes = Up | Down

node two (m : modes) returns (o : int)
var swi$m$1 : modes when (m=Up); swi$o$2 : int when (m=Up);
    swi$m$3 : modes when (m=Up); swi$o$4 : int when (m=Down);
let
  let
    o = merge m (Up -> swi$o$2) (Down -> swi$o$4);
    swi$o$2 = (1 when (m=Up)) fby (swi$o$2 + (1 when (m=Down)));
    swi$m$1 = m when (m=Up);
    swi$o$4 = 0 when (m=Down);
    swi$m$3 = m when (m=Up);
  tel
tel
```

Hierarchical State Machines - Example

```
node updown(b : bool) returns (y : int)
let
  automaton
  | U −>
    y = start fby (y + inc);
    until y > 1 restart D;
    until y > 2 restart U
  | D −>
    y = start fby (y − inc);
    until y < −2 resume U
  end
  initially D if false; I otherwise
tel
```

base	T	T	T	T	T	T	T	T	T	T	T	...
b	F	F	F	F	F	F	F	F	F	F	F	...
y	0	1	2	0	-1	-2	3	0	1	2	0	...

Hierarchical State Machines - Example

```
node updown(b : bool) returns (y : int)
let
  automaton
  | U ->
    y = start fby (y + inc);
    until y > 1 restart D;
    until y > 2 restart U
  | D ->
    y = start fby (y - inc);
    until y < -2 resume U
  end
  initially D if false; I otherwise
tel
```

base	T	T	T	T	T	T	T	T	T	T	T	...
b	F	F	F	F	F	F	F	F	F	F	F	...
y	0	1	2	0	-1	-2	3	0	1	2	0	...
state	U	U	U	D	D	D	U	U	U	U	D	...
reset	F	F	F	T	F	F	F	T	F	F	T	...

Hierarchical State Machines - Transition stream

Hierarchical State Machines semantics can be encoded reactive or coiterative semantics

- [Colaço, Hamon, and Pouzet (2006): Mixing Signals and Modes in Synchronous Data-flow Systems]
- [Caspi and Pouzet (1997): A Co-iterative Characterization of Synchronous Stream Functions]

Hierarchical State Machines - Transition stream

Hierarchical State Machines semantics can be encoded reactive or coiterative semantics

- [Colaço, Hamon, and Pouzet (2006): Mixing Signals and Modes in Synchronous Data-flow Systems]
- [Caspi and Pouzet (1997): A Co-iterative Characterization of Synchronous Stream Functions]

It doesn't seem to be possible to mix-and-match these styles with our stream semantics
Instead, we encode a stream of entering transitions

- at each instant, indicates which state is entered, and if it is entered with reset
- transitions can be absent if the state machine is inactive
- only weak transitions (for the moment)

Hierarchical State Machines - Transitions

$$\text{const-st}\ (\mathrm{T} \cdot \mathit{bs})\ \mathit{st} \equiv \langle \mathit{st} \rangle \cdot \text{const-st}\ \mathit{bs}\ \mathit{st}$$

$$\text{const-st}\ (\mathrm{F} \cdot \mathit{bs})\ \mathit{st} \equiv \langle\, \rangle \cdot \text{const-st}\ \mathit{bs}\ \mathit{st}$$

$$\frac{G, H, \mathit{bs} \vdash e \Downarrow [\mathit{ys}] \qquad \text{bools-of}\ \mathit{ys} \doteq \mathit{bs}'}{G, H, \mathit{bs} \vdash \mathtt{until}\ e\ \mathtt{resume}\ C \Downarrow (\text{const-st}\ \mathit{bs}'\ (C, \mathrm{F}))}$$

$$\frac{G, H, \mathit{bs} \vdash e \Downarrow [\mathit{ys}] \qquad \text{bools-of}\ \mathit{ys} \doteq \mathit{bs}'}{G, H, \mathit{bs} \vdash \mathtt{until}\ e\ \mathtt{restart}\ C \Downarrow (\text{const-st}\ \mathit{bs}'\ (C, \mathrm{T}))}$$

Hierarchical State Machines - Transitions

$$\text{const-st}\ (\mathrm{T} \cdot bs)\ st \equiv \langle st \rangle \cdot \text{const-st}\ bs\ st$$

$$\text{const-st}\ (\mathrm{F} \cdot bs)\ st \equiv \langle\rangle \cdot \text{const-st}\ bs\ st$$

$$\frac{G, H, bs \vdash e \Downarrow [ys] \qquad \text{bools-of}\ ys \doteq bs'}{G, H, bs \vdash \texttt{until}\ e\ \texttt{resume}\ C \Downarrow (\text{const-st}\ bs'\ (C, \mathrm{F}))}$$

$$\frac{G, H, bs \vdash e \Downarrow [ys] \qquad \text{bools-of}\ ys \doteq bs'}{G, H, bs \vdash \texttt{until}\ e\ \texttt{restart}\ C \Downarrow (\text{const-st}\ bs'\ (C, \mathrm{T}))}$$

$$\text{choose-fst}\ (\langle\rangle \cdot vs_1) \ldots (\langle v \rangle \cdot vs_k) \ldots (v_n \cdot vs_n) \equiv \langle v \rangle \cdot (\text{choose-fst}\ vs_1 \ldots vs_n)$$

$$\text{choose-fst}\ (\langle\rangle \cdot vs_1) \ldots (\langle\rangle \cdot vs_n) \equiv \langle\rangle \cdot (\text{choose-fst}\ vs_1 \ldots vs_n)$$

$$\frac{G, H, bs \vdash until_i \Downarrow ts_i}{G, H, bs \vdash (C \to \texttt{until\_1} \ldots \texttt{until\_n}) \Downarrow \text{choose-fst}\ ts_1 \ldots ts_n\ (\text{const-st}\ bs\ (C, \mathrm{F}))}$$

Hierarchical State Machines - Putting it all together

$$\frac{\begin{array}{c}(H_i, bs_i) = \mathrm{filter}^{C}_{\pi_1(ts)}\,(H, bs) \qquad rs_i = \mathrm{filter}^{C}_{\pi_1(ts)}\,\pi_2(ts) \\ \forall k. G, \mathrm{mask}^{k}_{rs_i}(H_i, bs_i) \vdash B \qquad \forall k. G, \mathrm{mask}^{k}_{rs_i}(H_i, bs_i) \vdash \mathit{untils} \Downarrow ts'\end{array}}{G, H, bs, ts \vdash (C \rightarrow B; \mathit{untils}) \Downarrow ts'}$$

Hierarchical State Machines - Putting it all together

$$\frac{\begin{array}{c}(H_i, bs_i) = \mathrm{filter}^{C}_{\pi_1(ts)}\,(H, bs) \qquad rs_i = \mathrm{filter}^{C}_{\pi_1(ts)}\,\pi_2(ts) \\ \forall k. G, \mathrm{mask}^{k}_{rs_i}(H_i, bs_i) \vdash B \qquad \forall k. G, \mathrm{mask}^{k}_{rs_i}(H_i, bs_i) \vdash \mathit{untils} \Downarrow ts'\end{array}}{G, H, bs, ts \vdash (C \rightarrow B; \mathit{untils}) \Downarrow ts'}$$

$$\frac{\begin{array}{c}G, H, bs \vdash \mathit{autinits} \Downarrow ts_0 \\ G, H, bs, ts \vdash \mathit{autst}_i \Downarrow ts_i \\ \mathrm{fby}\ ts_0\ ts_1 \doteq ts\end{array}}{G, H, bs \vdash \texttt{automaton}\ \mathit{autst}_1 \ldots \mathit{autst}_n\ \texttt{initially}\ \mathit{autinits}}$$

Hierarchical State Machines - Putting it all together

$$\frac{\begin{array}{c}(H_i, bs_i) = \mathrm{filter}^{C}_{\pi_1(ts)}\,(H, bs) \qquad rs_i = \mathrm{filter}^{C}_{\pi_1(ts)}\,\pi_2(ts) \\ \forall k. G, \mathrm{mask}^{k}_{rs_i}(H_i, bs_i) \vdash B \qquad \forall k. G, \mathrm{mask}^{k}_{rs_i}(H_i, bs_i) \vdash \mathit{untils} \Downarrow ts'\end{array}}{G, H, bs, ts \vdash (C \rightarrow B; \mathit{untils}) \Downarrow ts'}$$

$$\frac{\text{constrains-present } xs\ ys}{\text{constrains-present } (\langle x \rangle \cdot xs)\ (\langle x \rangle \cdot ys)} \qquad \frac{\text{constrains-present } xs\ ys}{\text{constrains-present } (\langle\,\rangle \cdot xs)\ (y \cdot ys)}$$

$$\frac{\begin{array}{c}G, H, bs \vdash \mathit{autinits} \Downarrow ts_0 \\ G, H, bs, ts \vdash \mathit{autst}_i \Downarrow ts_i \qquad \text{constrains-present } ts_i\ ts_1 \\ \text{fby } ts_0\ ts_1 \doteq ts\end{array}}{G, H, bs \vdash \mathtt{automaton}\ \mathit{autst}_1 \ldots \mathit{autst}_n\ \mathtt{initially}\ \mathit{autinits}}$$

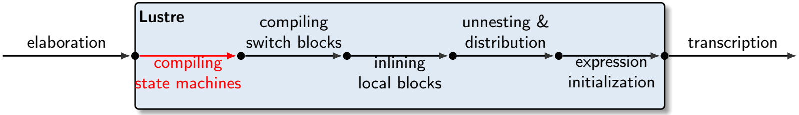
Hierarchical State Machines - Putting it all together

$$\frac{\begin{array}{c}(H_i, bs_i) = \text{filter}^{C}_{\pi_1(ts)}\,(H, bs) \qquad rs_i = \text{filter}^{C}_{\pi_1(ts)}\,\pi_2(ts) \\ \forall k. G, \text{mask}^{k}_{rs_i}(H_i, bs_i) \vdash B \qquad \forall k. G, \text{mask}^{k}_{rs_i}(H_i, bs_i) \vdash \mathit{untils} \Downarrow ts'\end{array}}{G, H, bs, ts \vdash (C \to B; \mathit{untils}) \Downarrow ts'}$$

$$\frac{\text{constrains-present}\ xs\ ys}{\text{constrains-present}\ (\langle x \rangle \cdot xs)\ (\langle x \rangle \cdot ys)} \quad \frac{\text{constrains-present}\ xs\ ys}{\text{constrains-present}\ (\langle\rangle \cdot xs)\ (y \cdot ys)}$$

$$\frac{\begin{array}{c}G, H, bs \vdash \mathit{autinits} \Downarrow ts_0 \\ G, H, bs, ts \vdash \mathit{autst}_i \Downarrow ts_i \qquad \text{constrains-present}\ ts_i\ ts_1 \qquad \text{slower}\ ts_1\ bs \\ \text{fby}\ ts_0\ ts_1 \doteq ts\end{array}}{G, H, bs \vdash \texttt{automaton}\ \mathit{autst}_1 \ldots \mathit{autst}_n\ \texttt{initially}\ \mathit{autinits}}$$

Hierarchical State Machines - Compilation



Lustre

elaboration

compiling state machines

compiling switch blocks

inlining local blocks

unnesting & distribution

expression initialization

transcription

```
node updown(b : bool) returns (y : int)
let
  automaton
  | U ->
    y = start fby (y + inc);
    until y > 1 restart D;
    until y > 2 restart U
  | D ->
    y = start fby (y - inc);
    until y < -2 resume U
  end
  initially D if false; I otherwise
tel
```

```
type ty$1 = U | D

node updown(b : bool) returns (y : int)
var st$1, pst$1 : ty$1; res$1, pres$1 : bool;
let
  st$1 = (if b then D else I) fby pst$1;
  res$1 = false fby pres$1;
  switch st$1
  | U ->
    reset
      y = start fby (y + inc);
      (pst$1, pres$1) = if y > 1 then (D, true) else if y > 2 then (U, true) else (U, false);
    every res$1
  | D ->
    reset
      y = start fby (y - inc);
      (pst$1, pres$1) = if y < -2 then (U, false) else (D, false);
    every res$1
  end
tel
```

Shared variables ?

```
node updown() returns (y : int)
var last x : int = 0;
let y = x;
    automaton
    | Up ->
        x = last x + 1;
        until x > 2 resume Down
    | Down ->
        x = last x − 1;
        until x <= 0 resume Up
    initially Up
tel
```

last x	0	1	2	3	2	1	0	1	...
x, y	1	2	3	2	1	0	1	2	...

Shared variables ?

```
node updown() returns (y : int)
var last x : int = 0;
let y = x;
    automaton
    | Up –>
        x = last x + 1;
        until x > 2 resume Down
    | Down –>
        x = last x − 1;
        until x <= 0 resume Up
    initially Up
tel
```

last x	0	1	2	3	2	1	0	1	...
x, y	1	2	3	2	1	0	1	2	...

Shared variables ?

```
node updown() returns (y : int)
var last x : int = 0;
let y = x;
    automaton
    | Up ->
        x = last x + 1;
        until x > 2 resume Down
    | Down ->
        x = last x - 1;
        until x <= 0 resume Up
    initially Up
tel
```

last x	0	1	2	3	2	1	0	1	...
x, y	1	2	3	2	1	0	1	2	...

```
node updown() returns (y : int)
var x, px : int;
let y = x;
    px = 0 fby x;
    automaton
    | Up ->
        x = px + 1;
        until x > 2 resume Down
    | Down ->
        x = px - 1;
        until x <= 0 resume Up
    initially Up
tel
```

What's next ?

What's left to do:

- Dead code optimization
- Specification and compilation of state machines
- Specification and compilation of last expressions

What's next ?

What's left to do:

- Dead code optimization
- Specification and compilation of state machines
- Specification and compilation of last expressions

What I want to explore next:

- Coiterative interpreter in Velus / proof of existence
- Link with the work of Paul Jeanmaire

What’s next ?

What’s left to do:

- Dead code optimization
- Specification and compilation of state machines
- Specification and compilation of last expressions

What I want to explore next:

- Coiterative interpreter in Velus / proof of existence
- Link with the work of Paul Jeanmaire
- Specifying and adding external (C) nodes in Velus

References

Caspi, P. and M. Pouzet (Oct. 1997). *A Co-iterative Characterization of Synchronous Stream Functions*. Research Report 97-07. Gières, France: VERIMAG.

Colaço, J.-L., G. Hamon, and M. Pouzet (Oct. 2006). "Mixing Signals and Modes in Synchronous Data-flow Systems". In: *Proc. 6th ACM Int. Conf. on Embedded Software (EMSOFT 2006)*. Ed. by S. L. Min and Y. Wang. Seoul, South Korea: ACM Press, pp. 73–82.

Colaço, J.-L., B. Pagano, and M. Pouzet (Sept. 2005). "A Conservative Extension of Synchronous Data-flow with State Machines". In: *Proc. 5th ACM Int. Conf. on Embedded Software (EMSOFT 2005)*. Ed. by W. Wolf. Jersey City, USA: ACM Press, pp. 173–182.