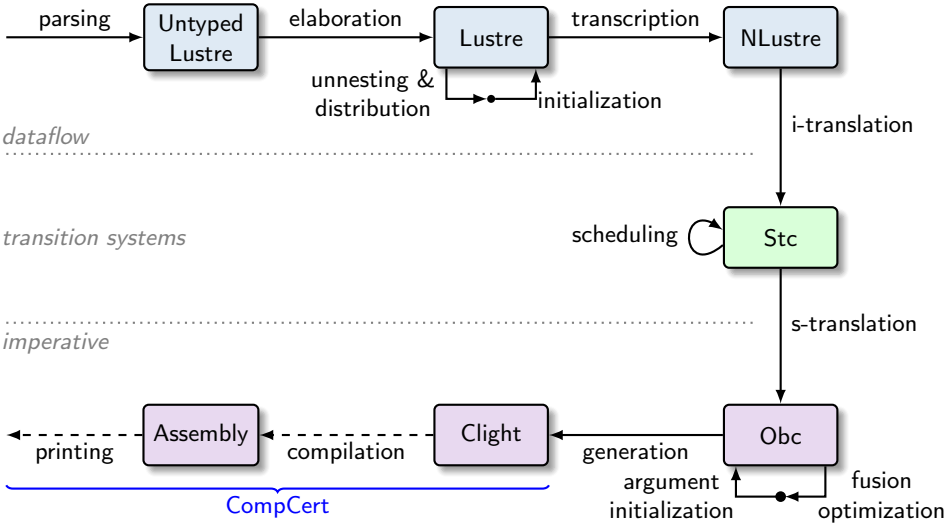
Verified Compilation of Synchronous Dataflow with State Machines

Basile Pesin, Timothy Bourke and Marc Pouzet

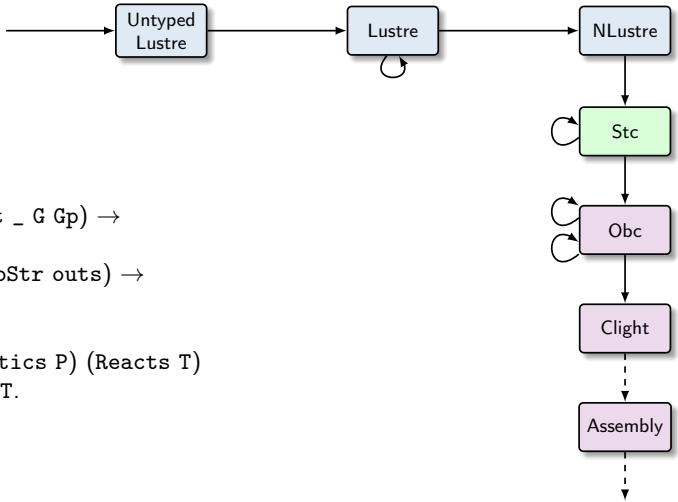
Inria - PARKAS

SYNCHRON 2022

The Vélus Compiler, v2.0

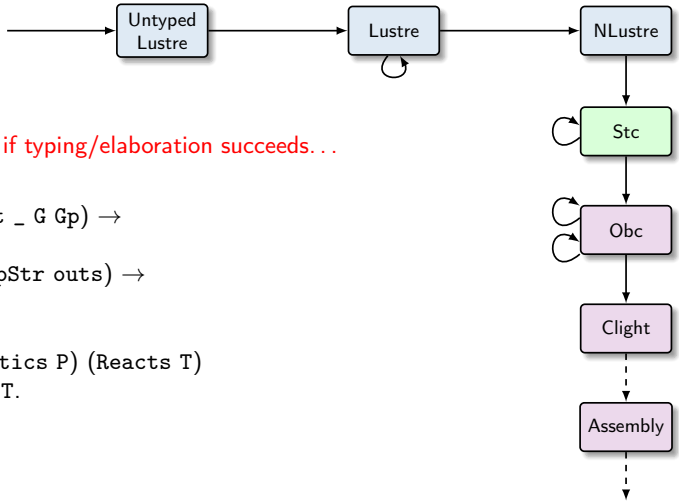


Main Correctness Theorem




```
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 Correctness Theorem



if typing/elaboration succeeds...

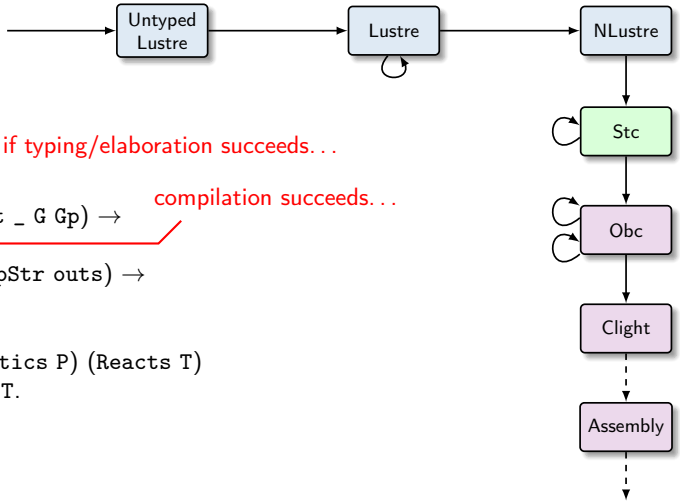
```
Theorem behavior_asm:
  ∀ D G Gp P main ins outs,
    elab_declarations D = OK (exist _ G Gp) →
    compile D main = OK P →
    Sem.sem_node G main (pStr ins) (pStr outs) →
    wt_ins G main ins →
    wc_ins G main ins →
    ∃ T, program_behaves (Asm.semantics P) (Reacts T)
         ∧ bisim_IO G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

Main Correctness 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...



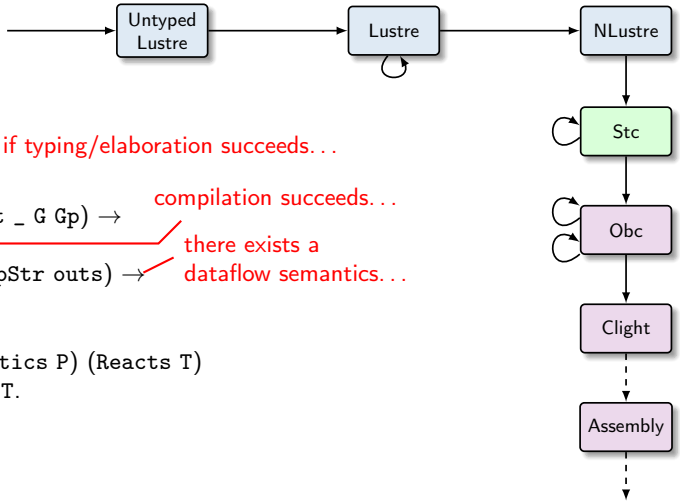
Main Correctness Theorem

```
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.
```

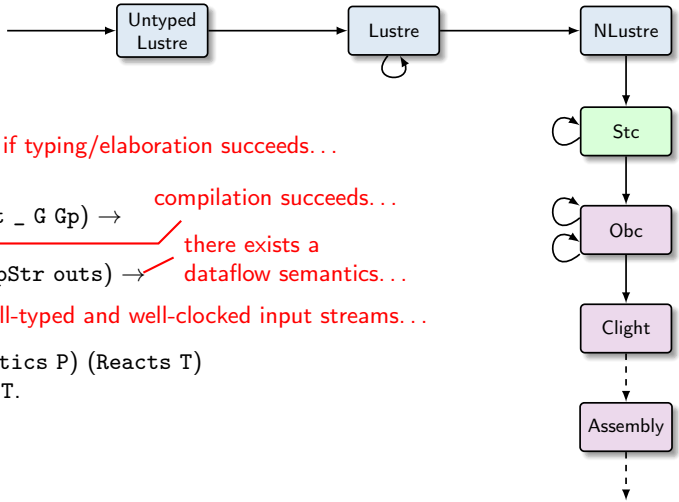
if typing/elaboration succeeds...

compilation succeeds...

there exists a dataflow semantics...



Main Correctness 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.
```

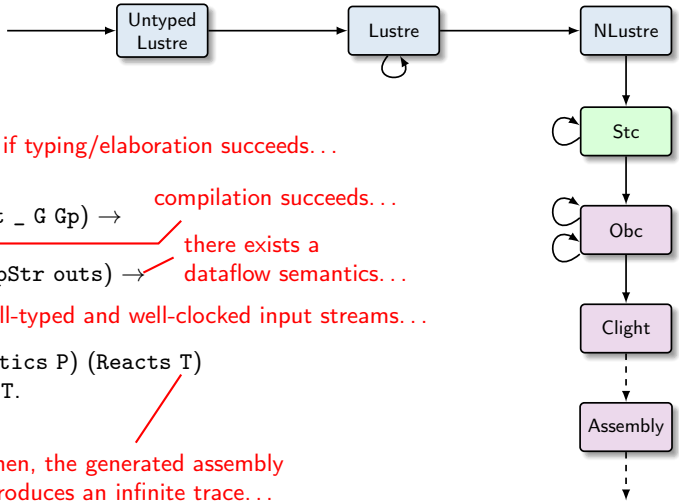
if typing/elaboration succeeds...

compilation succeeds...

there exists a dataflow semantics...

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

Main Correctness 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...

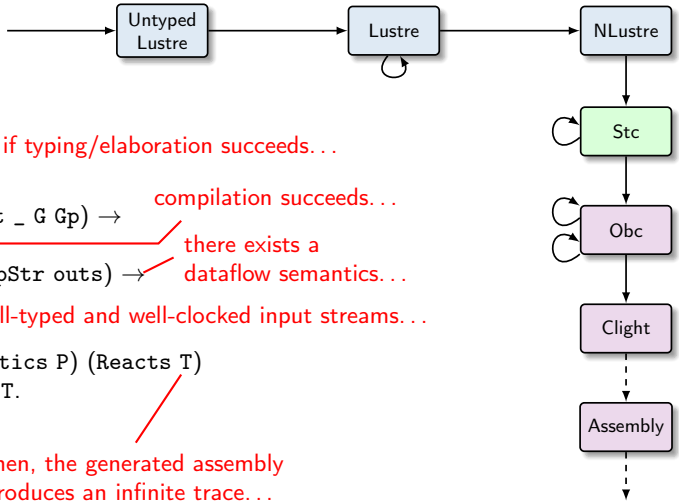
compilation succeeds...

there exists a dataflow semantics...

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

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

Main Correctness 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.

Previously, in SYNCHRON

Last year, during a cold November afternoon, we discussed:

- Stream semantics for Reset blocks, Switch blocks, State Machines with Weak Preemption
- Verified Compilation of Reset and Switch blocks [Colaço, Pagano, and Pouzet (2005): A Conservative Extension of Synchronous Data-flow with State Machines]

Previously, in SYNCHRON

Last year, during a cold November afternoon, we discussed:

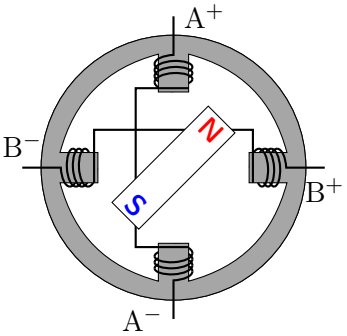
- Stream semantics for Reset blocks, Switch blocks, State Machines with Weak Preemption
- Verified Compilation of Reset and Switch blocks [Colaço, Pagano, and Pouzet (2005): A Conservative Extension of Synchronous Data-flow with State Machines]

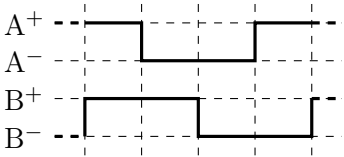
Since then, I've been working on:

- Adding shared variables (`last`)
- Stream semantics for State Machines with Strong Preemption
- Verified compilation of State Machines
- Refining dependency analysis and associated proofs

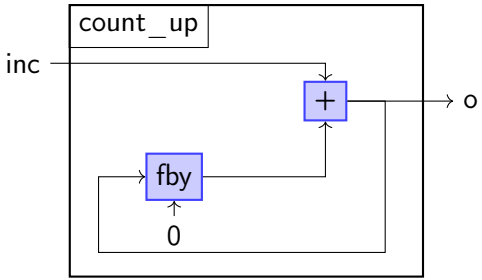
An embedded system example : stepper motor for a small printer

- 4 windings , energized by `enable` and controlled by `mA` and `mB`
- can be paused: the motor needs to stop turning
- when starting/restarting, needs to speed up first
- when paused, less energy should be sent to the windings : PWM



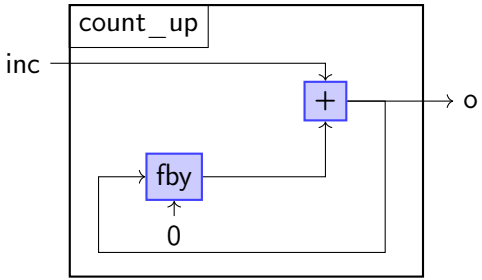


The Lustre Programming Language



inc	15	15	15	15	15	15	15	...
out	15	30	45	60	75	90	105	...

The Lustre Programming Language



inc	15	15	15	15	15	15	15	...
out	15	30	45	60	75	90	105	...

```
node count_up(inc : int) returns (o : int)
let
  o = (0 fby o) + inc;
tel
```

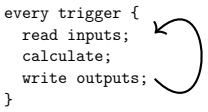
Dataflow Semantics of Lustre

inc	15	15	15	15	15	15	15	...
out	15	30	45	60	75	90	105	...

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

Dataflow Semantics of Lustre

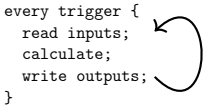
inc	15	15	15	15	15	15	15	...
out	15	30	45	60	75	90	105	...



$$\frac{H(x) \equiv vs}{G, H, bs \vdash x \Downarrow [vs]}$$

Dataflow Semantics of Lustre

inc	15	15	15	15	15	15	15	...
out	15	30	45	60	75	90	105	...

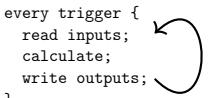


$$\frac{H(x) \equiv vs}{G, H, bs \vdash x \Downarrow [vs]}$$

$$\frac{G, H, bs \vdash es \Downarrow (vs_1, \ldots, vs_n) \qquad \forall i \in 1...n,\ H(x_i) \equiv vs_i}{G, H, bs \vdash (x_1, \ldots, x_n) = es}$$

Dataflow Semantics of Lustre

inc	15	15	15	15	15	15	15	...
out	15	30	45	60	75	90	105	...



$$\frac{H(x) \equiv vs}{G, H, bs \vdash x \Downarrow [vs]} \qquad \frac{G, H, bs \vdash es \Downarrow (vs_1, \ldots, vs_n) \qquad \forall i \in 1...n,\ H(x_i) \equiv vs_i}{G, H, bs \vdash (x_1, \ldots, x_n) = es}$$

$$\frac{G(f) = \texttt{node}\ f(x_1, \ldots, x_n)\ \texttt{returns}\ (y_1, \ldots, y_m)\ blk \qquad \forall i \in 1...n,\ H(x_i) \equiv xs_i \qquad \forall i \in 1...m,\ H(y_i) \equiv ys_i \qquad G, H, (\text{base-of}\ (xs_1, \ldots, xs_n)) \vdash blk}{G \vdash f(xs_1, \ldots, xs_n) \Downarrow (ys_1, \ldots, ys_m)}$$

Lustre fby operator semantics

$$\begin{aligned}
\text{fby}\ (\langle\rangle \cdot xs)\ (\langle\rangle \cdot ys) &\equiv \langle\rangle \cdot \text{fby}\ xs\ ys \\
\text{fby}\ (\langle v_1 \rangle \cdot xs)\ (\langle v_2 \rangle \cdot ys) &\equiv \langle v_1 \rangle \cdot \text{fby1}\ v_2\ xs\ ys
\end{aligned}$$

Lustre fby operator semantics

$$\begin{aligned}
\text{fby}\ (\langle\rangle \cdot xs)\ (\langle\rangle \cdot ys) &\equiv \langle\rangle \cdot \text{fby}\ xs\ ys \\
\text{fby}\ (\langle v_1 \rangle \cdot xs)\ (\langle v_2 \rangle \cdot ys) &\equiv \langle v_1 \rangle \cdot \text{fby1}\ v_2\ xs\ ys
\end{aligned}$$

$$\begin{aligned}
\text{fby1}\ v_0\ (\langle\rangle \cdot xs)\ (\langle\rangle \cdot ys) &\equiv \langle\rangle \cdot \text{fby1}\ v_0\ xs\ ys \\
\text{fby1}\ v_0\ (\langle v_1 \rangle \cdot xs)\ (\langle v_2 \rangle \cdot ys) &\equiv \langle v_0 \rangle \cdot \text{fby1}\ v_2\ xs\ ys
\end{aligned}$$

Lustre fby operator semantics

$$
\begin{aligned}
\text{fby}\ (\langle\rangle \cdot xs)\ (\langle\rangle \cdot ys) &\equiv \langle\rangle \cdot \text{fby}\ xs\ ys \\
\text{fby}\ (\langle v_1 \rangle \cdot xs)\ (\langle v_2 \rangle \cdot ys) &\equiv \langle v_1 \rangle \cdot \text{fby1}\ v_2\ xs\ ys
\end{aligned}
$$

$$
\begin{aligned}
\text{fby1}\ v_0\ (\langle\rangle \cdot xs)\ (\langle\rangle \cdot ys) &\equiv \langle\rangle \cdot \text{fby1}\ v_0\ xs\ ys \\
\text{fby1}\ v_0\ (\langle v_1 \rangle \cdot xs)\ (\langle v_2 \rangle \cdot ys) &\equiv \langle v_0 \rangle \cdot \text{fby1}\ v_2\ xs\ ys
\end{aligned}
$$

$$
\frac{G, H, bs \vdash es_0 \Downarrow (xs_1, \ldots, xs_n) \qquad G, H, bs \vdash es_1 \Downarrow (ys_1, \ldots, ys_n) \qquad \forall i \in 1..n,\ \text{fby}\ xs_i\ ys_i \equiv vs_i}{G, H, bs \vdash es_0\ \texttt{fby}\ es_1 \Downarrow (vs_1, \ldots, vs_n)}
$$

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

```
type t = A | B | C

node f(c : t; x : int)
returns (y : int)
let
    switch c
    | A do y = (0 fby y) + x
    | B do y = (0 fby y) - x
    | C do y = 0
    end
tel
```

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

$$
\begin{array}{lcl}
\text{when}^C\ (\langle\rangle \cdot xs)\ (\langle\rangle \cdot ys) & \equiv & \langle\rangle \cdot \text{when}^C\ xs\ ys \\
\text{when}^C\ (\langle v\rangle \cdot xs)\ (\langle C\rangle \cdot ys) & \equiv & \langle v\rangle \cdot \text{when}^C\ xs\ ys \\
\text{when}^C\ (\langle v\rangle \cdot xs)\ (\langle C'\rangle \cdot ys) & \equiv & \langle\rangle \cdot \text{when}^C\ xs\ ys
\end{array}
$$

```
type t = A | B | C

node f(c : t; x : int)
returns (y : int)
let
    switch c
    | A do y = (0 fby y) + x
    | B do y = (0 fby y) - x
    | C do y = 0
    end
tel
```

cs										...
xs	1	1	1	1	1	1	1	1	1	...
$\text{when}^A\ ys\ cs$										...
$\text{when}^B\ ys\ cs$										...
$\text{when}^C\ ys\ cs$										...
ys										...

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

$$
\begin{array}{lcl}
\text{when}^C\ (\langle\rangle \cdot xs)\ (\langle\rangle \cdot ys) & \equiv & \langle\rangle \cdot \text{when}^C\ xs\ ys \\
\text{when}^C\ (\langle v\rangle \cdot xs)\ (\langle C\rangle \cdot ys) & \equiv & \langle v\rangle \cdot \text{when}^C\ xs\ ys \\
\text{when}^C\ (\langle v\rangle \cdot xs)\ (\langle C'\rangle \cdot ys) & \equiv & \langle\rangle \cdot \text{when}^C\ xs\ ys
\end{array}
$$

```
type t = A | B | C

node f(c : t; x : int)
returns (y : int)
let
    switch c
    | A do y = (0 fby y) + x
    | B do y = (0 fby y) - x
    | C do y = 0
    end
tel
```

cs	A	A				A				...
xs	1	1	1	1	1	1	1	1	1	...
$\text{when}^A\ ys\ cs$	1	2				3				...
$\text{when}^B\ ys\ cs$										...
$\text{when}^C\ ys\ cs$										...
ys	1	2				3				...

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

$$
\begin{array}{lcl}
\text{when}^C\ (\langle\rangle \cdot xs)\ (\langle\rangle \cdot ys) & \equiv & \langle\rangle \cdot \text{when}^C\ xs\ ys \\
\text{when}^C\ (\langle v\rangle \cdot xs)\ (\langle C\rangle \cdot ys) & \equiv & \langle v\rangle \cdot \text{when}^C\ xs\ ys \\
\text{when}^C\ (\langle v\rangle \cdot xs)\ (\langle C'\rangle \cdot ys) & \equiv & \langle\rangle \cdot \text{when}^C\ xs\ ys
\end{array}
$$

```
type t = A | B | C

node f(c : t; x : int)
returns (y : int)
let
    switch c
    | A do y = (0 fby y) + x
    | B do y = (0 fby y) - x
    | C do y = 0
    end
tel
```

cs			B		B				B	...
xs	1	1	1	1	1	1	1	1	1	...
$\text{when}^A\ ys\ cs$										...
$\text{when}^B\ ys\ cs$			-1		-2				-3	...
$\text{when}^C\ ys\ cs$										...
ys			-1		-2				-3	...

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

$$
\begin{aligned}
\text{when}^C\ (\langle\rangle \cdot xs)\ (\langle\rangle \cdot ys) &\equiv \langle\rangle \cdot \text{when}^C\ xs\ ys \\
\text{when}^C\ (\langle v\rangle \cdot xs)\ (\langle C\rangle \cdot ys) &\equiv \langle v\rangle \cdot \text{when}^C\ xs\ ys \\
\text{when}^C\ (\langle v\rangle \cdot xs)\ (\langle C'\rangle \cdot ys) &\equiv \langle\rangle \cdot \text{when}^C\ xs\ ys
\end{aligned}
$$

```
type t = A | B | C

node f(c : t; x : int)
returns (y : int)
let
    switch c
    | A do y = (0 fby y) + x
    | B do y = (0 fby y) - x
    | C do y = 0
    end
tel
```

cs				C			C	C		...
xs	1	1	1	1	1	1	1	1	1	...
$\text{when}^A\ ys\ cs$										...
$\text{when}^B\ ys\ cs$										...
$\text{when}^C\ ys\ cs$				0			0	0		...
ys				0			0	0		...

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

$$
\begin{array}{lcl}
\text{when}^C\ (\langle\rangle \cdot xs)\ (\langle\rangle \cdot ys) & \equiv & \langle\rangle \cdot \text{when}^C\ xs\ ys \\
\text{when}^C\ (\langle v\rangle \cdot xs)\ (\langle C\rangle \cdot ys) & \equiv & \langle v\rangle \cdot \text{when}^C\ xs\ ys \\
\text{when}^C\ (\langle v\rangle \cdot xs)\ (\langle C'\rangle \cdot ys) & \equiv & \langle\rangle \cdot \text{when}^C\ xs\ ys
\end{array}
$$

```
type t = A | B | C

node f(c : t; x : int)
returns (y : int)
let
    switch c
    | A do y = (0 fby y) + x
    | B do y = (0 fby y) - x
    | C do y = 0
    end
tel
```

cs	A	A	B	C	B	A	C	C	B	...
xs	1	1	1	1	1	1	1	1	1	...
$\text{when}^A\ ys\ cs$	1	2	⟨⟩	⟨⟩	⟨⟩	3	⟨⟩	⟨⟩	⟨⟩	...
$\text{when}^B\ ys\ cs$	⟨⟩	⟨⟩	-1	⟨⟩	-2	⟨⟩	⟨⟩	⟨⟩	-3	...
$\text{when}^C\ ys\ cs$	⟨⟩	⟨⟩	⟨⟩	0	⟨⟩	⟨⟩	0	0	⟨⟩	...
ys	1	2	-1	0	-2	3	0	0	-3	...

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

```
node f(r : bool)
returns (y : int)
let
  reset
    y = (0 fby y) + 1;
  every r;
tel
```

$$\begin{aligned}
\mathsf{mask}^{k}_{k'}\,(\mathtt{F}\cdot rs)\,(sv\cdot xs) &\equiv (\text{if } k'=k \text{ then } sv \text{ else } \langle\rangle)\cdot \mathsf{mask}^{k}_{k'}\; rs\; xs\\
\mathsf{mask}^{k}_{k'}\,(\mathtt{T}\cdot rs)\,(sv\cdot xs) &\equiv (\text{if } k'+1=k \text{ then } sv \text{ else } \langle\rangle)\cdot \mathsf{mask}^{k}_{k'+1}\; rs\; xs
\end{aligned}$$

```
node f(r : bool)
returns (y : int)
let
  reset
    y = (0 fby y) + 1;
  every r;
tel
```

rs	F	F	T	T	F	F	T	F	...
$\mathsf{mask}^0\; rs\; ys$									...
$\mathsf{mask}^1\; rs\; ys$									...
$\mathsf{mask}^2\; rs\; ys$									...
$\mathsf{mask}^3\; rs\; ys$									...
...									...
ys									...

$$\begin{aligned}
\mathrm{mask}^{k}_{k'}\,(\mathtt{F}\cdot rs)\,(sv\cdot xs) &\equiv (\text{if } k'=k \text{ then } sv \text{ else } \langle\rangle)\cdot \mathrm{mask}^{k}_{k'}\,rs\;xs\\
\mathrm{mask}^{k}_{k'}\,(\mathtt{T}\cdot rs)\,(sv\cdot xs) &\equiv (\text{if } k'+1=k \text{ then } sv \text{ else } \langle\rangle)\cdot \mathrm{mask}^{k}_{k'+1}\,rs\;xs
\end{aligned}$$

```
node f(r : bool)
returns (y : int)
let
  reset
    y = (0 fby y) + 1;
  every r;
tel
```

rs	F	F	T	T	F	F	T	F	...
$\mathrm{mask}^0\ rs\ ys$	1	2							...
$\mathrm{mask}^1\ rs\ ys$									...
$\mathrm{mask}^2\ rs\ ys$									...
$\mathrm{mask}^3\ rs\ ys$									...
...									...
ys	1	2							...

$$\begin{aligned}
\mathrm{mask}^{k}_{k'}\,(\mathtt{F}\cdot rs)\,(sv\cdot xs) &\equiv (\text{if } k'=k \text{ then } sv \text{ else } \langle\rangle)\cdot \mathrm{mask}^{k}_{k'}\,rs\,xs\\
\mathrm{mask}^{k}_{k'}\,(\mathtt{T}\cdot rs)\,(sv\cdot xs) &\equiv (\text{if } k'+1=k \text{ then } sv \text{ else } \langle\rangle)\cdot \mathrm{mask}^{k}_{k'+1}\,rs\,xs
\end{aligned}$$

```
node f(r : bool)
returns (y : int)
let
  reset
    y = (0 fby y) + 1;
  every r;
tel
```

rs	F	F	T	T	F	F	T	F	...
$\mathrm{mask}^0\ rs\ ys$									...
$\mathrm{mask}^1\ rs\ ys$			1						...
$\mathrm{mask}^2\ rs\ ys$									...
$\mathrm{mask}^3\ rs\ ys$									...
...									...
ys			1						...

$$\begin{aligned}
\mathrm{mask}^{k}_{k'}\,(\mathtt{F}\cdot rs)\,(sv\cdot xs) &\equiv (\text{if } k' = k \text{ then } sv \text{ else } \langle\rangle)\cdot \mathrm{mask}^{k}_{k'}\; rs\; xs\\
\mathrm{mask}^{k}_{k'}\,(\mathtt{T}\cdot rs)\,(sv\cdot xs) &\equiv (\text{if } k'+1 = k \text{ then } sv \text{ else } \langle\rangle)\cdot \mathrm{mask}^{k}_{k'+1}\; rs\; xs
\end{aligned}$$

```
node f(r : bool)
returns (y : int)
let
  reset
    y = (0 fby y) + 1;
  every r;
tel
```

rs	F	F	T	T	F	F	T	F	...
mask^0 *rs ys*									...
mask^1 *rs ys*									...
mask^2 *rs ys*				1	2	3			...
mask^3 *rs ys*									...
...									...
ys				1	2	3			...

$$\begin{aligned}
\mathsf{mask}^k_{k'}\,(\mathtt{F}\cdot rs)\,(sv\cdot xs) &\equiv (\text{if } k'=k \text{ then } sv \text{ else } \langle\rangle)\cdot \mathsf{mask}^k_{k'}\,rs\,xs\\
\mathsf{mask}^k_{k'}\,(\mathtt{T}\cdot rs)\,(sv\cdot xs) &\equiv (\text{if } k'+1=k \text{ then } sv \text{ else } \langle\rangle)\cdot \mathsf{mask}^k_{k'+1}\,rs\,xs
\end{aligned}$$

```
node f(r : bool)
returns (y : int)
let
  reset
    y = (0 fby y) + 1;
  every r;
tel
```

rs	F	F	T	T	F	F	T	F	...
$\mathsf{mask}^0\ rs\ ys$									...
$\mathsf{mask}^1\ rs\ ys$									...
$\mathsf{mask}^2\ rs\ ys$									...
$\mathsf{mask}^3\ rs\ ys$							1	2	...
...									...
ys							1	2	...

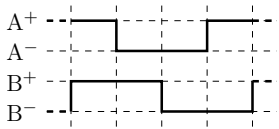
$$\begin{aligned}
\mathrm{mask}^k_{k'}\,(\mathtt{F}\cdot rs)\,(sv\cdot xs) &\equiv (\text{if } k'=k \text{ then } sv \text{ else } \langle\rangle)\cdot \mathrm{mask}^k_{k'}\ rs\ xs\\
\mathrm{mask}^k_{k'}\,(\mathtt{T}\cdot rs)\,(sv\cdot xs) &\equiv (\text{if } k'+1=k \text{ then } sv \text{ else } \langle\rangle)\cdot \mathrm{mask}^k_{k'+1}\ rs\ xs
\end{aligned}$$

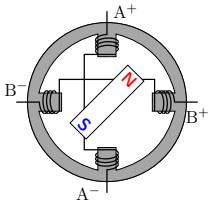
```
node f(r : bool)
returns (y : int)
let
  reset
    y = (0 fby y) + 1;
  every r;
tel
```

rs	F	F	T	T	F	F	T	F	...
mask^0 *rs ys*	1	2	⟨⟩	⟨⟩	⟨⟩	⟨⟩	⟨⟩	⟨⟩	...
mask^1 *rs ys*	⟨⟩	⟨⟩	1	⟨⟩	⟨⟩	⟨⟩	⟨⟩	⟨⟩	...
mask^2 *rs ys*	⟨⟩	⟨⟩	⟨⟩	1	2	3	⟨⟩	⟨⟩	...
mask^3 *rs ys*	⟨⟩	⟨⟩	⟨⟩	⟨⟩	⟨⟩	⟨⟩	1	2	...
...									...
ys	1	2	1	1	2	3	1	2	...

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

```
mA = true fby not mB;
mB = false fby mA;
```

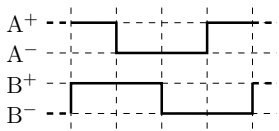


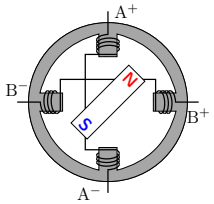


mA	T	T	F	F	T	T	F	F	...
mB	F	T	T	F	F	T	T	F	...

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

```
node drive_sequence(step : bool)
returns (mA, mB : bool)
var mA, mB : bool;
let
  switch step
  | true do
    mA = true fby not mB;
    mB = false fby mA;
  | false do (mA, mB) = (false, false)
  end;
tel
```



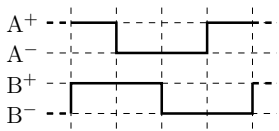


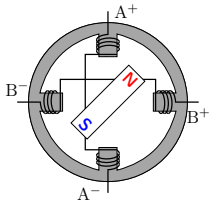
mA	T	T	F	F	T	T	F	F	...
mB	F	T	T	F	F	T	T	F	...

step	F	T	T	F	F	T	F	T	F	T	F	T	F	F	T	...
mA	F	T	T	F	F	F	F	F	F	T	F	T	F	F	F	...
mB	F	F	T	F	F	T	F	F	F	F	F	T	F	F	T	...

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

```
node drive_sequence(step : bool)
returns (motorA, motorB : bool)
var last mA : bool = true; last mB : bool = false;
let
  switch step
  | true do
     mA = not (last mB);
     mB = last mA;
  | false do (mA, mB) = (last mA, last mB)
  end;
  (motorA, motorB) = (mA, mB);
tel
```

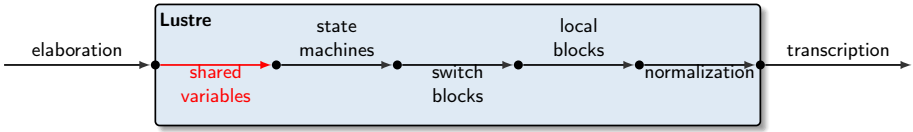




mA	T	T	F	F	T	T	F	F	...
mB	F	T	T	F	F	T	T	F	...

step	F	T	T	F	F	T	F	T	F	T	F	T	F	F	T	...
last mA	T	T	T	F	F	F	F	F	T	T	T	T	F	F	F	...
last mB	F	F	T	T	T	T	F	F	F	F	T	T	T	T	T	...
mA	T	T	F	F	F	F	F	T	T	T	T	F	F	F	F	...
mB	F	T	T	T	T	F	F	F	F	T	T	T	T	T	F	...

Shared variables - Compilation




```
node drive_sequence(step : bool)
returns (motorA, motorB : bool)
var last mA : bool = true; last mB : bool = false;
let
  switch step
  | true do
     mA = not (last mB);
     mB = last mA;
  | false do (mA, mB) = (last mA, last mB)
  end;
  (motorA, motorB) = (mA, mB);
tel
```

```
node drive_sequence (step : bool)
returns (motorA, motorB : bool)
var mA, mB, last$mA, last$mB : bool)
let
  last$mA = true fby mA;
  last$mB = false fby mB;
  switch step
  | true do (mA, mB) = (not last$mB, last$mA)
  | false do (mA, mB) = (last$mA, last$mB)
  end;
  (motorA, motorB) = (mA, mB)
tel
```

Shared variables - Semantic rules

$$\frac{H(\text{last } x) \equiv vs}{G, H, bs \vdash \texttt{last}\ x \Downarrow [vs]}$$

Shared variables - Semantic rules

$$\frac{H(\text{last } x) \equiv vs}{G, H, bs \vdash \texttt{last}\, x \Downarrow [vs]}$$

$$\frac{G, H + H', bs \vdash e \Downarrow [vs_0] \qquad H'(x) \equiv vs_1 \qquad H'(\text{last } x) \equiv \text{fby } vs_0\ vs_1}{G, H + H', bs \vdash \texttt{last}\, x = e}$$

$$(H_1 + H_2)(x) = \begin{cases} H_2(x) & \text{if } x \in H_2 \\ H_1(x) & \text{otherwise.} \end{cases}$$

Hierarchical State Machines - Example

```
node feed_pause(pause : bool) returns (ena, step : bool)
var time : int;
let
  automaton initially R
  state R do time = count_up(50);
  until step then R;

  automaton initially Feeding
    state Feeding do
      ena = true;
      automaton initially Starting
        state Starting do
          step = true -> false;
        unless false -> time >= 750 then Moving

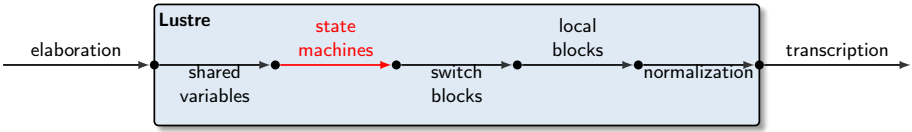
        state Moving do
          step = true -> false;
        unless time >= 500 then Moving
      end;
    unless pause then Holding

    state Holding do
      step = false;
      automaton initially Waiting
        state Waiting do
          ena = true;
        unless time >= 500 then Modulating

        state Modulating do
          ena = pwm(true);
      end;
    unless
      | not pause and time >= 750 then Feeding
      | not pause continue Feeding
  end
tel
```

Hierarchical State Machines - Compilation

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



$\lfloor$ `automaton initially inits (state` C_i `var locs do blks until trans)`$_i$ `end` $\rfloor \triangleq$

`var` $x_{st}, x_{res}, x_{nst}, x_{nres}$ `let`

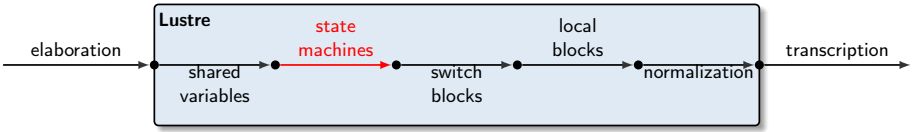
$(x_{st}, x_{res}) = \lfloor inits \rfloor$ `fby` (x_{nst}, x_{nres});

`switch` x_{st} $(C_i$ `do reset var locs let` $\lfloor blks \rfloor; (x_{nst}, x_{nres}) = \lfloor trans \rfloor_{C_i}$ `tel every` $x_{res})_i$ `end`

`tel`

Hierarchical State Machines - Compilation

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



$\lfloor$`automaton initially inits (state` C_i `var locs do blks until trans)`$_i$ `end` $\rfloor \triangleq$

`var` $x_{st}, x_{res}, x_{nst}, x_{nres}$ `let`

$(x_{st}, x_{res}) = \lfloor inits \rfloor$ `fby` (x_{nst}, x_{nres});

`switch` x_{st} $(C_i$ `do reset var locs let` $\lfloor blks \rfloor; (x_{nst}, x_{nres}) = \lfloor trans \rfloor_{C_i}$ `tel every` $x_{res})_i$ `end`

`tel`

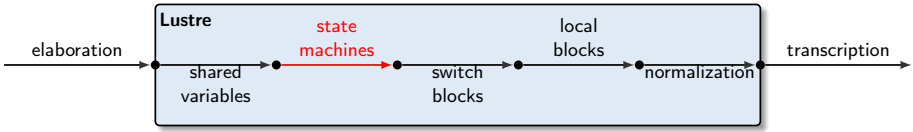
$$\lfloor \epsilon \rfloor_{C_d} \triangleq (C_d, \texttt{false})$$

$$\lfloor \texttt{| e then C;} trans \rfloor_{C_d} \triangleq \texttt{if e then (C, true) else} \lfloor trans \rfloor_{C_d}$$

$$\lfloor \texttt{| e continue C;} trans \rfloor_{C_d} \triangleq \texttt{if e then (C, false) else} \lfloor trans \rfloor_{C_d}$$

Hierarchical State Machines - Compilation

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



$$\lfloor \texttt{automaton initially C (state } C_i \texttt{ do } blks_i \texttt{ unless } trans_i)_i \texttt{ end} \rfloor \triangleq$$

$$\texttt{var } x_{st}, x_{res}, x_{nst}, x_{nres} \texttt{ let}$$

$$(x_{nst}, x_{nres}) = (C, \texttt{false}) \texttt{ fby } (x_{st}, x_{res});$$

$$\texttt{switch } x_{nst} \ (C_i \texttt{ do reset } (x_{st}, x_{res}) = \lfloor trans_i \rfloor_{C_i} \texttt{ every } x_{nres})_i \texttt{ end};$$

$$\texttt{switch } x_{st} \ (C_i \texttt{ do reset } \lfloor blks_i \rfloor \texttt{ every } x_{res})_i \texttt{ end}$$

$$\texttt{tel}$$

$$\lfloor \epsilon \rfloor_{C_d} \triangleq (C_d, \texttt{false})$$

$$\lfloor \texttt{| e then C;} trans \rfloor_{C_d} \triangleq \texttt{if e then (C, true) else} \lfloor trans \rfloor_{C_d}$$

$$\lfloor \texttt{| e continue C;} trans \rfloor_{C_d} \triangleq \texttt{if e then (C, false) else} \lfloor trans \rfloor_{C_d}$$

Hierarchical State Machines - Stream interpretation

`automaton` = `switch` + `reset`; translation semantics ? [Colaço, Pagano, and Pouzet (2005): A Conservative Extension of Synchronous Data-flow with State Machines]

Hierarchical State Machines - Stream interpretation

`automaton` = `switch` + `reset`; translation semantics ? [Colaço, Pagano, and Pouzet (2005): A Conservative Extension of Synchronous Data-flow with State Machines]

$$\text{select}^{C,k}_{k'}\ sts\ xs \equiv \text{mask}^{k}_{k'}\ (\text{when}^{C}\ \pi_2(sts)\ \pi_1(sts))\ (\text{when}^{C}\ xs\ \pi_1(sts))$$

Hierarchical State Machines - Stream interpretation

`automaton` = `switch` + `reset`; translation semantics ? [Colaço, Pagano, and Pouzet (2005): A Conservative Extension of Synchronous Data-flow with State Machines]

$$\text{select}_{k'}^{C,k}\ sts\ xs \equiv \text{mask}_{k'}^{k}\ (\text{when}^{C}\ \pi_2(sts)\ \pi_1(sts))\ (\text{when}^{C}\ xs\ \pi_1(sts))$$

Case-by-base coinductive definition:

$$\begin{aligned}
&\text{select}_{k'}^{C,k}\ (\langle\rangle \cdot sts)\ (\langle\rangle \cdot xs) && \equiv && \langle\rangle \cdot \text{select}_{k'}^{C,k}\ sts\ xs \\
&\text{select}_{k'}^{C,k}\ (\langle C, \mathrm{F}\rangle \cdot sts)\ (\langle v\rangle \cdot xs) && \equiv && (\text{if } k' = k \text{ then } \langle v\rangle \text{ else } \langle\rangle) \cdot \text{select}_{k'}^{C,k}\ sts\ xs \\
&\text{select}_{k'}^{C,k}\ (\langle C, \mathrm{T}\rangle \cdot sts)\ (\langle v\rangle \cdot xs) && \equiv && (\text{if } k'+1 = k \text{ then } \langle v\rangle \text{ else } \langle\rangle) \cdot \text{select}_{k'+1}^{C,k}\ sts\ xs \\
&\text{select}_{k'}^{C,k}\ (\langle C', b\rangle \cdot sts)\ (\langle v\rangle \cdot xs) && \equiv && \langle\rangle \cdot \text{select}_{k'}^{C,k}\ sts\ xs
\end{aligned}$$

Dependency Analysis

The semantics presented above doesn't guarantee everything, even for a well-typed program. Consider a program with the following definitions:

Dependency Analysis

The semantics presented above doesn't guarantee everything, even for a well-typed program. Consider a program with the following definitions:

- `x = x;` could take any value

Dependency Analysis

The semantics presented above doesn't guarantee everything, even for a well-typed program. Consider a program with the following definitions:

- `x = x;` could take any value
- `x = x + 1;` doesn't have any value

Dependency Analysis

The semantics presented above doesn't guarantee everything, even for a well-typed program. Consider a program with the following definitions:

- `x = x;` could take any value
- `x = x + 1;` doesn't have any value
- `x = if c then y + 1 else 0;`
  `y = if c then 0 else x - 1;`
  does have a single value, but cannot be statically scheduled

We reject such programs with a static analysis [Halbwachs, Caspi, Raymond, and Pilaud (1991): The synchronous dataflow programming language LUSTRE]

Dependency Analysis

The semantics presented above doesn't guarantee everything, even for a well-typed program. Consider a program with the following definitions:

- `x = x;` could take any value
- `x = x + 1;` doesn't have any value
- ```
  x = if c then y + 1 else 0;
  y = if c then 0 else x - 1;
  ```
  does have a single value, but cannot be statically scheduled

We reject such programs with a static analysis [Halbwachs, Caspi, Raymond, and Pilaud (1991): The synchronous dataflow programming language LUSTRE]

First version presented in [Bourke, Jeanmaire, Pesin, and Pouzet (2021): Verified Lustre Normalization with Node Subsampling], extended since.

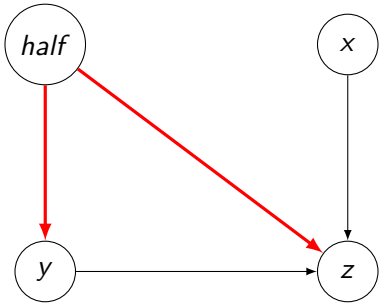
- Treatement of activation structures using labels
- Improved graph analysis algorithm
- Used to prove more properties of the semantics (clock system correctness, determinism)
  ```

Dependency Analysis of the Dataflow Language

```
node f(x : int) returns (y, z : int)
var half : bool;
let
    half = true fby (not half);
    (y, z) = if half then (0, x) else (1, y);
tel
```

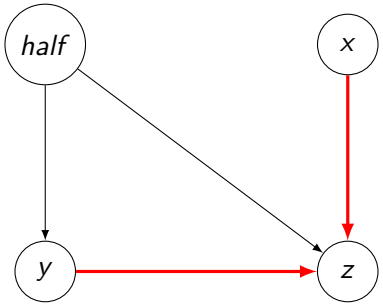
Dependency Analysis of the Dataflow Language

```
node f(x : int) returns (y, z : int)
var half : bool;
let
    half = true fby (not half);
    (y, z) = if half then (0, x) else (1, y);
tel
```



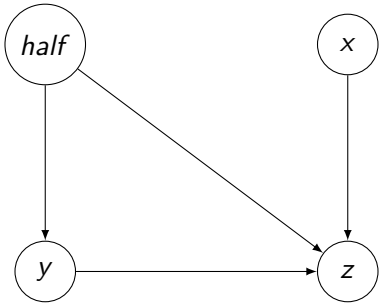
Dependency Analysis of the Dataflow Language

```
node f(x : int) returns (y, z : int)
var half : bool;
let
    half = true fby (not half);
    (y, z) = if half then (0, x) else (1, y);
tel
```



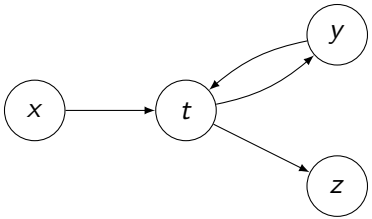
Dependency Analysis of the Dataflow Language

```
node f(x : int) returns (y, z : int)
var half : bool;
let
    half = true fby (not half);
    (y, z) = if half then (0, x) else (1, y);
tel
```



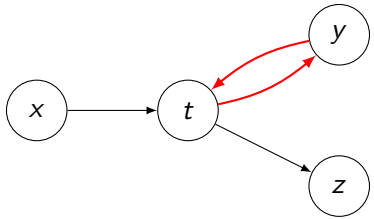
Dependencies in Local Scopes

```
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
```



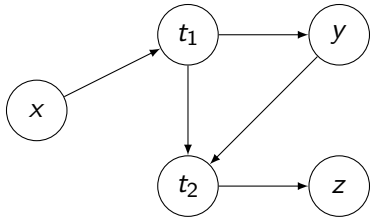
Dependencies in Local Scopes

```
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
```



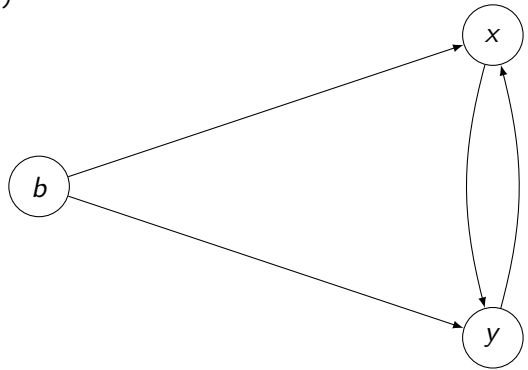
Dependencies in Local Scopes

```
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
```



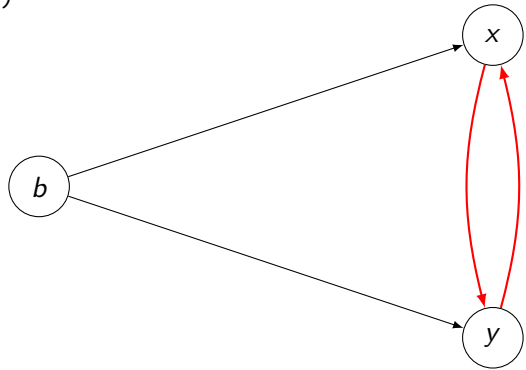
Dependencies in branches

```
node f(b : bool) returns (x, y : bool)
let
    switch b
    | true do
        x = 0 fby (x - 1);
        y = x * 2;
    | false do
        y = 0 fby (y + 1);
        x = y * 2;
    end
tel
```



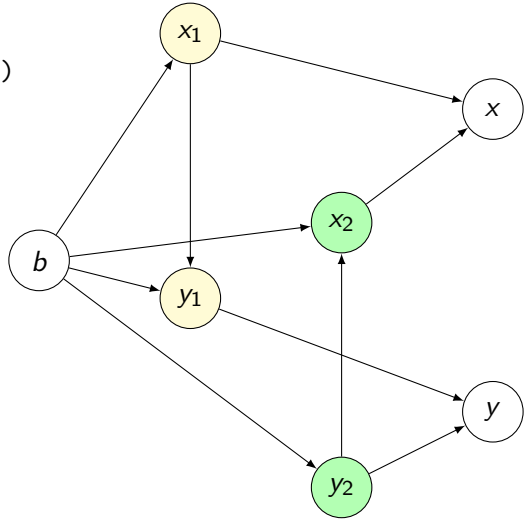
Dependencies in branches

```
node f(b : bool) returns (x, y : bool)
let
    switch b
    | true do
        x = 0 fby (x - 1);
        y = x * 2;
    | false do
        y = 0 fby (y + 1);
        x = y * 2;
    end
tel
```



Dependencies in branches

```
node f(b : bool) returns (x, y : bool)
let
    switch b
    | true do
        x^x1 = 0 fby (x^x1 - 1);
        y^y1 = x^x1 * 2;
    | false do
        y^y2 = 0 fby (y^y2 + 1);
        x^x2 = y^y2 * 2;
    end
tel
```



Dependencies and shared variables

```
var last x = 0;
let x = last x + 1;
tel
```

last x	0	1	2	3	...
x	1	2	3	4	...

```
var last x = x + 1;
let x = 0;
tel
```

x	0	0	0	0	...
last x	1	1	1	1	...

Dependencies and shared variables

```
var last x^x2 = 0;
let x^x1 = last x^x2 + 1;
tel
```

last x	0	1	2	3	...
x	1	2	3	4	...

```
var last x^x2 = x^x1 + 1;
let x^x1 = 0;
tel
```

x	0	0	0	0	...
last x	1	1	1	1	...

Dependency graph analysis

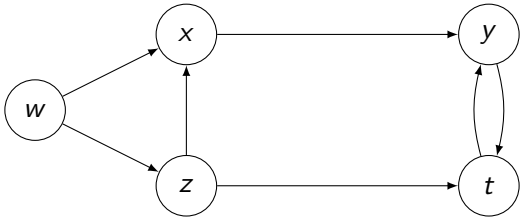
$$\frac{}{\text{AcyGraph } \emptyset\ \emptyset}$$

$$\frac{\text{AcyGraph } V\ E}{\text{AcyGraph } (V \cup \{x\})\ E}$$

$$\frac{\text{AcyGraph } V\ E \qquad x, y \in V \qquad y \twoheadrightarrow_E^* x}{\text{AcyGraph } V\ (E \cup \{x \to y\})}$$

Dependency graph analysis

$$\frac{}{\text{AcyGraph } \emptyset\ \emptyset} \qquad \frac{\text{AcyGraph } V\ E}{\text{AcyGraph } (V \cup \{x\})\ E} \qquad \frac{\text{AcyGraph } V\ E \qquad x, y \in V \qquad y \twoheadrightarrow_E^* x}{\text{AcyGraph } V\ (E \cup \{x \to y\})}$$

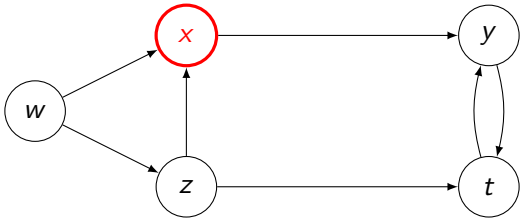


Dependency graph analysis

$$\frac{}{\text{AcyGraph } \emptyset\ \emptyset}$$

$$\frac{\text{AcyGraph } V\ E}{\text{AcyGraph } (V \cup \{x\})\ E}$$

$$\frac{\text{AcyGraph } V\ E \qquad x, y \in V \qquad y \twoheadrightarrow_E^* x}{\text{AcyGraph } V\ (E \cup \{x \to y\})}$$

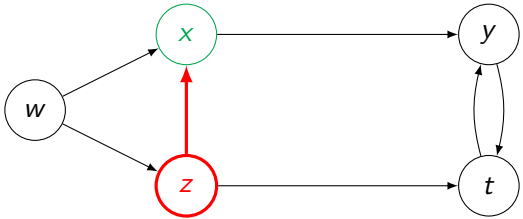


Dependency graph analysis

$$\frac{}{\mathsf{AcyGraph}\ \emptyset\ \emptyset}$$

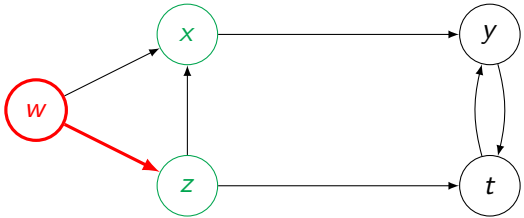
$$\frac{\mathsf{AcyGraph}\ V\ E}{\mathsf{AcyGraph}\ (V \cup \{x\})\ E}$$

$$\frac{\mathsf{AcyGraph}\ V\ E \qquad x, y \in V \qquad y \twoheadrightarrow_E^* x}{\mathsf{AcyGraph}\ V\ (E \cup \{x \to y\})}$$



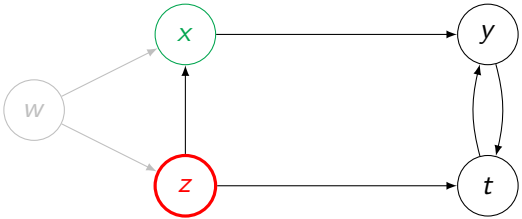
Dependency graph analysis

$$\frac{}{\text{AcyGraph } \emptyset\ \emptyset} \qquad \frac{\text{AcyGraph } V\ E}{\text{AcyGraph } (V \cup \{x\})\ E} \qquad \frac{\text{AcyGraph } V\ E \qquad x, y \in V \qquad y \twoheadrightarrow_E^* x}{\text{AcyGraph } V\ (E \cup \{x \to y\})}$$



Dependency graph analysis

$$\frac{}{\text{AcyGraph } \emptyset\ \emptyset} \qquad \frac{\text{AcyGraph } V\ E}{\text{AcyGraph } (V \cup \{x\})\ E} \qquad \frac{\text{AcyGraph } V\ E \qquad x, y \in V \qquad y \twoheadrightarrow_E^* x}{\text{AcyGraph } V\ (E \cup \{x \to y\})}$$

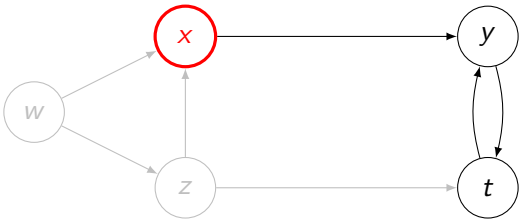


Dependency graph analysis

$$\frac{}{\text{AcyGraph } \emptyset\ \emptyset}$$

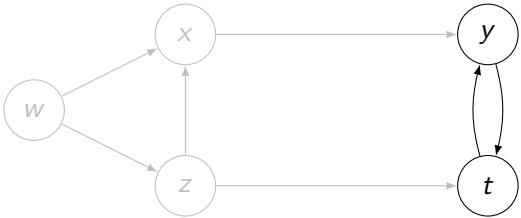
$$\frac{\text{AcyGraph } V\ E}{\text{AcyGraph } (V \cup \{x\})\ E}$$

$$\frac{\text{AcyGraph } V\ E \qquad x, y \in V \qquad y \twoheadrightarrow_E^* x}{\text{AcyGraph } V\ (E \cup \{x \to y\})}$$



Dependency graph analysis

$$\frac{}{\text{AcyGraph } \emptyset\ \emptyset} \qquad \frac{\text{AcyGraph } V\ E}{\text{AcyGraph } (V \cup \{x\})\ E} \qquad \frac{\text{AcyGraph } V\ E \qquad x, y \in V \qquad y \twoheadrightarrow_E^* x}{\text{AcyGraph } V\ (E \cup \{x \to y\})}$$

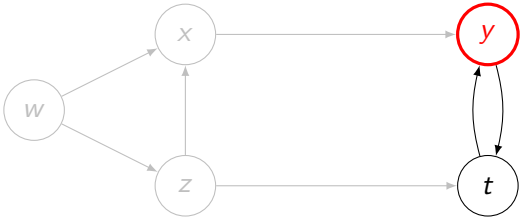


Dependency graph analysis

$$\frac{}{\text{AcyGraph } \emptyset\ \emptyset}$$

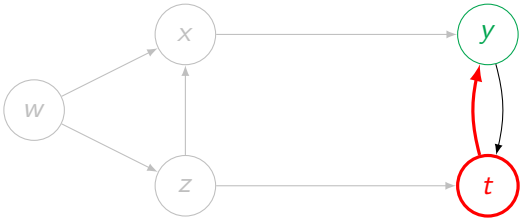
$$\frac{\text{AcyGraph } V\ E}{\text{AcyGraph } (V \cup \{x\})\ E}$$

$$\frac{\text{AcyGraph } V\ E \qquad x, y \in V \qquad y \twoheadrightarrow_E^* x}{\text{AcyGraph } V\ (E \cup \{x \to y\})}$$



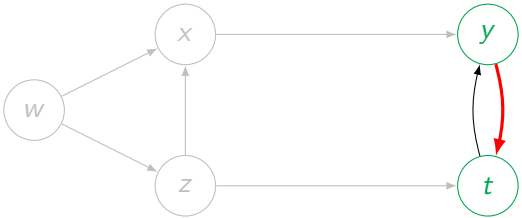
Dependency graph analysis

$$\frac{}{\text{AcyGraph } \emptyset\ \emptyset} \qquad \frac{\text{AcyGraph } V\ E}{\text{AcyGraph } (V \cup \{x\})\ E} \qquad \frac{\text{AcyGraph } V\ E \qquad x, y \in V \qquad y \twoheadrightarrow_E^* x}{\text{AcyGraph } V\ (E \cup \{x \to y\})}$$



Dependency graph analysis

$$\frac{}{\text{AcyGraph } \emptyset\ \emptyset} \qquad \frac{\text{AcyGraph } V\ E}{\text{AcyGraph } (V \cup \{x\})\ E} \qquad \frac{\text{AcyGraph } V\ E \qquad x, y \in V \qquad y \twoheadrightarrow_E^* x}{\text{AcyGraph } V\ (E \cup \{x \to y\})}$$



Dependency graph analysis

$$\frac{}{\mathsf{AcyGraph}\ \emptyset\ \emptyset} \qquad \frac{\mathsf{AcyGraph}\ V\ E}{\mathsf{AcyGraph}\ (V \cup \{x\})\ E} \qquad \frac{\mathsf{AcyGraph}\ V\ E \qquad x, y \in V \qquad y \twoheadrightarrow_E^* x}{\mathsf{AcyGraph}\ V\ (E \cup \{x \to y\})}$$

```
Definition visited (p : set) (v : set) : Prop :=
    (∀ x, x ∈ p → ¬(x ∈ v))
  ∧ ∃ a, AcyGraph v a
        ∧ (∀ x, x ∈ v → ∃ zs, graph(x) = Some zs
                          ∧ (∀ y, y ∈ zs → has_arc a y x)).

Program Fixpoint dfs'
  (s : { p | ∀ x, x ∈ p → x ∈ graph }) (x : ident)
  (v : { v | visited s v }) {measure (|graph| - |s|)}
  : option { v' | visited s v' & x ∈ v' ∧ v ⊆ v' } := ...
```

Dependency graph analysis

$$\frac{}{\mathsf{AcyGraph}\ \emptyset\ \emptyset} \qquad \frac{\mathsf{AcyGraph}\ V\ E}{\mathsf{AcyGraph}\ (V \cup \{x\})\ E} \qquad \frac{\mathsf{AcyGraph}\ V\ E \qquad x, y \in V \qquad y \twoheadrightarrow_E^* x}{\mathsf{AcyGraph}\ V\ (E \cup \{x \to y\})}$$

```
Definition visited (p : set) (v : set) : Prop :=
    (∀ x, x ∈ p → ¬(x ∈ v))
  ∧ ∃ a, AcyGraph v a
        ∧ (∀ x, x ∈ v → ∃ zs, graph(x) = Some zs
                              ∧ (∀ y, y ∈ zs → has_arc a y x)).

Program Fixpoint dfs'
  (s : { p | ∀ x, x ∈ p → x ∈ graph }) (x : ident)
  (v : { v | visited s v }) {measure (|graph| - |s|)}
  : option { v' | visited s v' & x ∈ v' ∧ v ⊆ v' } := ...
```

Dependency graph analysis

$$\frac{}{\mathsf{AcyGraph}\ \emptyset\ \emptyset} \qquad \frac{\mathsf{AcyGraph}\ V\ E}{\mathsf{AcyGraph}\ (V \cup \{x\})\ E} \qquad \frac{\mathsf{AcyGraph}\ V\ E \qquad x, y \in V \qquad y \twoheadrightarrow_E^* x}{\mathsf{AcyGraph}\ V\ (E \cup \{x \to y\})}$$

```
Definition visited (p : set) (v : set) : Prop :=
    (∀ x, x ∈ p → ¬(x ∈ v))
  ∧ ∃ a, AcyGraph v a
        ∧ (∀ x, x ∈ v → ∃ zs, graph(x) = Some zs
                              ∧ (∀ y, y ∈ zs → has_arc a y x)).

Program Fixpoint dfs'
  (s : { p | ∀ x, x ∈ p → x ∈ graph }) (x : ident)
  (v : { v | visited s v }) {measure (|graph| - |s|)}
  : option { v' | visited s v' & x ∈ v' ∧ v ⊆ v' } := ...
```

Dependency graph analysis

$$\frac{}{\mathsf{AcyGraph}\ \emptyset\ \emptyset} \qquad \frac{\mathsf{AcyGraph}\ V\ E}{\mathsf{AcyGraph}\ (V \cup \{x\})\ E} \qquad \frac{\mathsf{AcyGraph}\ V\ E \qquad x, y \in V \qquad y \twoheadrightarrow_E^* x}{\mathsf{AcyGraph}\ V\ (E \cup \{x \to y\})}$$

```
Definition visited (p : set) (v : set) : Prop :=
    (∀ x, x ∈ p → ¬(x ∈ v))
  ∧ ∃ a, AcyGraph v a
        ∧ (∀ x, x ∈ v → ∃ zs, graph(x) = Some zs
                              ∧ (∀ y, y ∈ zs → has_arc a y x)).

Program Fixpoint dfs'
  (s : { p | ∀ x, x ∈ p → x ∈ graph }) (x : ident)
  (v : { v | visited s v }) {measure (|graph| - |s|)}
  : option { v' | visited s v' & x ∈ v' ∧ v ⊆ v' } := ...
```

Proving with dependencies

$$\frac{}{\mathsf{TopoOrder}\ (\mathsf{AcyGraph}\ V\ E)\ []}$$

$$\frac{\mathsf{TopoOrder}\ (\mathsf{AcyGraph}\ V\ E)\ l \qquad x \in V \qquad \neg \mathsf{In}\ x\ l \qquad (\forall y,\ y \to_E^* x \implies \mathsf{In}\ y\ l)}{\mathsf{TopoOrder}\ (\mathsf{AcyGraph}\ V\ E)\ (x :: l)}$$

Proving with dependencies

$$\frac{}{\mathsf{TopoOrder}\ (\mathsf{AcyGraph}\ V\ E)\ []} \qquad \frac{\mathsf{TopoOrder}\ (\mathsf{AcyGraph}\ V\ E)\ l \qquad x \in V \qquad \neg \mathsf{In}\ x\ l \qquad (\forall y,\ y \rightarrow_E^* x \implies \mathsf{In}\ y\ l)}{\mathsf{TopoOrder}\ (\mathsf{AcyGraph}\ V\ E)\ (x :: l)}$$

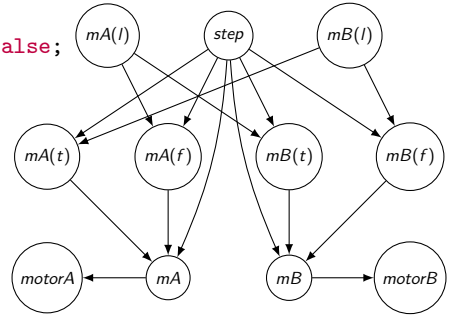
```
node drive_sequence(step : bool)
returns (motorA, motorB : bool)
var last mA^mA(l) : bool = true; last mB^mB(l) : bool = false;
let
  switch step
  | true do
     mA^mA(t) = not (last mB);
     mB^mB(t) = last mA;
  | false do (mA^mA(f), mB^mB(f)) = (last mA, last mB)
  end;
  (motorA, motorB) = (mA, mB);
tel
```

Proving with dependencies

$$\frac{}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ []}$$

$$\frac{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ l \qquad x \in V \qquad \neg \text{In}\ x\ l \qquad (\forall y,\ y \rightarrow_E^* x \implies \text{In}\ y\ l)}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ (x :: l)}$$

```
node drive_sequence(step : bool)
returns (motorA, motorB : bool)
var last mA^mA(l) : bool = true; last mB^mB(l) : bool = false;
let
  switch step
  | true do
     mA^mA(t) = not (last mB);
     mB^mB(t) = last mA;
  | false do (mA^mA(f), mB^mB(f)) = (last mA, last mB)
  end;
  (motorA, motorB) = (mA, mB);
tel
```



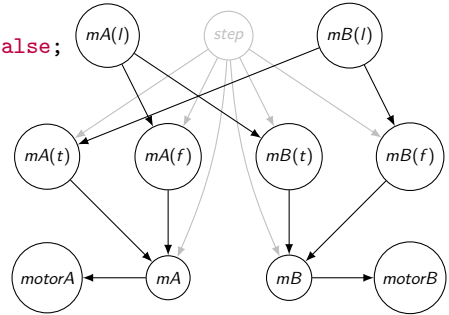
Proving with dependencies

$$\frac{}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ []}$$

$$\frac{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ l \qquad x \in V \qquad \neg \text{In}\ x\ l \qquad (\forall y,\ y \rightarrow_E^* x \implies \text{In}\ y\ l)}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ (x :: l)}$$

```
node drive_sequence(step : bool)
returns (motorA, motorB : bool)
var last mA^mA(l) : bool = true; last mB^mB(l) : bool = false;
let
  switch step
  | true do
     mA^mA(t) = not (last mB);
     mB^mB(t) = last mA;
  | false do (mA^mA(f), mB^mB(f)) = (last mA, last mB)
  end;
  (motorA, motorB) = (mA, mB);
tel
```

step

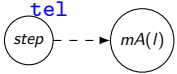


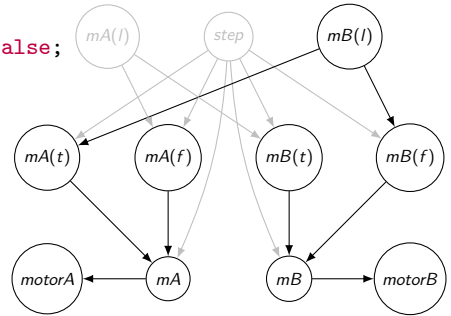
Proving with dependencies

$$\frac{}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ []}$$

$$\frac{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ l \qquad x \in V \qquad \neg \text{In}\ x\ l \qquad (\forall y,\ y \rightarrow_E^* x \implies \text{In}\ y\ l)}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ (x :: l)}$$

```
node drive_sequence(step : bool)
returns (motorA, motorB : bool)
var last mA^mA(l) : bool = true; last mB^mB(l) : bool = false;
let
  switch step
  | true do
     mA^mA(t) = not (last mB);
     mB^mB(t) = last mA;
  | false do (mA^mA(f), mB^mB(f)) = (last mA, last mB)
  end;
  (motorA, motorB) = (mA, mB);
tel
```



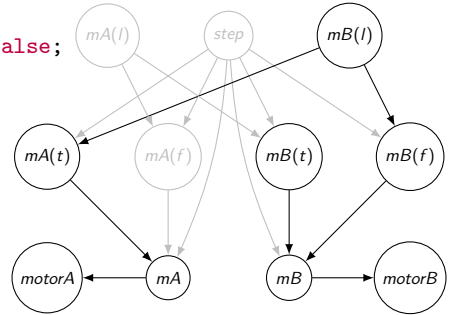


Proving with dependencies

$$\frac{}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ []}$$

$$\frac{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ l \qquad x \in V \qquad \neg \text{In}\ x\ l \qquad (\forall y,\ y \rightarrow_E^* x \implies \text{In}\ y\ l)}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ (x :: l)}$$

```
node drive_sequence(step : bool)
returns (motorA, motorB : bool)
var last mA^mA(l) : bool = true; last mB^mB(l) : bool = false;
let
  switch step
  | true do
     mA^mA(t) = not (last mB);
     mB^mB(t) = last mA;
  | false do (mA^mA(f), mB^mB(f)) = (last mA, last mB)
  end;
  (motorA, motorB) = (mA, mB);
tel
```

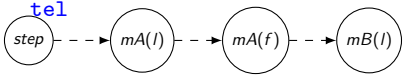


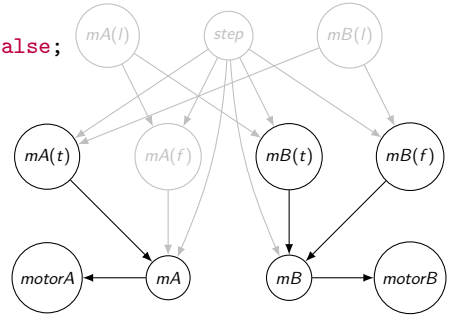
Proving with dependencies

$$\frac{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ l \qquad x \in V \qquad \neg \ln x\ l \qquad (\forall y,\ y \rightarrow_E^* x \implies \ln y\ l)}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ (x :: l)}$$

$$\frac{}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ []}$$

```
node drive_sequence(step : bool)
returns (motorA, motorB : bool)
var last mA^mA(l) : bool = true; last mB^mB(l) : bool = false;
let
  switch step
  | true do
     mA^mA(t) = not (last mB);
     mB^mB(t) = last mA;
  | false do (mA^mA(f), mB^mB(f)) = (last mA, last mB)
  end;
  (motorA, motorB) = (mA, mB);
tel
```



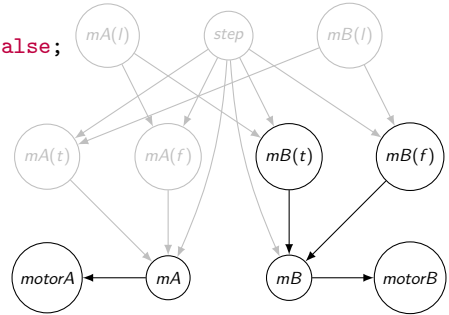


Proving with dependencies

$$\frac{}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ []}$$

$$\frac{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ l \qquad x \in V \qquad \neg \text{In}\ x\ l \qquad (\forall y,\ y \rightarrow_E^* x \implies \text{In}\ y\ l)}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ (x :: l)}$$

```
node drive_sequence(step : bool)
returns (motorA, motorB : bool)
var last mA^mA(l) : bool = true; last mB^mB(l) : bool = false;
let
  switch step
  | true do
     mA^mA(t) = not (last mB);
     mB^mB(t) = last mA;
  | false do (mA^mA(f), mB^mB(f)) = (last mA, last mB)
  end;
  (motorA, motorB) = (mA, mB);
tel
```

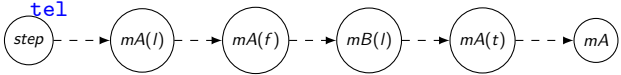


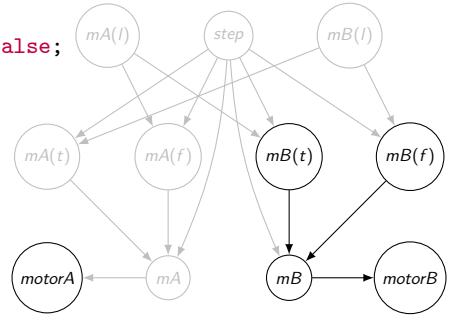
Proving with dependencies

$$\frac{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ l \qquad x \in V \qquad \neg \text{In}\ x\ l \qquad (\forall y,\ y \rightarrow_E^* x \implies \text{In}\ y\ l)}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ (x :: l)}$$

$$\frac{}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ []}$$

```
node drive_sequence(step : bool)
returns (motorA, motorB : bool)
var last mA^mA(l) : bool = true; last mB^mB(l) : bool = false;
let
  switch step
  | true do
     mA^mA(t) = not (last mB);
     mB^mB(t) = last mA;
  | false do (mA^mA(f), mB^mB(f)) = (last mA, last mB)
  end;
  (motorA, motorB) = (mA, mB);
tel
```



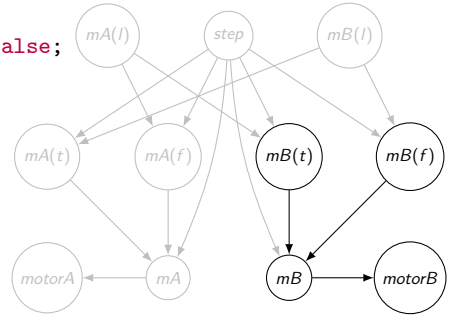


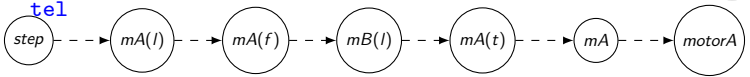
Proving with dependencies

$$\frac{}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ []}$$

$$\frac{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ l \qquad x \in V \qquad \neg \text{In}\ x\ l \qquad (\forall y,\ y \rightarrow_E^* x \implies \text{In}\ y\ l)}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ (x :: l)}$$

```
node drive_sequence(step : bool)
returns (motorA, motorB : bool)
var last mA^mA(l) : bool = true; last mB^mB(l) : bool = false;
let
  switch step
  | true do
     mA^mA(t) = not (last mB);
     mB^mB(t) = last mA;
  | false do (mA^mA(f), mB^mB(f)) = (last mA, last mB)
  end;
  (motorA, motorB) = (mA, mB);
tel
```



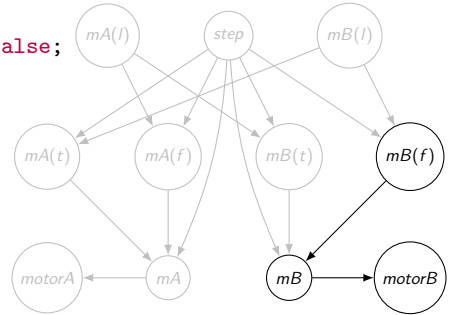


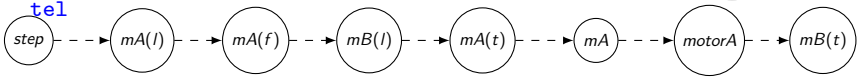
Proving with dependencies

$$\frac{}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ []}$$

$$\frac{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ l \qquad x \in V \qquad \neg \text{In}\ x\ l \qquad (\forall y,\ y \rightarrow_E^* x \implies \text{In}\ y\ l)}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ (x :: l)}$$

```
node drive_sequence(step : bool)
returns (motorA, motorB : bool)
var last mA^mA(l) : bool = true; last mB^mB(l) : bool = false;
let
  switch step
  | true do
     mA^mA(t) = not (last mB);
     mB^mB(t) = last mA;
  | false do (mA^mA(f), mB^mB(f)) = (last mA, last mB)
  end;
  (motorA, motorB) = (mA, mB);
tel
```



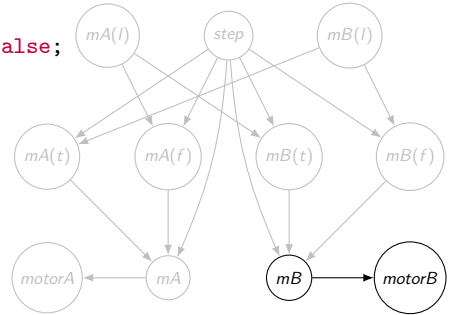


Proving with dependencies

$$\frac{}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ []}$$

$$\frac{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ l \qquad x \in V \qquad \neg \text{In}\ x\ l \qquad (\forall y,\ y \rightarrow_E^* x \implies \text{In}\ y\ l)}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ (x :: l)}$$

```
node drive_sequence(step : bool)
returns (motorA, motorB : bool)
var last mA^mA(l) : bool = true; last mB^mB(l) : bool = false;
let
  switch step
  | true do
     mA^mA(t) = not (last mB);
     mB^mB(t) = last mA;
  | false do (mA^mA(f), mB^mB(f)) = (last mA, last mB)
  end;
  (motorA, motorB) = (mA, mB);
tel
```

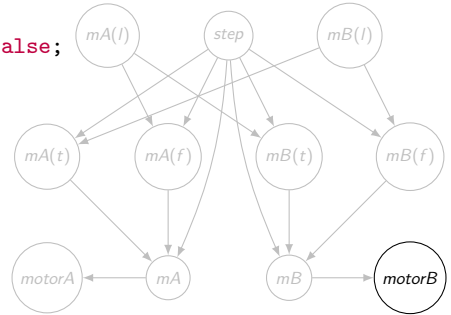


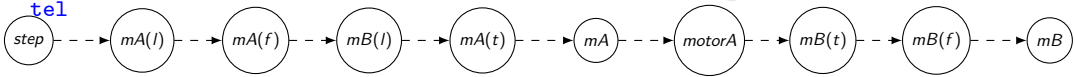
Proving with dependencies

$$\frac{}{\mathsf{TopoOrder}\ (\mathsf{AcyGraph}\ V\ E)\ []}$$

$$\frac{\mathsf{TopoOrder}\ (\mathsf{AcyGraph}\ V\ E)\ l \qquad x \in V \qquad \neg \mathsf{In}\ x\ l \qquad (\forall y,\ y \rightarrow_E^* x \implies \mathsf{In}\ y\ l)}{\mathsf{TopoOrder}\ (\mathsf{AcyGraph}\ V\ E)\ (x :: l)}$$

```
node drive_sequence(step : bool)
returns (motorA, motorB : bool)
var last mA^mA(l) : bool = true; last mB^mB(l) : bool = false;
let
  switch step
  | true do
     mA^mA(t) = not (last mB);
     mB^mB(t) = last mA;
  | false do (mA^mA(f), mB^mB(f)) = (last mA, last mB)
  end;
  (motorA, motorB) = (mA, mB);
tel
```



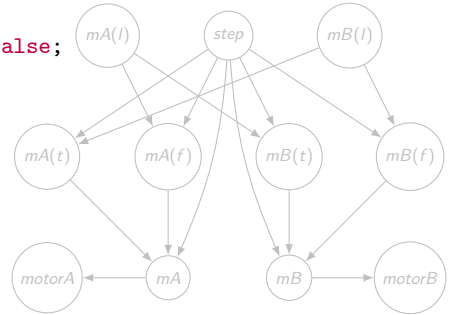


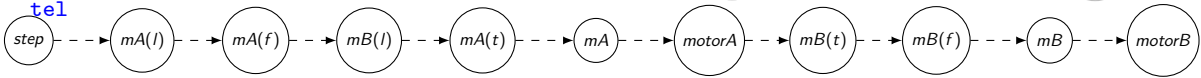
Proving with dependencies

$$\frac{}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ []}$$

$$\frac{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ l \qquad x \in V \qquad \neg \text{In}\ x\ l \qquad (\forall y,\ y \rightarrow_E^* x \implies \text{In}\ y\ l)}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ (x :: l)}$$

```
node drive_sequence(step : bool)
returns (motorA, motorB : bool)
var last mA^mA(l) : bool = true; last mB^mB(l) : bool = false;
let
  switch step
  | true do
     mA^mA(t) = not (last mB);
     mB^mB(t) = last mA;
  | false do (mA^mA(f), mB^mB(f)) = (last mA, last mB)
  end;
  (motorA, motorB) = (mA, mB);
tel
```





Proving with dependencies

$$\frac{}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ []} \qquad \frac{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ l \qquad x \in V \qquad \neg \text{In}\ x\ l \qquad (\forall y,\ y \rightarrow_E^* x \implies \text{In}\ y\ l)}{\text{TopoOrder}\ (\text{AcyGraph}\ V\ E)\ (x :: l)}$$

Used to prove:

- Determinism of the semantics:
  if $G \vdash f(xs) \Downarrow ys_1$ **and** $G \vdash f(xs) \Downarrow ys_2$ **then** $ys_1 \equiv ys_2$
- Correctness of the clock system:
  if $\Gamma \vdash e : ck$ **and** $G, H, bs \vdash e \Downarrow vs$ **then** $H, bs \vdash ck \Downarrow (\text{abstract-clock}\ vs)$

Prototype Implementation

	Files	Spec.	Code	Proofs	Admin.	Total	Diff.
Lexing/Parsing (Menhir)	3	0	987	0	0	987	+200
Elaboration	1	180	1 215	309	56	1 760	+442
Lustre	12	5 844	847	10 592	601	17 884	+9 303
Shared Variables	5	417	79	1 499	221	2 216	n/a
State Machines	5	255	114	1 652	216	2 237	n/a
Switch Blocks	9	729	117	2 007	390	3 243	n/a
Local Scopes	5	413	50	1 285	214	1 962	n/a
Normalization	8	2 355	325	6 965	447	10 092	-1 722
Transcription	8	749	236	2 701	442	4 128	+1 250
NLustre $\rightsquigarrow$ Clight	71	9 432	1 777	18 928	3 037	33 174	+4 728
Common definitions & driver	27	6 659	611	10 451	1 322	19 043	+4 293
Total	154	27 033	6 358	56 389	6 946	96 726	+28 152

Conclusion

What we have:

- An end-to-end verified compiler for a Dataflow-Synchronous Language with State Machines
- A relational semantics for the language
- A generic approach to prove complex properties of the semantics

Conclusion

What we have:

- An end-to-end verified compiler for a Dataflow-Synchronous Language with State Machines
- A relational semantics for the language
- A generic approach to prove complex properties of the semantics

What we would want:

- Shorter proofs : automate or simplify definitions ?
- Some additional flexibility for the programmer: `last` on outputs, default completion
- Discuss other semantic models (see the work of Paul this afternoon) and their applicability to proving program transformations

References

Bourke, T., P. Jeanmaire, B. Pesin, and M. Pouzet (Oct. 2021). "Verified Lustre Normalization with Node Subsampling". In: ACM Trans. Embedded Computing Systems 20.5s, Article 98.

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.

Halbwachs, N., P. Caspi, P. Raymond, and D. Pilaud (Sept. 1991). "The synchronous dataflow programming language LUSTRE". In: Proc. IEEE 79.9, pp. 1305–1320.

Vélus with Activation Blocks - Syntax

$$
\begin{array}{rcl}
e & ::= & c \mid C \mid x \mid \texttt{last}\ x \\
& \mid & \diamond\ e \mid e \oplus e \\
& \mid & e^+\ \texttt{fby}\ e^+ \mid e^+\ \texttt{->}\ e^+ \\
& \mid & e^+\ \texttt{when}\ C\ (\ x\ ) \\
& \mid & \texttt{merge}\ x\ (\ C\ \texttt{=>}\ e^+\ )^+ \\
& \mid & \texttt{case}\ e\ \texttt{of}\ (\ C\ \texttt{=>}\ e^+\ )^+ \\
& \mid & f\ (\ e^+\ ) \mid (\ \texttt{reset}\ f\ \texttt{every}\ e\ )\ (\ e^+\ )
\end{array}
$$

$$td ::= \texttt{type}\ tx = C^+$$

$$d ::= x^{ck}_{ty}$$

$$n ::= \texttt{node}\ f\ (\ d^+\ )\ \texttt{returns}\ (\ d^+\ )\ blk$$

$$G ::= td^*\ n^+$$

$$
\begin{array}{rcl}
blk & ::= & x^+ = e^+\ ; \\
& \mid & \texttt{var}\ loc^*\ \texttt{let}\ blk^+\ \texttt{tel} \\
& \mid & \texttt{reset}\ blk^+\ \texttt{every}\ e \\
& \mid & \texttt{switch}\ e\ (\ C\ \texttt{do}\ blk^+\ )^+\ \texttt{end} \\
& \mid & \texttt{automaton initially}\ autinits \\
& & (\texttt{state}\ C\ autscope\ )^+\ \texttt{end} \\
& \mid & \texttt{automaton initially}\ C \\
& & (\texttt{state}\ C\ \texttt{do}\ blk^+\ \texttt{unless}\ trans^+\ )^+\ \texttt{end}
\end{array}
$$

$$loc ::= d \mid \texttt{last}\ d = e$$

$$autinits ::= C \mid \texttt{if}\ e\ \texttt{then}\ C\ \texttt{else}\ autinits$$

$$autscope ::= \texttt{var}\ loc^*\ \texttt{do}\ blk^+\ \texttt{until}\ trans^+$$

$$trans ::= \mid e\ \texttt{continue}\ C \mid \mid e\ \texttt{then}\ C$$

Local Scopes - Semantic rules

$$\frac{H(\text{last}\ x) \equiv vs}{G, H, bs \vdash \texttt{last}\ x \Downarrow [vs]}$$

$$\frac{\begin{array}{c}\forall x,\ x \in H' \iff x \in locs \\ \forall x\ e,\ (\texttt{last}\ x = e) \in locs \implies G, H + H', bs \vdash \texttt{last}\ x = e \qquad G, H + H', bs \vdash blks\end{array}}{G, H, bs \vdash \texttt{var}\ locs\ \texttt{let}\ blks\ \texttt{tel}}$$

Local Scopes - Semantic rules

$$\frac{H(\text{last } x) \equiv vs}{G, H, bs \vdash \texttt{last}\, x \Downarrow [vs]}$$

$$\frac{\begin{array}{c}\forall x,\ x \in H' \iff x \in locs \\ \forall x\ e,\ (\texttt{last}\, x = e) \in locs \implies G, H + H', bs \vdash \texttt{last}\, x = e \qquad G, H + H', bs \vdash blks\end{array}}{G, H, bs \vdash \texttt{var}\ locs\ \texttt{let}\ blks\ \texttt{tel}}$$

$$\frac{G, H + H', bs \vdash e \Downarrow [vs_0] \qquad H'(x) \equiv vs_1 \qquad H'(\text{last } x) \equiv \text{fby } vs_0\ vs_1}{G, H + H', bs \vdash \texttt{last}\, x = e}$$

$$(H_1 + H_2)(x) = \begin{cases} H_2(x) & \text{if } x \in H_2 \\ H_1(x) & \text{otherwise.} \end{cases}$$

Switch - Semantic rules

$$
\begin{array}{lcl}
\mathsf{when}^C\ (\langle\rangle \cdot xs)\ (\langle\rangle \cdot ys) & \equiv & \langle\rangle \cdot \mathsf{when}^C\ xs\ ys \\
\mathsf{when}^C\ (\langle v\rangle \cdot xs)\ (\langle C\rangle \cdot ys) & \equiv & \langle v\rangle \cdot \mathsf{when}^C\ xs\ ys \\
\mathsf{when}^C\ (\langle v\rangle \cdot xs)\ (\langle C'\rangle \cdot ys) & \equiv & \langle\rangle \cdot \mathsf{when}^C\ xs\ ys
\end{array}
$$

$$
(\mathsf{when}^C\ H\ cs)(x) \equiv \mathsf{when}^C\ (H(x))\ cs
$$

$$
\frac{G, H, bs \vdash e \Downarrow [cs] \qquad \forall i,\ G, \mathsf{when}^{C_i}\ (H, bs)\ cs \vdash blks_i}{G, H, bs \vdash \texttt{switch}\ e\ \overline{(C_i\ \texttt{do}\ blks_i)}^{\,i}\ \texttt{end}}
$$

Reset - Semantics rules

$$\begin{array}{lcl} \mathsf{mask}_{k'}^{k}\ (\mathtt{F}\cdot rs)\ (sv\cdot xs) & \equiv & (\text{if } k'=k \text{ then } sv \text{ else } \langle\rangle)\cdot \mathsf{mask}_{k'}^{k}\ rs\ xs \\ \mathsf{mask}_{k'}^{k}\ (\mathtt{T}\cdot rs)\ (sv\cdot xs) & \equiv & (\text{if } k'+1=k \text{ then } sv \text{ else } \langle\rangle)\cdot \mathsf{mask}_{k'+1}^{k}\ rs\ xs \end{array}$$

$$\frac{\begin{array}{c} G, H, bs \vdash es \Downarrow xss \\ G, H, bs \vdash e \Downarrow [ys] \qquad \text{bools-of } ys \equiv rs \\ \forall k,\ G \vdash f(\mathsf{mask}_0^k\ rs\ xss) \Downarrow (\mathsf{mask}_0^k\ rs\ yss) \end{array}}{G, H, bs \vdash (\ \mathtt{reset}\ f\ \mathtt{every}\ e)(es) \Downarrow yss}$$

$$\frac{\begin{array}{c} G, H, bs \vdash e \Downarrow [ys] \qquad \text{bools-of } ys \equiv rs \\ \forall k,\ G, \mathsf{mask}_0^k\ rs\ (H, bs) \vdash blks \end{array}}{G, H, bs \vdash \mathtt{reset}\ blks\ \mathtt{every}\ e}$$

Hierarchical State Machines - Semantic rules

$$\frac{\begin{array}{c} H, bs \vdash ck \Downarrow bs' \qquad G, H, bs' \vdash autinits \Downarrow sts_0 \\ \text{fby } sts_0\ sts_1 \equiv sts \qquad \forall i,\ \forall k,\ G, (\text{select}_0^{C_i,k}\ sts\ (H, bs)), C_i \vdash autscope_i \Downarrow (\text{select}_0^{C_i,k}\ sts\ sts_1) \end{array}}{G, H, bs \vdash \texttt{automaton initially}\ autinits^{ck}\ \overline{(\texttt{state}\ C_i\ autscope_i)}^{\,i}\ \texttt{end}}$$

$$\frac{\begin{array}{c} \forall x,\ x \in H' \iff x \in locs \qquad \forall x\ e,\ (\texttt{last}\ x = e) \in locs \implies G, H + H', bs \vdash \texttt{last}\ x = e \\ G, H + H', bs \vdash blks \qquad G, H + H', bs, C_i \vdash trans \Downarrow sts \end{array}}{G, H, bs, C_i \vdash \texttt{var}\ locs\ \texttt{do}\ blks\ \texttt{until}\ trans \Downarrow sts}$$

$$\frac{\begin{array}{c} H, bs \vdash ck \Downarrow bs' \qquad \text{fby } (\text{const } bs'\ (C, \texttt{F}))\ sts_1 \equiv sts \\ \forall i,\ \forall k,\ G, (\text{select}_0^{C_i,k}\ sts\ (H, bs)), C_i \vdash trans_i \Downarrow (\text{select}_0^{C_i,k}\ sts\ sts_1) \\ \forall i,\ \forall k,\ G, (\text{select}_0^{C_i,k}\ sts_1\ (H, bs)) \vdash blks_i \end{array}}{G, H, bs \vdash \texttt{automaton initially}\ C^{ck}\ \overline{(\texttt{state}\ C_i\ \texttt{do}\ blks_i\ \texttt{unless}\ trans_i)}^{\,i}\ \texttt{end}}$$

Hierarchical State Machines - Transitions

$$\frac{G, H, bs \vdash e \Downarrow [ys] \qquad \text{bools-of } ys \equiv bs' \qquad G, H, bs \vdash autinits \Downarrow sts' \qquad sts \equiv \text{first-of } (\text{const } bs' \ (C, \mathtt{F}))\ sts'}{G, H, bs \vdash C\ \texttt{if}\ e; autinits \Downarrow sts}$$

$$\frac{sts \equiv \text{const } bs\ (C, \mathtt{F})}{G, H, bs \vdash \texttt{otherwise}\ C \Downarrow sts}$$

$$\begin{aligned} \text{first-of } (\langle C, b \rangle \cdot sts_1)\ (st_2 \cdot sts_2) &\equiv \langle C, b \rangle \cdot \text{first-of } sts_1\ sts_2 \\ \text{first-of } (\langle\rangle \cdot sts_1)\ (st_2 \cdot sts_2) &\equiv st_2 \cdot \text{first-of } sts_1\ sts_2 \end{aligned}$$

$$\frac{sts \equiv \text{const } bs\ (C_i, \mathtt{F})}{G, H, bs, C_i \vdash \epsilon \Downarrow sts}$$

$$\frac{G, H, bs \vdash e \Downarrow [ys] \qquad \text{bools-of } ys \equiv bs' \qquad G, H, bs, C_i \vdash trans \Downarrow sts' \qquad sts \equiv \text{first-of } (\text{const } bs' \ (C, \mathtt{F}))\ sts'}{G, H, bs, C_i \vdash |e\ \texttt{continue}\ C\ trans \Downarrow sts}$$

$$\frac{G, H, bs \vdash e \Downarrow [ys] \qquad \text{bools-of } ys \equiv bs' \qquad G, H, bs, C_i \vdash trans \Downarrow sts' \qquad sts \equiv \text{first-of } (\text{const } bs' \ (C, \mathtt{T}))\ sts'}{G, H, bs, C_i \vdash |e\ \texttt{then}\ C\ trans \Downarrow sts}$$