Mechanized Semantics and Verified Compilation for a Dataflow Synchronous Language with Reset

PhD thesis defense

Lélio Brun[1,2]

July 6, 2020

[1]Inria Paris – PARKAS Team

[2]École normale supérieure – PSL University

Context

Embedded Systems Design

Embedded systems

- computer systems within physical systems that interact with the real world, often with real-time constraints
- software usually written in low-level languages: C, Ada, Assembly

Embedded Systems Design

Embedded systems

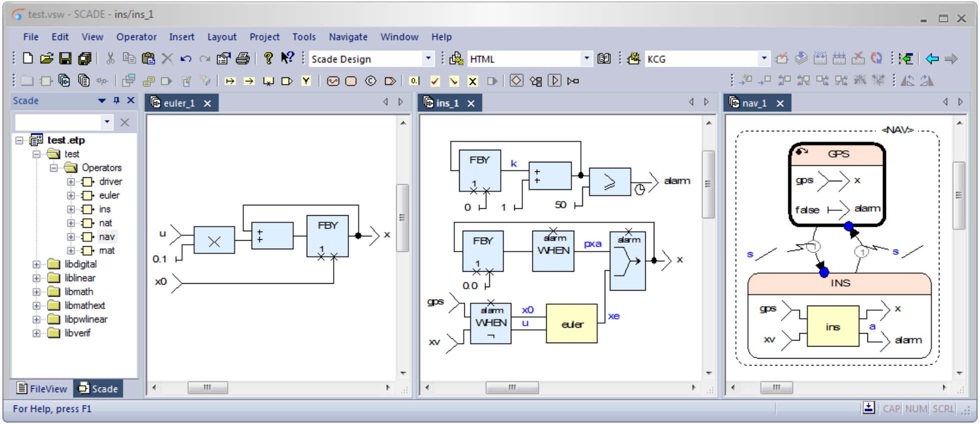
- computer systems within physical systems that interact with the real world, often with real-time constraints
- software usually written in low-level languages: C, Ada, Assembly

Model-Based Design

Executable high-level abstract specifications

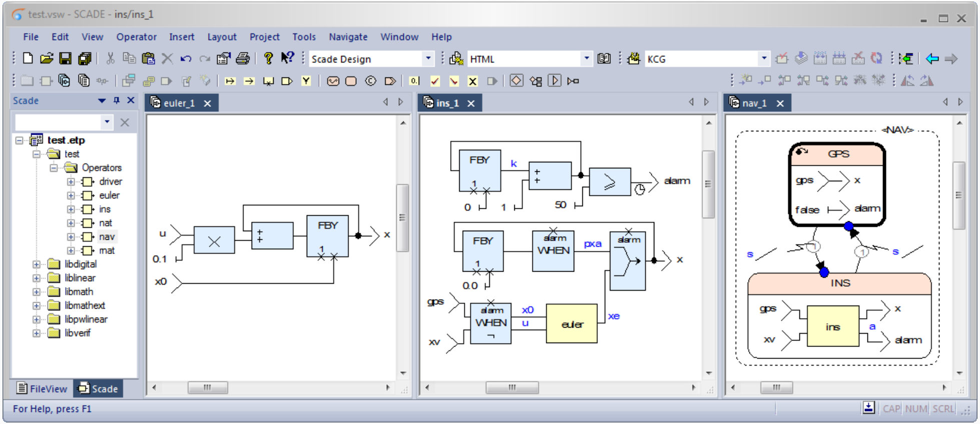
Model-Based Design in SCADE Suite

www.ansys.com/products/embedded-software/ansys-scade-suite



Model-Based Design in SCADE Suite

www.ansys.com/products/embedded-software/ansys-scade-suite

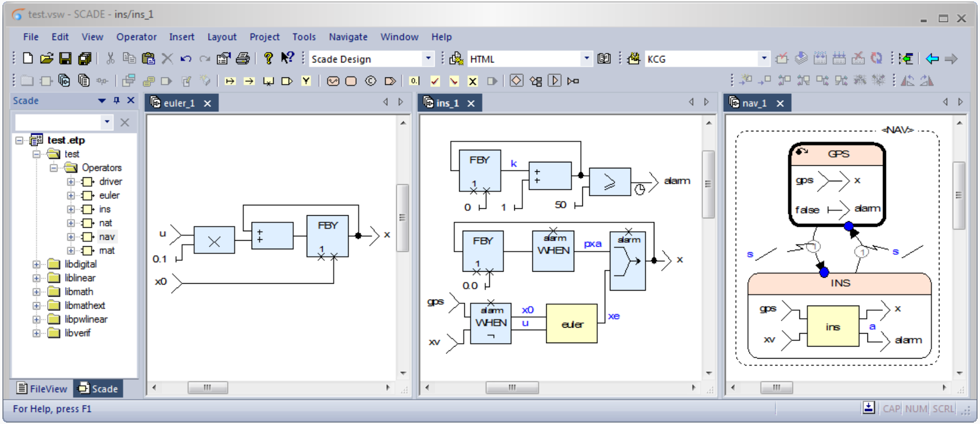


block / node = system

line = signal

Model-Based Design in SCADE Suite

www.ansys.com/products/embedded-software/ansys-scade-suite

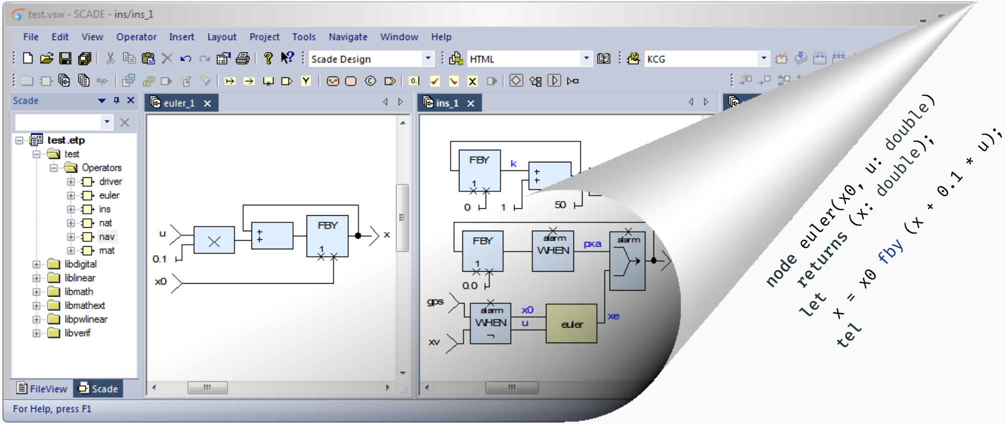


block / node = system = stream function

line = signal = stream of values

Model-Based Design in SCADE Suite

www.ansys.com/products/embedded-software/ansys-scade-suite

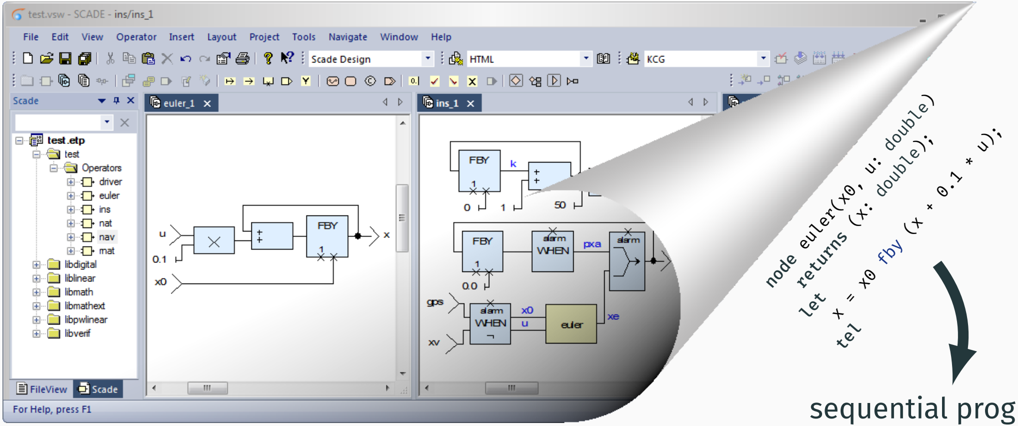


block / node = system = stream function

line = signal = stream of values

Model-Based Design in SCADE Suite

www.ansys.com/products/embedded-software/ansys-scade-suite



sequential program

(C, Ada, assembly)

block / node = system = stream function

line = signal = stream of values

Criticality

Systems that must not fail

- Flight control systems
- Automated train systems
- Power plant monitoring software

Systems that must not fail

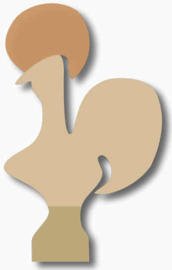
- Flight control systems
- Automated train systems
- Power plant monitoring software

State of the art: industrial certification of the development process, sometimes using *formal methods*, eg. SCADE

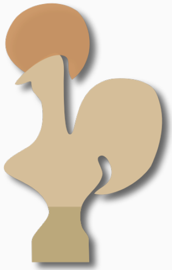
Scientific question: can we mechanize the formal definitions and produce an end-to-end correctness proof?

Interactive Theorem Provers, or Proof Assistants

- Software tools to assist statement of theorems, and development and checking of their proofs
- Mizar, Isabelle, HOL, Coq, ACL2, PVS, Agda, …



Klein et al. 2009
Kumar, Myreen, Norrish, and Owens 2014
Leroy 2009



Interactive Theorem Provers, or Proof Assistants

- Software tools to assist statement of theorems, and development and checking of their proofs
- Mizar, Isabelle, HOL, Coq, ACL2, PVS, Agda, …

Existing mechanized formalizations

seL4: a verified micro-kernel in Isabelle

CakeML: a verified compiler for a functional language in HOL

CompCert: a milestone

Formal mechanization in Coq of the C language and of the correctness proof of its compilation to Assembly code.

Caspi, Pilaud, Halbwachs, and Plaice 1987
Halbwachs, Caspi, Raymond, and Pilaud 1991
Colaço, Pagano, and Pouzet 2017

Model-Based Design Languages
Scade 6, Lustre

+

Interactive Theorem Provers
Coq

Challenges

1. Mechanize the semantics
2. Prove the compilation algorithms correct

Caspi, Pilaud, Halbwachs, and Plaice 1987
Halbwachs, Caspi, Raymond, and Pilaud 1991
Colaço, Pagano, and Pouzet 2017

Model-Based Design Languages
Scade 6, Lustre

+

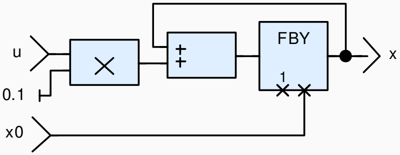
Interactive Theorem Provers
Coq

Challenges

1. Mechanize the semantics
2. Prove the compilation algorithms correct

Focus: modular reset

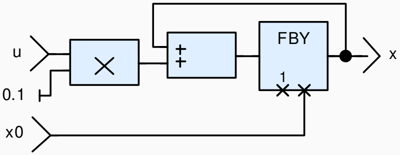
Example




```
node euler(x0, u: double)
  returns (x: double);
let
  x = x0 fby (x + 0.1 * u);
tel
```

x_0	0.00	1.55	3.62	5.46	$\cdots$
u	15.00	20.00	17.00	12.00	$\cdots$
$x + 0.1 \times u$	1.50	3.50	5.20	6.70	$\cdots$
x	0.00	1.50	3.50	5.20	$\cdots$

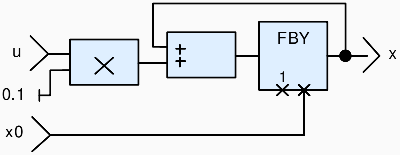
Example




```
node euler(x0, u: double)
  returns (x: double);
let
  x = x0 fby (x + 0.1 * u);
tel
```

x_0	0.00	1.55	3.62	5.46	$\cdots$
u	15.00	20.00	17.00	12.00	$\cdots$
$x + 0.1 \times u$	1.50	3.50	5.20	6.70	$\cdots$
x	0.00	1.50	3.50	5.20	$\cdots$

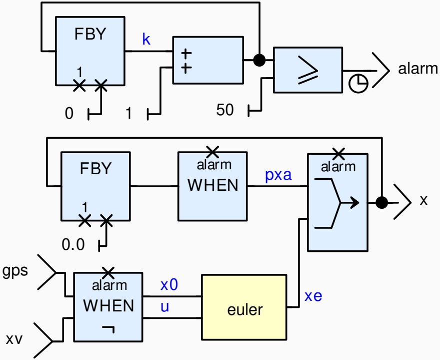
Example




```
node euler(x0, u: double)
  returns (x: double);
let
  x = x0 fby (x + 0.1 * u);
tel
```

x_0	0.00	1.55	3.62	5.46	$\cdots$
u	15.00	20.00	17.00	12.00	$\cdots$
$x + 0.1 \times u$	1.50	3.50	5.20	6.70	$\cdots$
x	0.00	1.50	3.50	5.20	$\cdots$

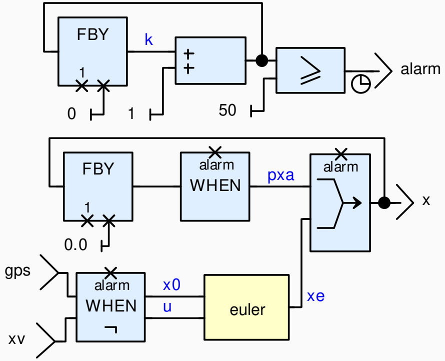
Example




```
node ins(gps, xv: double)
  returns (x: double, alarm: bool)
  var pxa, xe: double; k: int;
let
  k = 0 fby (k + 1);
  alarm = (k >= 50);
  xe = euler((gps, xv) when not alarm);
  pxa = (0. fby x) when alarm;
  x = merge alarm pxa xe;
tel
```

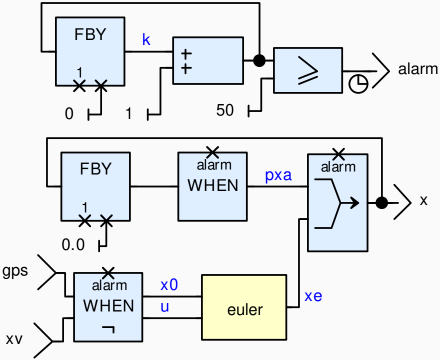
gps	0.00	1.55	3.62	5.46	···	86.52	88.40	90.91	···
xv	15.00	20.00	17.00	12.00	···	18.00	23.00	20.00	···
k	0	1	2	3	···	49	50	51	···
alarm	F	F	F	F	···	F	T	T	···
xe	0.00	1.50	3.50	5.20	···	77.35			···
pxa					···		77.35	77.35	···
x	0.00	1.50	3.50	5.20	···	77.35	77.35	77.35	···

Example




```
node ins(gps, xv: double)
  returns (x: double, alarm: bool)
  var pxa, xe: double; k: int;
let
  k = 0 fby (k + 1);
  alarm = (k >= 50);
  xe = euler((gps, xv) when not alarm);
  pxa = (0. fby x) when alarm;
  x = merge alarm pxa xe;
tel
```

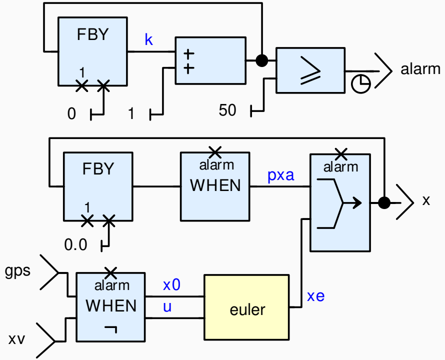
gps	0.00	1.55	3.62	5.46	···	86.52	88.40	90.91	···
xv	15.00	20.00	17.00	12.00	···	18.00	23.00	20.00	···
k	0	1	2	3	···	49	50	51	···
alarm	F	F	F	F	···	F	T	T	···
xe	0.00	1.50	3.50	5.20	···	77.35			···
pxa					···		77.35	77.35	···
x	0.00	1.50	3.50	5.20	···	77.35	77.35	77.35	···




```
node ins(gps, xv: double)
  returns (x: double, alarm: bool)
  var pxa, xe: double; k: int;
let
  k = 0 fby (k + 1);
  alarm = (k >= 50);
  xe = euler((gps, xv) when not alarm);
  pxa = (0. fby x) when alarm;
  x = merge alarm pxa xe;
tel
```

gps	0.00	1.55	3.62	5.46	...	86.52	88.40	90.91	...
xv	15.00	20.00	17.00	12.00	...	18.00	23.00	20.00	...
k	0	1	2	3	...	49	50	51	...
alarm	F	F	F	F	...	F	T	T	...
xe	0.00	1.50	3.50	5.20	...	77.35			...
pxa					...		77.35	77.35	...
x	0.00	1.50	3.50	5.20	...	77.35	77.35	77.35	...

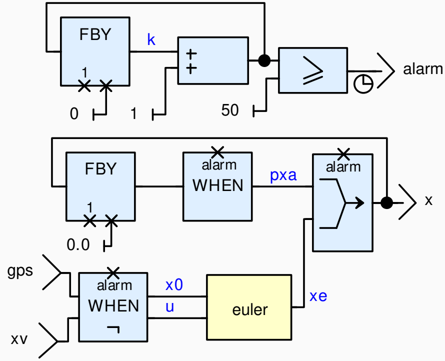
Example




```
node ins(gps, xv: double)
  returns (x: double, alarm: bool)
  var pxa, xe: double; k: int;
let
  k = 0 fby (k + 1);
  alarm = (k >= 50);
  xe = euler((gps, xv) when not alarm);
  pxa = (0. fby x) when alarm;
  x = merge alarm pxa xe;
tel
```

gps	0.00	1.55	3.62	5.46	⋯	86.52	88.40	90.91	⋯
xv	15.00	20.00	17.00	12.00	⋯	18.00	23.00	20.00	⋯
k	0	1	2	3	⋯	49	50	51	⋯
alarm	F	F	F	F	⋯	F	T	T	⋯
xe	0.00	1.50	3.50	5.20	⋯	77.35			⋯
pxa					⋯		77.35	77.35	⋯
x	0.00	1.50	3.50	5.20	⋯	77.35	77.35	77.35	⋯

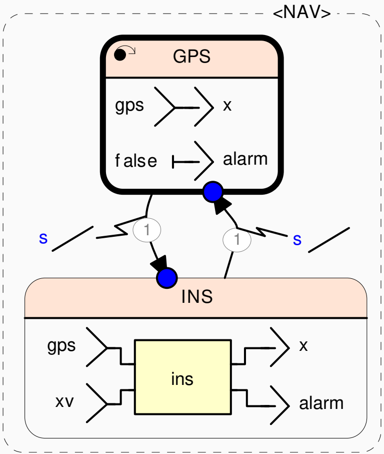
Example




```
node ins(gps, xv: double)
  returns (x: double, alarm: bool)
  var pxa, xe: double; k: int;
let
  x = merge alarm pxa xe;
  k = 0 fby (k + 1);
  pxa = (0. fby x) when alarm;
  xe = euler((gps, xv) when not alarm);
  alarm = (k >= 50);
tel
```

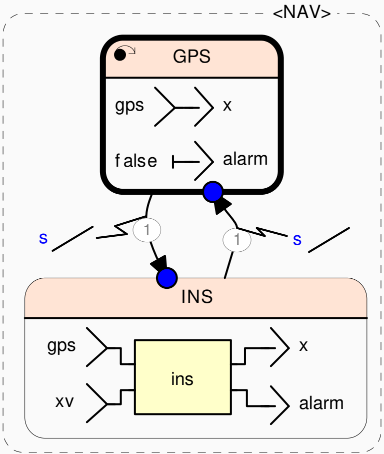
gps	0.00	1.55	3.62	5.46	⋯	86.52	88.40	90.91	⋯
xv	15.00	20.00	17.00	12.00	⋯	18.00	23.00	20.00	⋯
k	0	1	2	3	⋯	49	50	51	⋯
alarm	F	F	F	F	⋯	F	T	T	⋯
xe	0.00	1.50	3.50	5.20	⋯	77.35			⋯
pxa					⋯		77.35	77.35	⋯
x	0.00	1.50	3.50	5.20	⋯	77.35	77.35	77.35	⋯

Example




```
node nav(gps: double, xv: double, s: bool)
  returns (x: double, alarm: bool)
  var r: bool, c: bool;
let
  (x, alarm) = merge c
                 (gps when c, false)
                 ((restart ins every r)
                    ((gps, xv) whenot c));
  c = true fby (merge c (not s when c)
                        (s whenot c));
  r = false fby (s and c);
tel
```

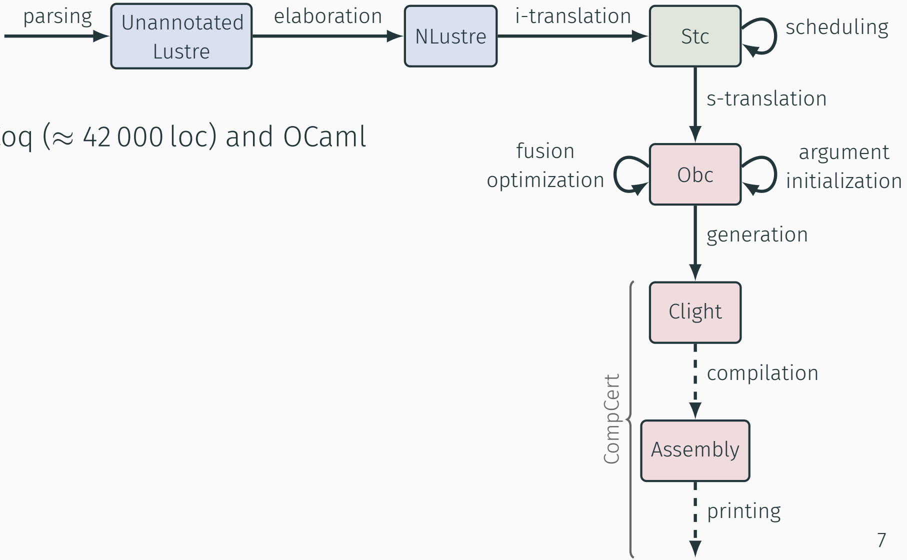
Example




```
node nav(gps: double, xv: double, s: bool)
  returns (x: double, alarm: bool)
  var r: bool, c: bool;
let
  (x, alarm) = merge c
                   (gps when c, false)
                   ((restart ins every r)
                       ((gps, xv) whenot c));
  c = true fby (merge c (not s when c)
                          (s whenot c));
  r = false fby (s and c);
tel
```

We need a way to reset the state of a node

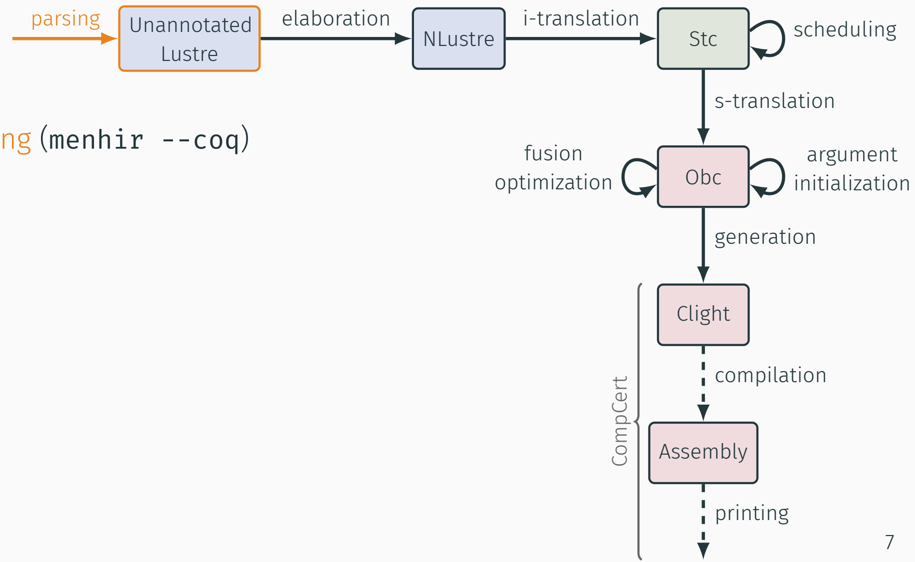
Biernacki, Colaço, Hamon, and Pouzet 2008
Bourke, Brun, Dagand, Leroy, Pouzet, and Rieg 2017
Bourke, Brun, and Pouzet 2020



Implemented in Coq (≈ 42 000 loc) and OCaml

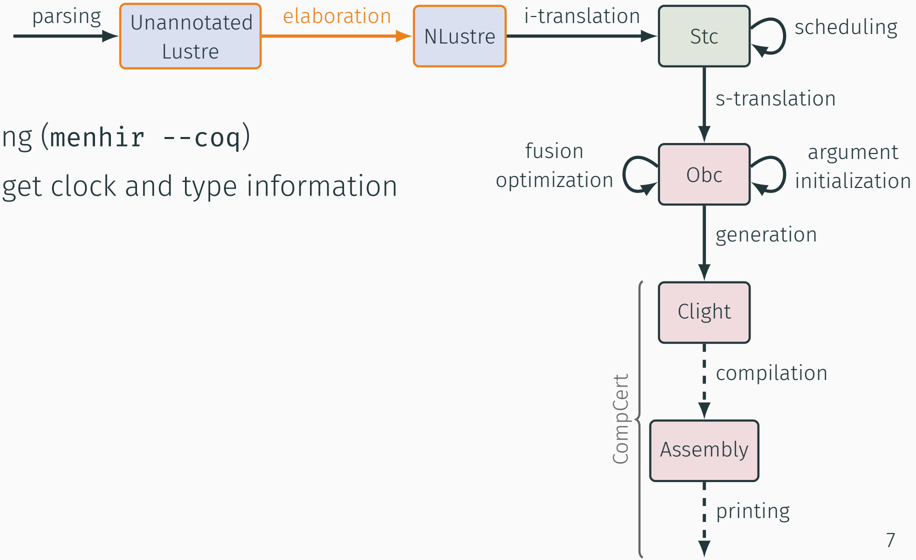
Vélus: a Verified Lustre Compiler

Jourdan, Pottier, and Leroy 2012



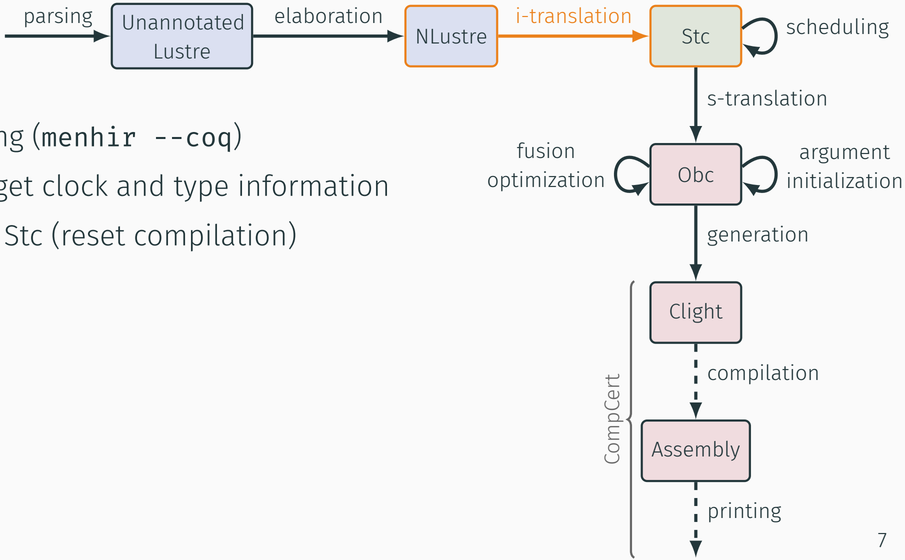
- validated parsing (`menhir --coq`)

Vélus: a Verified Lustre Compiler



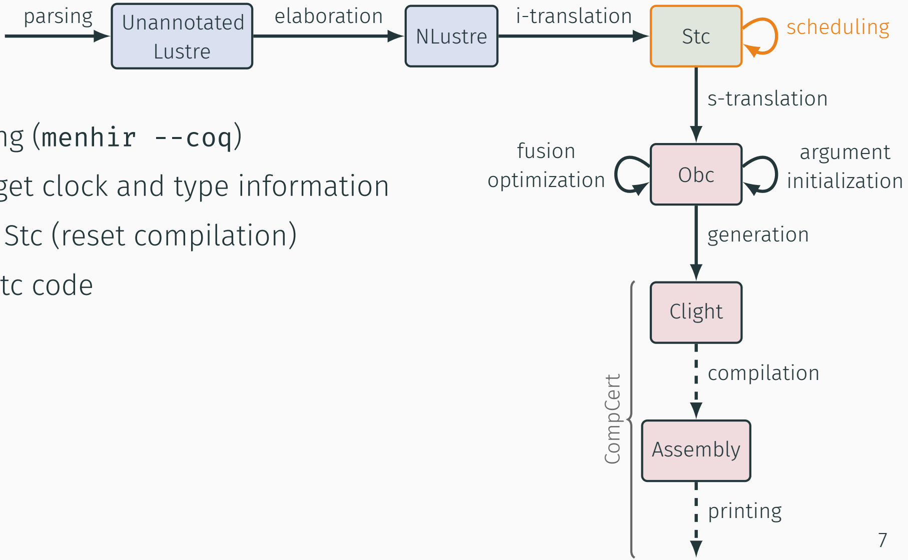
- validated parsing (`menhir --coq`)
- elaboration to get clock and type information

Vélus: a Verified Lustre Compiler



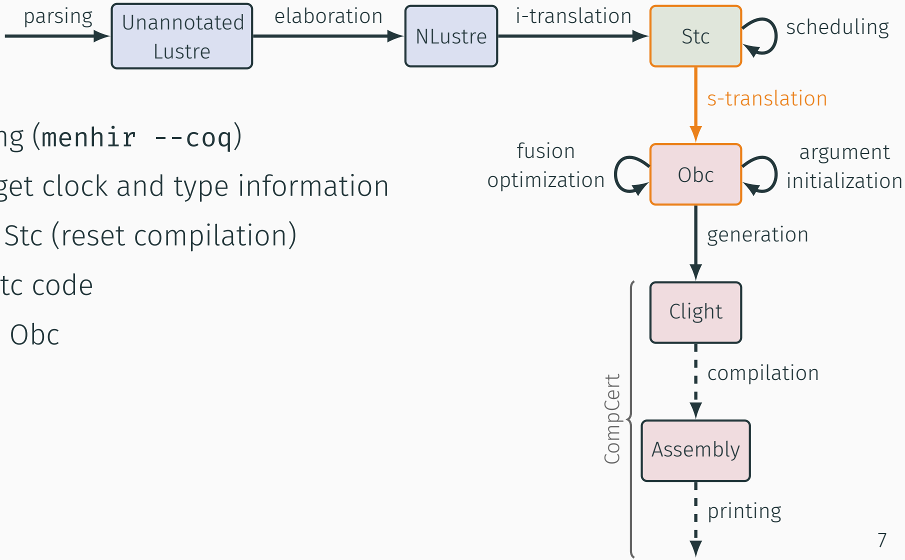
- validated parsing (`menhir --coq`)
- elaboration to get clock and type information
- i-translation to Stc (reset compilation)

Vélus: a Verified Lustre Compiler



- validated parsing (`menhir --coq`)
- elaboration to get clock and type information
- i-translation to Stc (reset compilation)
- scheduling of Stc code

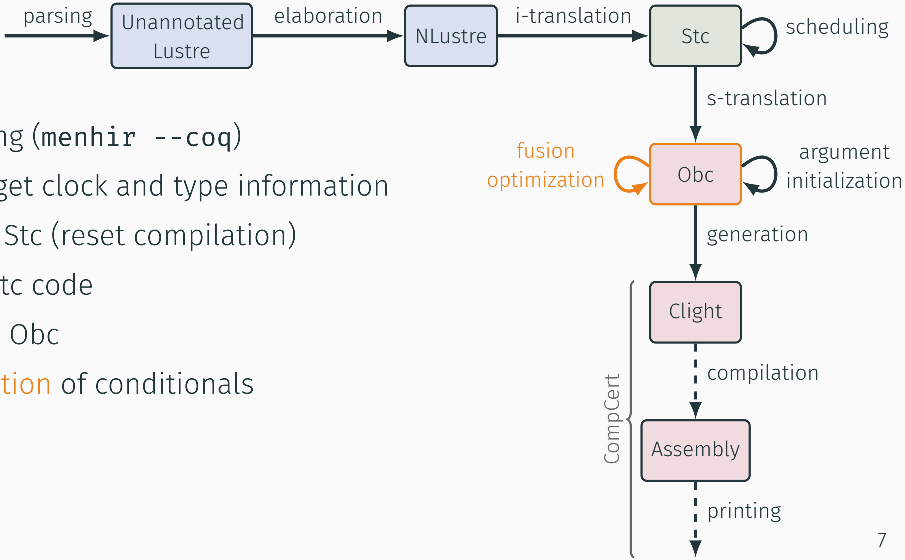
Vélus: a Verified Lustre Compiler



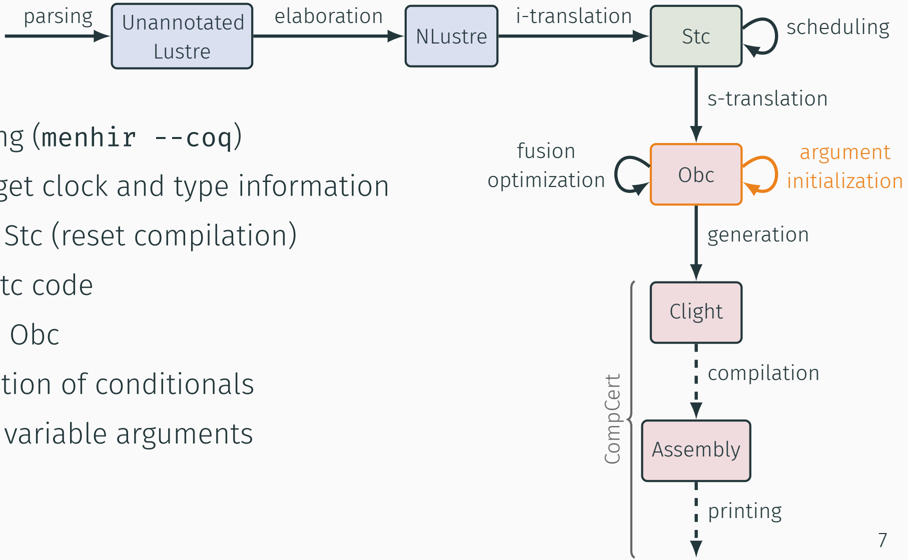
- validated parsing (`menhir --coq`)
- elaboration to get clock and type information
- i-translation to Stc (reset compilation)
- scheduling of Stc code
- s-translation to Obc

Vélus: a Verified Lustre Compiler

- validated parsing (`menhir --coq`)
- elaboration to get clock and type information
- i-translation to Stc (reset compilation)
- scheduling of Stc code
- s-translation to Obc
- fusion optimization of conditionals

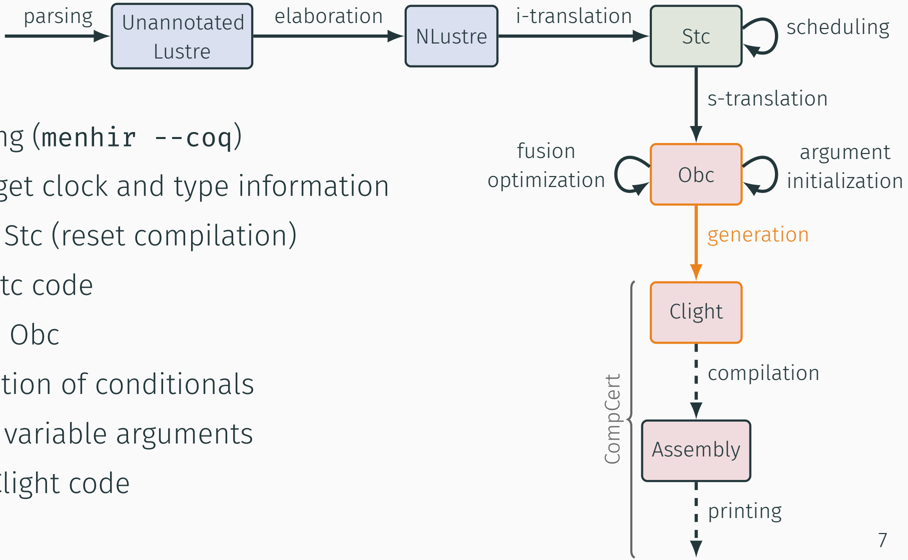


Vélus: a Verified Lustre Compiler



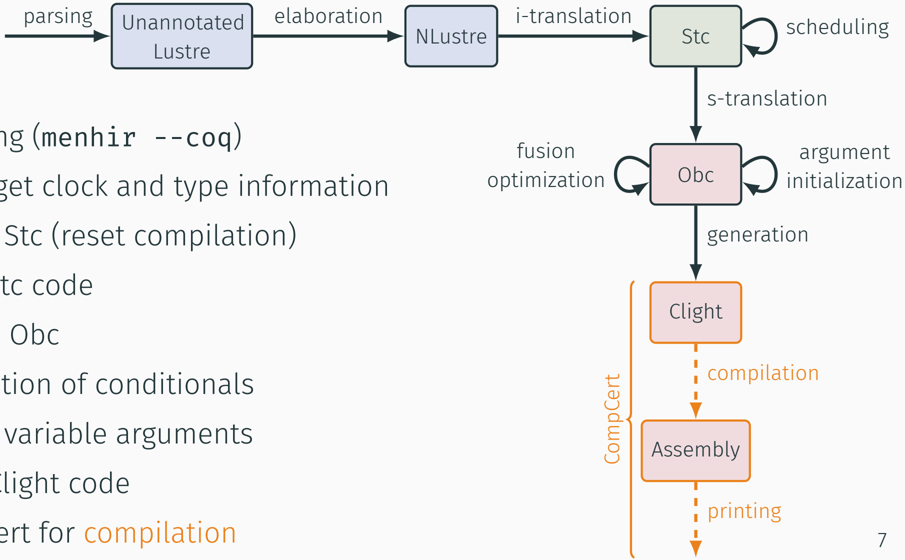
- validated parsing (`menhir --coq`)
- elaboration to get clock and type information
- i-translation to Stc (reset compilation)
- scheduling of Stc code
- s-translation to Obc
- fusion optimization of conditionals
- initialization of variable arguments

Vélus: a Verified Lustre Compiler



- validated parsing (`menhir --coq`)
- elaboration to get clock and type information
- i-translation to Stc (reset compilation)
- scheduling of Stc code
- s-translation to Obc
- fusion optimization of conditionals
- initialization of variable arguments
- Generation of Clight code

Vélus: a Verified Lustre Compiler



- validated parsing (`menhir --coq`)
- elaboration to get clock and type information
- i-translation to Stc (reset compilation)
- scheduling of Stc code
- s-translation to Obc
- fusion optimization of conditionals
- initialization of variable arguments
- Generation of Clight code
- Rely on CompCert for compilation

Lustre

```
node euler(x0, u: double)
  returns (x: double);
let
  x = x0 fby (x + 0.1 * u);
tel

node ins(gps, xv: double)
  returns (x: double, alarm: bool)
  var pxa, xe: double; k: int;
let
  k = 0 fby (k + 1);
  alarm = (k >= 50);
  xe = euler((gps, xv) when not alarm);
  pxa = (0. fby x) when alarm;
  x = merge alarm pxa xe;
tel

node nav(gps: double, xv: double, s: bool)
  returns (x: double, alarm: bool)
  var r: bool, c: bool;
let
  (x, alarm) = merge c
               (gps when c, false)
               ((restart ins every r)
                   ((gps, xv) whenot c));
  c = true fby (merge c (not s when c)
                         (s whenot c));
  r = false fby (s and c);
tel
```

Lustre

```
node euler(x0, u: double)
  returns (x: double);
let
  x = x0 fby (x + 0.1 * u);
tel

node ins(gps, xv: double)
  returns (x: double, alarm: bool)
  var pxa, xe: double; k: int;
let
  k = 0 fby (k + 1);
  alarm = (k >= 50);
  xe = euler((gps, xv) when not alarm);
  pxa = (0. fby x) when alarm;
  x = merge alarm pxa xe;
tel

node nav(gps: double, xv: double, s: bool)
  returns (x: double, alarm: bool)
  var r: bool, c: bool;
let
  (x, alarm) = merge c
                 (gps when c, false)
                 ((restart ins every r)
                    ((gps, xv) whenot c));
  c = true fby (merge c (not s when c)
                        (s whenot c));
  r = false fby (s and c);
tel
```

C

```c
struct euler {
  bool i;
  double px;
};
struct ins {
  int k;
  double px;
  struct euler xe;
};
struct fun$ins$step {
  double x;
  bool alarm;
};
struct nav {
  bool c;
  bool r;
  struct ins insr;
};
struct fun$nav$step {
  double x;
  bool alarm;
};

double fun$euler$step(struct euler *self,
                      double x0, double u) {
  register double x;
  if (self->i) {
    x = x0;
  } else {
    x = self->px;
  }
  self->i = false;
  self->px = x + 0.10000000000000006 * u;
  return x;
}

void fun$euler$reset(struct euler *self) {
  self->i = true;
  self->px = 0;
  return;
}

void fun$ins$step(struct ins *self,
                  struct fun$ins$step *out,
                  double gps, double xv) {
  register double step$x;
  register double xe;
  out->alarm = self->k >= 50;
  self->k = self->k + 1;
  if (out->alarm) { out->x = self->px; }
  else {
    step$x = fun$euler$step(&(self->xe), gps, xv);
    xe = step$x;
    out->x = xe;
  }
  self->px = out->x;
  return;
}

void fun$ins$reset(struct ins *self) {
  self->k = 0;
  self->px = 0;
  fun$euler$reset(&(self->xe));
  return;
}

void fun$nav$step(struct nav *self,
                  struct fun$nav$step *out,
                  double gps, double xv, bool s) {
  struct fun$ins$step out$insr$step;
  register bool cm;
  register double insr;
  register bool alr;
  if (self->r) { fun$ins$reset(&(self->insr)); }
  self->r = s & self->c;
  if (self->c) {
    cm = !s;
    out->x = gps;
    out->alarm = false;
  } else {
    fun$ins$step(&(self->insr, &out$insr$step, gps, xv);
    insr = out$insr$step.x;
    alr = out$insr$step.alarm;
    cm = s;
    out->x = insr;
    out->alarm = alr;
  }
  self->c = cm;
  return;
}

void fun$nav$reset(struct nav *self) {
  self->c = true;
  self->r = false;
  fun$ins$reset(&(self->insr);
  return;
}

struct nav self$;
double volatile gps$;
double volatile xv$;
bool volatile s$;
double volatile x$;
bool volatile alarm$;

int main(void) {
  struct fun$nav$step out$step;
  register double gps;
  register double xv;
  register bool s;

  fun$nav$reset(&self$);

  while (true) {
    gps = volatile_load(&gps$);
    xv = volatile_load(&xv$);
    s = volatile_load(&s$);

    fun$nav$step(&self$, &out$step, gps, xv, s);

    volatile_store(&x$, out$step.x);
    volatile_store(&alarm$, out$step.alarm);
  }
}
```

Lustre

```
node euler(x0, u: double)
  returns (x: double);
let
  x = x0 fby (x + 0.1 * u);
tel

node ins(gps, xv: double)
  returns (x: double, alarm: bool)
  var pxa, xe: double; k: int;
let
  k = 0 fby (k + 1);
  alarm = (k >= 50);
  xe = euler((gps, xv) when not alarm);
  pxa = (0. fby x) when alarm;
  x = merge alarm pxa xe;
tel

node nav(gps: double, xv: double, s: bool)
  returns (x: double, alarm: bool)
  var r: bool, c: bool;
let
  (x, alarm) = merge c
                 (gps when c, false)
                 ((restart ins every r)
                    ((gps, xv) whenot c));
  c = true fby (merge c (not s when c)
                        (s whenot c));
  r = false fby (s and c);
tel
```

C

```c
struct euler {
  bool i;
  double px;
};
struct ins {
  int k;
  double px;
  struct euler xe;
};
struct fun$ins$step {
  double x;
  bool alarm;
};
struct nav {
  bool c;
  bool r;
  struct ins insr;
};
struct fun$nav$step {
  double x;
  bool alarm;
};

double fun$euler$step(struct euler *self,
                      double x0, double u) {
  register double x;
  if (self->i) {
    x = x0;
  } else {
    x = self->px;
  }
  self->i = false;
  self->px = x + 0.10000000000000006 * u;
  return x;
}

void fun$euler$reset(struct euler *self) {
  self->i = true;
  self->px = 0;
  return;
}

void fun$ins$step(struct ins *self,
                  struct fun$ins$step *out,
                  double gps, double xv) {
  register double step$x;
  register double xe;
  out->alarm = self->k >= 50;
  self->k = self->k + 1;
  if (out->alarm) { out->x = self->px; }
  else {
    step$x = fun$euler$step(&(self->xe), gps, xv);
    xe = step$x;
    out->x = xe;
  }
  self->px = out->x;
  return;
}

void fun$ins$reset(struct ins *self) {
  self->k = 0;
  self->px = 0;
  fun$euler$reset(&(self->xe));
  return;
}

void fun$nav$step(struct nav *self,
                  struct fun$nav$step *out,
                  double gps, double xv, bool s) {
  struct fun$ins$step out$insr$step;
  register bool cm;
  register double insr;
  register bool alr;
  if (self->r) { fun$ins$reset(&(self->insr)); }
  self->r = s & self->c;
  if (self->c) {
    cm = !s;
    out->x = gps;
    out->alarm = false;
  } else {
    fun$ins$step(&(self->insr, &out$insr$step, gps, xv);
    insr = out$insr$step.x;
    alr = out$insr$step.alarm;
    cm = s;
    out->x = insr;
    out->alarm = alr;
  }
  self->c = cm;
  return;
}

void fun$nav$reset(struct nav *self) {
  self->c = true;
  self->r = false;
  fun$ins$reset(&(self->insr);
  return;
}

struct nav self$;
double volatile gps$;
double volatile xv$;
bool volatile s$;
double volatile x$;
bool volatile alarm$;

int main(void) {
  struct fun$nav$step out$step;
  register double gps;
  register double xv;
  register bool s;

  fun$nav$reset(&self$);

  while (true) {
    gps = volatile_load(&gps$);
    xv = volatile_load(&xv$);
    s = volatile_load(&s$);

    fun$nav$step(&self$, &out$step, gps, xv, s);

    volatile_store(&x$, out$step.x);
    volatile_store(&alarm$, out$step.alarm);
  }
}
```

translated code

Lustre

```
node euler(x0, u: double)
  returns (x: double);
let
  x = x0 fby (x + 0.1 * u);
tel

node ins(gps, xv: double)
  returns (x: double, alarm: bool)
  var pxa, xe: double; k: int;
let
  k = 0 fby (k + 1);
  alarm = (k >= 50);
  xe = euler((gps, xv) when not alarm);
  pxa = (0. fby x) when alarm;
  x = merge alarm pxa xe;
tel

node nav(gps: double, xv: double, s: bool)
  returns (x: double, alarm: bool)
  var r: bool, c: bool;
let
  (x, alarm) = merge c
                 (gps when c, false)
                 ((restart ins every r)
                    ((gps, xv) whenot c));
  c = true fby (merge c (not s when c)
                        (s whenot c));
  r = false fby (s and c);
tel
```

C

```
struct euler {
  bool i;
  double px;
};
struct ins {
  int k;
  double px;
  struct euler xe;
};
struct fun$ins$step {
  double x;
  bool alarm;
};
struct nav {
  bool c;
  bool r;
  struct ins insr;
};
struct fun$nav$step {
  double x;
  bool alarm;
};

double fun$euler$step(struct euler *self,
                      double x0, double u) {
  register double x;
  if (self->i) {
    x = x0;
  } else {
    x = self->px;
  }
  self->i = false;
  self->px = x + 0.10000000000000006 * u;
  return x;
}

void fun$euler$reset(struct euler *self) {
  self->i = true;
  self->px = 0;
  return;
}

void fun$ins$step(struct ins *self,
                  struct fun$ins$step *out,
                  double gps, double xv) {
  register double step$x;
  register double xe;
  out->alarm = self->k >= 50;
  self->k = self->k + 1;
  if (out->alarm) { out->x = self->px; }
  else {
    step$x = fun$euler$step(&(self->xe), gps, xv);
    xe = step$x;
    out->x = xe;
  }
  self->px = out->x;
  return;
}

void fun$ins$reset(struct ins *self) {
  self->k = 0;
  self->px = 0;
  fun$euler$reset(&(self->xe));
  return;
}

void fun$nav$step(struct nav *self,
                  struct fun$nav$step *out,
                  double gps, double xv, bool s) {
  struct fun$ins$step out$insr$step;
  register bool cm;
  register double insr;
  register bool alr;
  if (self->r) { fun$ins$reset(&(self->insr)); }
  self->r = s & self->c;
  if (self->c) {
    cm = !s;
    out->x = gps;
    out->alarm = false;
  } else {
    fun$ins$step(&(self->insr, &out$insr$step, gps, xv);
    insr = out$insr$step.x;
    alr = out$insr$step.alarm;
    cm = s;
    out->x = insr;
    out->alarm = alr;
  }
  self->c = cm;
  return;
}
```

main loop

```
void fun$nav$reset(struct nav *self) {
  self->c = true;
  self->r = false;
  fun$ins$reset(&(self->insr);
  return;
}

struct nav self$;
double volatile gps$;
double volatile xv$;
bool volatile s$;
double volatile x$;
bool volatile alarm$;

int main(void) {
  struct fun$nav$step out$step;
  register double gps;
  register double xv;
  register bool s;

  fun$nav$reset(&self$);

  while (true) {
    gps = volatile_load(&gps$);
    xv = volatile_load(&xv$);
    s = volatile_load(&s$);

    fun$nav$step(&self$, &out$step, gps, xv, s);

    volatile_store(&x$, out$step.x);
    volatile_store(&alarm$, out$step.alarm);
  }
}
```

Lustre

```
node euler(x0, u: double)
  returns (x: double);
let
  x = x0 fby (x + 0.1 * u);
tel

node ins(gps, xv: double)
  returns (x: double, alarm: bool)
  var pxa, xe: double; k: int;
let
  k = 0 fby (k + 1);
  alarm = (k >= 50);
  xe = euler((gps, xv) when not alarm);
  pxa = (0. fby x) when alarm;
  x = merge alarm pxa xe;
tel

node nav(gps: double, xv: double, s: bool)
  returns (x: double, alarm: bool)
  var r: bool, c: bool;
let
  (x, alarm) = merge c
                 (gps when c, false)
                 ((restart ins every r)
                   ((gps, xv) whenot c));
  c = true fby (merge c (not s when c)
                        (s whenot c));
  r = false fby (s and c);
tel
```

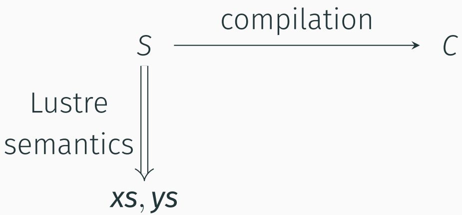
Assembly

```
	.data
	.align	8
	.globl	self$
self$:
	.space	40
	.type	self$, @object
	.size	self$, . - self$
	.data
	.align	8
	.globl	x$
x$:
	.space	8
	.type	x$, @object
	.size	x$, . - x$
	.data
	.align	8
	.globl	alarm$
alarm$:
	.space	1
	.type	alarm$, @object
	.size	alarm$, . - alarm$
	.data
	.align	8
	.globl	gps$
gps$:
	.space	8
	.type	gps$, @object
	.size	gps$, . - gps$
	.data
	.align	8
	.globl	xv$
xv$:
	.space	8
	.type	xv$, @object
	.size	xv$, . - xv$
	.data
	.align	8
	.globl	s$
s$:
	.space	1
	.type	s$, @object
	.size	s$, . - s$
	.text
	.align	16
	.globl	fun$euler$step
fun$euler$step:
	.cfi_startproc
	subq	$8, %rsp
	.cfi_adjust_cfa_offset	8
	leaq	16(%rsp), %rax
	movq	%rax, 0(%rsp)
	movzbl	0(%rdi), %eax
	cmpl	$0, %eax
	jne	.L100
	movsd	8(%rdi), %xmm0
.L100:
	xorl	%ecx, %ecx
	movb	%cl, 0(%rdi)
	movsd	.L101(%rip), %xmm2 # 0.100000000000000006
	mulsd	%xmm1, %xmm2
	movapd	%xmm0, %xmm1
	addsd	%xmm2, %xmm1
	movsd	%xmm1, 8(%rdi)
	addq	$8, %rsp
	ret
	.cfi_endproc
	.type	fun$euler$step, @function
	.size	fun$euler$step, . - fun$euler$step
	.section	.rodata.cst8,"aM",@progbits,8
	.align	8
.L101:	.quad	0x3fb999999999999a
	.text
	.align	16
	.globl	fun$euler$reset
fun$euler$reset:
	.cfi_startproc
	subq	$8, %rsp
	.cfi_adjust_cfa_offset	8
	leaq	16(%rsp), %rax
	movq	%rax, 0(%rsp)
	movl	$1, %eax
	movb	%al, 0(%rdi)
	xorpd	%xmm0, %xmm0
	movsd	%xmm0, 8(%rdi)
	addq	$8, %rsp
	ret
	.cfi_endproc
	.type	fun$euler$reset, @function
	.size	fun$euler$reset, . - fun$euler$reset
	.text
	.align	16
	.globl	fun$ins$step
fun$ins$step:
	.cfi_startproc
	subq	$24, %rsp
	.cfi_adjust_cfa_offset	24
	leaq	32(%rsp), %rax
	movq	%rax, 0(%rsp)
	movq	%rbx, 8(%rsp)
	movq	%rbp, 16(%rsp)
	movq	%rsi, %rbx
	movq	%rdi, %rbp
	movl	0(%rbp), %r9d
	cmpl	$50, %r9d
	setge	%r8b
	movzbl	%r8b, %r8d
	movb	%r8b, 8(%rbx)
	movl	0(%rbp), %ecx
	leal	1(%ecx), %eax
	movl	%eax, 0(%rbp)
	movzbl	8(%rbx), %edx
	cmpl	$0, %edx
	jne	.L102
	leaq	16(%rbp), %rdi
	call	fun$euler$step
	movsd	%xmm0, 0(%rbx)
	jmp	.L103
.L102:
	movsd	8(%rbp), %xmm2
	movsd	%xmm2, 0(%rbx)
.L103:
	movsd	0(%rbx), %xmm3
	movsd	%xmm3, 8(%rbp)
	movq	8(%rsp), %rbx
	movq	16(%rsp), %rbp
	addq	$24, %rsp
	ret
	.cfi_endproc
	.type	fun$ins$step, @function
	.size	fun$ins$step, . - fun$ins$step
	.text
	.align	16
	.globl	fun$ins$reset
fun$ins$reset:
	.cfi_startproc
	subq	$8, %rsp
	.cfi_adjust_cfa_offset	8
	leaq	16(%rsp), %rax
	movq	%rax, 0(%rsp)
	xorl	%eax, %eax
	movl	%eax, 0(%rdi)
	xorpd	%xmm0, %xmm0
	movsd	%xmm0, 8(%rdi)
	leaq	16(%rdi), %rdi
	addq	$8, %rsp
	jmp	fun$euler$reset
	.cfi_endproc
	.type	fun$ins$reset, @function
	.size	fun$ins$reset, . - fun$ins$reset
	.text
	.align	16
	.globl	fun$nav$step
fun$nav$step:
	.cfi_startproc
	subq	$72, %rsp
	.cfi_adjust_cfa_offset	72
	leaq	80(%rsp), %rax
	movq	%rax, 0(%rsp)
	movq	%rbx, 8(%rsp)
	movq	%rbp, 16(%rsp)
	movq	%r12, 24(%rsp)
	movq	%rdx, %rbx
	movsd	%xmm1, 40(%rsp)
	movsd	%xmm0, 32(%rsp)
	movq	%rsi, %rbp
	movq	%rdi, %r12
	movzbl	1(%r12), %edi
	cmpl	$0, %edi
	je	.L104
	leaq	8(%r12), %rdi
	call	fun$ins$reset
.L104:
	movzbl	0(%r12), %r9d
	movq	%rbx, %rax
	andl	%r9d, %eax
	cmpl	$0, %eax
	setne	%al
	movzbl	%al, %eax
	movb	%al, 1(%r12)
	movzbl	0(%r12), %r10d
	cmpl	$0, %r10d
	jne	.L105
	leaq	8(%r12), %rdi
	leaq	48(%rsp), %rsi
	movsd	40(%rsp), %xmm1
	movsd	32(%rsp), %xmm0
	call	fun$ins$step
	movsd	48(%rsp), %xmm3
	movzbl	56(%rsp), %esi
	movsd	%xmm3, 0(%rbp)
	cmpl	$0, %esi
	setne	%dl
	movzbl	%dl, %edx
	movb	%dl, 8(%rbp)
	jmp	.L106
.L105:
	cmpl	$0, %ebx
	sete	%bl
	movzbl	%bl, %ebx
	movsd	32(%rsp), %xmm0
	movsd	%xmm0, 0(%rbp)
	xorl	%r11d, %r11d
	movb	%r11b, 8(%rbp)
.L106:
	cmpl	$0, %ebx
	setne	%r8b
	movzbl	%r8b, %r8d
	movb	%r8b, 0(%r12)
	movq	8(%rsp), %rbx
	movq	16(%rsp), %rbp
	movq	24(%rsp), %r12
	addq	$72, %rsp
	ret
	.cfi_endproc
	.type	fun$nav$step, @function
	.size	fun$nav$step, . - fun$nav$step
	.text
	.align	16
	.globl	fun$nav$reset
fun$nav$reset:
	.cfi_startproc
	subq	$8, %rsp
	.cfi_adjust_cfa_offset	8
	leaq	16(%rsp), %rax
	movq	%rax, 0(%rsp)
	movl	$1, %eax
	movb	%al, 0(%rdi)
	xorl	%ecx, %ecx
	movb	%cl, 1(%rdi)
	leaq	8(%rdi), %rdi
	addq	$8, %rsp
	jmp	fun$ins$reset
	.cfi_endproc
	.type	fun$nav$reset, @function
	.size	fun$nav$reset, . - fun$nav$reset
	.text
	.align	16
	.globl	main_sync
main_sync:
	.cfi_startproc
	subq	$24, %rsp
	.cfi_adjust_cfa_offset	24
	leaq	32(%rsp), %rax
	movq	%rax, 0(%rsp)
	leaq	self$(%rip), %rdi
	call	fun$nav$reset
.L107:
	call	sync
	movsd	gps$(%rip), %xmm0
	movsd	xv$(%rip), %xmm1
	movzbl	s$(%rip), %ecx
	leaq	self$(%rip), %rdi
	leaq	8(%rsp), %rsi
	cmpl	$0, %ecx
	setne	%dl
	movzbl	%dl, %edx
	call	fun$nav$step
	movsd	8(%rsp), %xmm0
	movsd	%xmm0, x$(%rip)
	movzbl	16(%rsp), %eax
	cmpl	$0, %eax
	setne	%al
	movzbl	%al, %eax
	movb	%al, alarm$(%rip)
	jmp	.L107
	.cfi_endproc
	.type	main_sync, @function
	.size	main_sync, . - main_sync
	.text
	.align	16
	.globl	main_proved
main_proved:
	.cfi_startproc
	subq	$24, %rsp
	.cfi_adjust_cfa_offset	24
	leaq	32(%rsp), %rax
	movq	%rax, 0(%rsp)
	leaq	self$(%rip), %rdi
	call	fun$nav$reset
.L108:
	movsd	gps$(%rip), %xmm0
	movsd	xv$(%rip), %xmm1
	movzbl	s$(%rip), %eax
	leaq	self$(%rip), %rdi
	leaq	8(%rsp), %rsi
	cmpl	$0, %eax
	setne	%dl
	movzbl	%dl, %edx
	call	fun$nav$step
	movsd	8(%rsp), %xmm0
	movsd	%xmm0, x$(%rip)
	movzbl	16(%rsp), %eax
	cmpl	$0, %eax
	setne	%al
	movzbl	%al, %eax
	movb	%al, alarm$(%rip)
	jmp	.L108
	.cfi_endproc
	.type	main_proved, @function
	.size	main_proved, . - main_proved
	.text
	.align	16
	.globl	main
main:
	.cfi_startproc
	subq	$8, %rsp
	.cfi_adjust_cfa_offset	8
	leaq	16(%rsp), %rax
	movq	%rax, 0(%rsp)
	call	main_sync
	movq	%rbx, %rax
	addq	$8, %rsp
	ret
	.cfi_endproc
	.type	main, @function
	.size	main, . - main
```

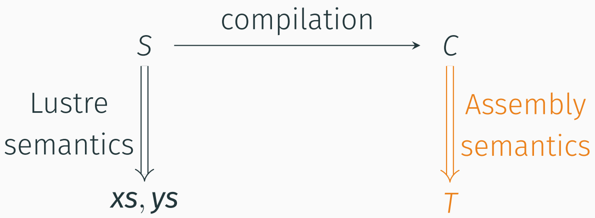
Correctness?

$$S \xrightarrow{\text{compilation}} C$$

Correctness?



Correctness?



Correctness?

Correctness?

Remark: we actually want the reverse direction, called *refinement*, that is, the observable behaviors of C are also observable behaviors of S.

Normalized Lustre Mechanization

4 types of top-level equations

$x = ce$	simple equation
$x = c$ `fby` e	`fby` equation
$\mathbf{x} = f(\mathbf{e})$	node instantiation
$\mathbf{x} = ($`restart` f `every` $r)(\mathbf{e})$	node instantiation with modular reset

Semantics

Streams as functions $\mathbb{N} \mapsto$ value:

$$\begin{array}{cccc} 0 & 1 & 2 & \cdots \\ \downarrow & \downarrow & \downarrow & \cdots \\ v_0 & v_1 & v_2 & \cdots \end{array}$$

lifted instantaneous semantics

Streams as coinductive types:

$$v_0 \cdot v_1 \cdot v_2 \cdot \cdots$$

coinductive description of the semantics

Instantaneous Semantics

$$\frac{}{R \vdash x \downarrow R(x)} \qquad \frac{R \vdash e_1 \downarrow \langle v_1 \rangle \quad R \vdash e_2 \downarrow \langle v_2 \rangle}{R \vdash e_1 + e_2 \downarrow \langle [\![+]\!](v_1, v_2) \rangle} \qquad \frac{R \vdash e_1 \downarrow \langle\,\rangle \quad R \vdash e_2 \downarrow \langle\,\rangle}{R \vdash e_1 + e_2 \downarrow \langle\,\rangle}$$

Lifted Semantics

$$\frac{\forall i,\ H_i(x) = s_i \quad \forall i,\ H_i \vdash e \downarrow s_i}{H \vdash x = e} \qquad \frac{\forall i,\ H_i \vdash e \downarrow s_i \quad \forall i,\ H_i(x) = \mathrm{fby}(c, s)_i}{H \vdash x = c \ \texttt{fby}\ e}$$

$$\frac{\forall i,\ H_i \vdash \boldsymbol{e} \downarrow xs_i \quad \vdash f(xs) \Downarrow ys \quad \forall i,\ H_i(\boldsymbol{x}) = ys_i}{H \vdash \boldsymbol{x} = f(\boldsymbol{e})}$$

Instantaneous Semantics

$$\frac{}{R \vdash x \downarrow R(x)} \qquad \frac{R \vdash e_1 \downarrow \langle v_1 \rangle \quad R \vdash e_2 \downarrow \langle v_2 \rangle}{R \vdash e_1 + e_2 \downarrow \langle [\![+]\!](v_1, v_2) \rangle} \qquad \frac{R \vdash e_1 \downarrow \langle\,\rangle \quad R \vdash e_2 \downarrow \langle\,\rangle}{R \vdash e_1 + e_2 \downarrow \langle\,\rangle}$$

Lifted Semantics

$$\frac{\forall i,\ H_i(x) = s_i \quad \forall i,\ H_i \vdash e \downarrow s_i}{H \vdash x = e} \qquad \frac{\forall i,\ H_i \vdash e \downarrow s_i \quad \forall i,\ H_i(x) = \mathrm{fby}(c, s)_i}{H \vdash x = c \ \mathtt{fby}\ e}$$

$$\frac{\forall i,\ H_i \vdash \boldsymbol{e} \downarrow xs_i \quad \vdash f(xs) \Downarrow ys \quad \forall i,\ H_i(\boldsymbol{x}) = ys_i}{H \vdash \boldsymbol{x} = f(\boldsymbol{e})}$$

Instantaneous Semantics

$$\frac{}{R \vdash x \downarrow R(x)} \qquad \frac{R \vdash e_1 \downarrow \langle v_1 \rangle \quad R \vdash e_2 \downarrow \langle v_2 \rangle}{R \vdash e_1 + e_2 \downarrow \langle [\![+]\!](v_1, v_2) \rangle} \qquad \frac{R \vdash e_1 \downarrow \langle\,\rangle \quad R \vdash e_2 \downarrow \langle\,\rangle}{R \vdash e_1 + e_2 \downarrow \langle\,\rangle}$$

Lifted Semantics

$$\frac{\forall i,\ H_i(x) = s_i \quad \forall i,\ H_i \vdash e \downarrow s_i}{H \vdash x = e} \qquad \frac{\forall i,\ H_i \vdash e \downarrow s_i \quad \forall i,\ H_i(x) = \mathrm{fby}(c, s)_i}{H \vdash x = c \texttt{ fby } e}$$

$$\frac{\forall i,\ H_i \vdash \boldsymbol{e} \downarrow xs_i \quad \vdash f(xs) \Downarrow ys \quad \forall i,\ H_i(\boldsymbol{x}) = ys_i}{H \vdash \boldsymbol{x} = f(\boldsymbol{e})}$$

Instantaneous Semantics

$$\frac{}{R \vdash x \downarrow R(x)} \qquad \frac{R \vdash e_1 \downarrow \langle v_1 \rangle \quad R \vdash e_2 \downarrow \langle v_2 \rangle}{R \vdash e_1 + e_2 \downarrow \langle [\![+]\!](v_1, v_2) \rangle} \qquad \frac{R \vdash e_1 \downarrow \langle\, \rangle \quad R \vdash e_2 \downarrow \langle\, \rangle}{R \vdash e_1 + e_2 \downarrow \langle\, \rangle}$$

Lifted Semantics

$$\frac{\forall i,\ H_i(x) = s_i \quad \forall i,\ H_i \vdash e \downarrow s_i}{H \vdash x = e} \qquad \frac{\forall i,\ H_i \vdash e \downarrow s_i \quad \forall i,\ H_i(x) = \mathrm{fby}(c, s)_i}{H \vdash x = c \ \mathtt{fby}\ e}$$

$$\frac{\forall i,\ H_i \vdash \boldsymbol{e} \downarrow xs_i \quad \vdash f(xs) \Downarrow ys \quad \forall i,\ H_i(\boldsymbol{x}) = ys_i}{H \vdash \boldsymbol{x} = f(\,\boldsymbol{e}\,)}$$

Instantaneous Semantics

$$\frac{}{R \vdash x \downarrow R(x)} \qquad \frac{R \vdash e_1 \downarrow \langle v_1 \rangle \quad R \vdash e_2 \downarrow \langle v_2 \rangle}{R \vdash e_1 + e_2 \downarrow \langle [\![+]\!](v_1, v_2) \rangle} \qquad \frac{R \vdash e_1 \downarrow \langle\ \rangle \quad R \vdash e_2 \downarrow \langle\ \rangle}{R \vdash e_1 + e_2 \downarrow \langle\ \rangle}$$

Lifted Semantics

$$\frac{\forall i,\ H_i(x) = s_i \quad \forall i,\ H_i \vdash e \downarrow s_i}{H \vdash x = e} \qquad \frac{\forall i,\ H_i \vdash e \downarrow s_i \quad \forall i,\ H_i(x) = \text{fby}(c, s)_i}{H \vdash x = c \ \texttt{fby}\ e}$$

$$\frac{\forall i,\ H_i \vdash \boldsymbol{e} \downarrow xs_i \quad \vdash f(xs) \Downarrow ys \quad \forall i,\ H_i(\boldsymbol{x}) = ys_i}{H \vdash \boldsymbol{x} = f(\boldsymbol{e})}$$

Instantaneous Semantics

$$\frac{}{R \vdash x \downarrow R(x)} \qquad \frac{R \vdash e_1 \downarrow \langle v_1 \rangle \quad R \vdash e_2 \downarrow \langle v_2 \rangle}{R \vdash e_1 + e_2 \downarrow \langle [\![+]\!](v_1, v_2) \rangle} \qquad \frac{R \vdash e_1 \downarrow \langle\, \rangle \quad R \vdash e_2 \downarrow \langle\, \rangle}{R \vdash e_1 + e_2 \downarrow \langle\, \rangle}$$

Lifted Semantics

$$\frac{\forall i,\ H_i(x) = s_i \quad \forall i,\ H_i \vdash e \downarrow s_i}{H \vdash x = e} \qquad \frac{\forall i,\ H_i \vdash e \downarrow s_i \quad \forall i,\ H_i(x) = \mathrm{fby}(c, s)_i}{H \vdash x = c \;\texttt{fby}\; e}$$

$$\frac{\forall i,\ H_i \vdash \boldsymbol{e} \downarrow xs_i \quad \vdash f(xs) \Downarrow ys \quad \forall i,\ H_i(\boldsymbol{x}) = ys_i}{H \vdash \boldsymbol{x} = f(\boldsymbol{e})}$$

Instantaneous Semantics

$$\frac{}{R \vdash x \downarrow R(x)} \qquad \frac{R \vdash e_1 \downarrow \langle v_1 \rangle \quad R \vdash e_2 \downarrow \langle v_2 \rangle}{R \vdash e_1 + e_2 \downarrow \langle [\![+]\!](v_1, v_2) \rangle} \qquad \frac{R \vdash e_1 \downarrow \langle\,\rangle \quad R \vdash e_2 \downarrow \langle\,\rangle}{R \vdash e_1 + e_2 \downarrow \langle\,\rangle}$$

Lifted Semantics

$$\frac{\forall i,\ H_i(x) = s_i \quad \forall i,\ H_i \vdash e \downarrow s_i}{H \vdash x = e} \qquad \frac{\forall i,\ H_i \vdash e \downarrow s_i \quad \forall i,\ H_i(x) = \mathrm{fby}(c, s)_i}{H \vdash x = c\ \texttt{fby}\ e}$$

$$\frac{\forall i,\ H_i \vdash \boldsymbol{e} \downarrow xs_i \quad \vdash f(xs) \Downarrow ys \quad \forall i,\ H_i(\boldsymbol{x}) = ys_i}{H \vdash \boldsymbol{x} = f(\boldsymbol{e})}$$

Instantaneous Semantics

$$\frac{}{R \vdash x \downarrow R(x)} \qquad \frac{R \vdash e_1 \downarrow \langle v_1 \rangle \quad R \vdash e_2 \downarrow \langle v_2 \rangle}{R \vdash e_1 + e_2 \downarrow \langle [\![+]\!](v_1, v_2) \rangle} \qquad \frac{R \vdash e_1 \downarrow \langle\, \rangle \quad R \vdash e_2 \downarrow \langle\, \rangle}{R \vdash e_1 + e_2 \downarrow \langle\, \rangle}$$

Lifted Semantics

$$\frac{\forall i,\ H_i(x) = s_i \quad \forall i,\ H_i \vdash e \downarrow s_i}{H \vdash x\ \texttt{=}\ e} \qquad \frac{\forall i,\ H_i \vdash e \downarrow s_i \quad \forall i,\ H_i(x) = \mathrm{fby}(c, s)_i}{H \vdash x\ \texttt{=}\ c\ \texttt{fby}\ e}$$

$$\frac{\forall i,\ H_i \vdash \boldsymbol{e} \downarrow xs_i \quad \vdash f(xs) \Downarrow ys \quad \forall i,\ H_i(\boldsymbol{x}) = ys_i}{H \vdash \boldsymbol{x}\ \texttt{=}\ f(\boldsymbol{e})}$$

Instantaneous Semantics

$$\frac{}{R \vdash x \downarrow R(x)} \qquad \frac{R \vdash e_1 \downarrow \langle v_1 \rangle \quad R \vdash e_2 \downarrow \langle v_2 \rangle}{R \vdash e_1 + e_2 \downarrow \langle [\![+]\!](v_1, v_2) \rangle} \qquad \frac{R \vdash e_1 \downarrow \langle\, \rangle \quad R \vdash e_2 \downarrow \langle\, \rangle}{R \vdash e_1 + e_2 \downarrow \langle\, \rangle}$$

Lifted Semantics

$$\frac{\forall i,\ H_i(x) = s_i \quad \forall i,\ H_i \vdash e \downarrow s_i}{H \vdash x = e} \qquad \frac{\forall i,\ H_i \vdash e \downarrow s_i \quad \forall i,\ H_i(x) = \mathrm{fby}(c, s)_i}{H \vdash x = c \ \mathtt{fby}\ e}$$

$$\frac{\forall i,\ H_i \vdash \boldsymbol{e} \downarrow xs_i \quad \vdash f(xs) \Downarrow ys \quad \forall i,\ H_i(\boldsymbol{x}) = ys_i}{H \vdash \boldsymbol{x} = f(\boldsymbol{e})}$$

Instantaneous Semantics

$$\frac{}{R \vdash x \downarrow R(x)} \qquad \frac{R \vdash e_1 \downarrow \langle v_1 \rangle \quad R \vdash e_2 \downarrow \langle v_2 \rangle}{R \vdash e_1 + e_2 \downarrow \langle [\![+]\!](v_1, v_2) \rangle} \qquad \frac{R \vdash e_1 \downarrow \langle\,\rangle \quad R \vdash e_2 \downarrow \langle\,\rangle}{R \vdash e_1 + e_2 \downarrow \langle\,\rangle}$$

Lifted Semantics

$$\frac{\forall i,\ H_i(x) = s_i \quad \forall i,\ H_i \vdash e \downarrow s_i}{H \vdash x = e} \qquad \frac{\forall i,\ H_i \vdash e \downarrow s_i \quad \forall i,\ H_i(x) = \mathrm{fby}(c, s)_i}{H \vdash x = c \ \texttt{fby}\ e}$$

$$\frac{\forall i,\ H_i \vdash \boldsymbol{e} \downarrow xs_i \quad \vdash f(xs) \Downarrow ys \quad \forall i,\ H_i(\boldsymbol{x}) = ys_i}{H \vdash \boldsymbol{x} = f(\boldsymbol{e})}$$

Instantaneous Semantics

$$\frac{}{R \vdash x \downarrow R(x)} \qquad \frac{R \vdash e_1 \downarrow \langle v_1 \rangle \quad R \vdash e_2 \downarrow \langle v_2 \rangle}{R \vdash e_1 + e_2 \downarrow \langle [\![+]\!](v_1, v_2) \rangle} \qquad \frac{R \vdash e_1 \downarrow \langle\ \rangle \quad R \vdash e_2 \downarrow \langle\ \rangle}{R \vdash e_1 + e_2 \downarrow \langle\ \rangle}$$

Lifted Semantics

$$\frac{\forall i,\ H_i(x) = s_i \quad \forall i,\ H_i \vdash e \downarrow s_i}{H \vdash x \texttt{ = } e} \qquad \frac{\forall i,\ H_i \vdash e \downarrow s_i \quad \forall i,\ H_i(x) = \mathrm{fby}(c, s)_i}{H \vdash x \texttt{ = } c \texttt{ fby } e}$$

$$\frac{\forall i,\ H_i \vdash \boldsymbol{e} \downarrow xs_i \quad \vdash f(xs) \Downarrow ys \quad \forall i,\ H_i(\boldsymbol{x}) = ys_i}{H \vdash \boldsymbol{x} \texttt{ = } f(\boldsymbol{e})}$$

Node Semantics

$$\frac{\mathrm{node}(G, f) = n \quad \forall i,\ H_i(n.\mathbf{in}) = xs_i \quad \forall i,\ H_i(n.\mathbf{out}) = ys_i \qquad \forall eq \in n.\mathbf{eqs},\ H \vdash eq}{\vdash f(xs) \Downarrow ys}$$

Node Semantics

$$\frac{\mathsf{node}(G, f) = n \quad \forall i,\ H_i(n.\mathbf{in}) = xs_i \quad \forall i,\ H_i(n.\mathbf{out}) = ys_i \quad \forall eq \in n.\mathbf{eqs},\ H \vdash eq}{\vdash f(xs) \Downarrow ys}$$

Node Semantics

$$\frac{\mathrm{node}(G, f) = n \quad \forall i,\ H_i(n.\mathbf{in}) = xs_i \quad \forall i,\ H_i(n.\mathbf{out}) = ys_i \quad \forall eq \in n.\mathbf{eqs},\ H \vdash eq}{\vdash f(xs) \Downarrow ys}$$

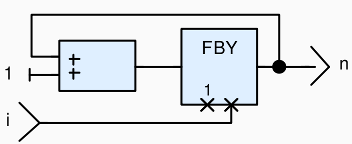
Node Semantics

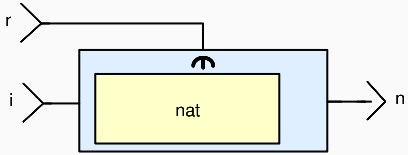
$$\frac{\mathrm{node}(G, f) = n \quad \forall i,\ H_i(n.\mathbf{in}) = xs_i \quad \forall i,\ H_i(n.\mathbf{out}) = ys_i \quad \forall eq \in n.\mathbf{eqs},\ H \vdash eq}{\vdash f(xs) \Downarrow ys}$$

Node Semantics

$$\frac{\text{node}(G, f) = n \quad \forall i,\ H_i(n.\textbf{in}) = xs_i \quad \forall i,\ H_i(n.\textbf{out}) = ys_i \qquad \forall eq \in n.\textbf{eqs},\ H \vdash eq}{\vdash f(xs) \Downarrow ys}$$

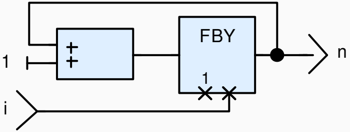
```
node nat(i: int)
  returns (n: int)
let
  n = i fby (n + 1);
tel
```

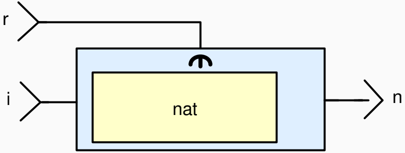




A Simpler Example: Intuitive Semantics of the Modular Reset

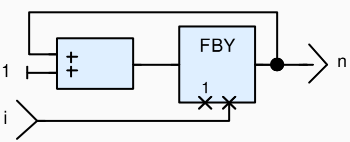
```
node nat(i: int)
  returns (n: int)
let
  n = i fby (n + 1);
tel
```

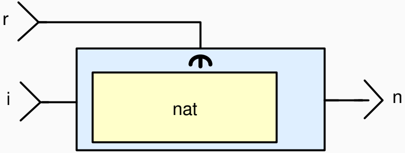




r	F
i	0
$nat(i)$	0
$(\texttt{restart}\ nat\ \texttt{every}\ r)(i)$	0

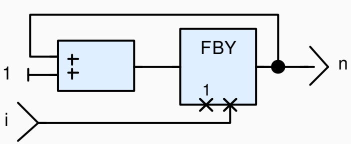
```
node nat(i: int)
  returns (n: int)
let
  n = i fby (n + 1);
tel
```

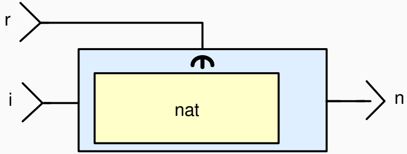




r	F	F
i	0	5
$nat(i)$	0	1
(**restart** nat **every** r)(i)	0	1

```
node nat(i: int)
  returns (n: int)
let
  n = i fby (n + 1);
tel
```

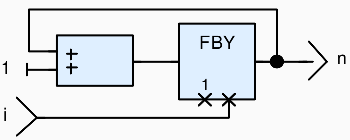


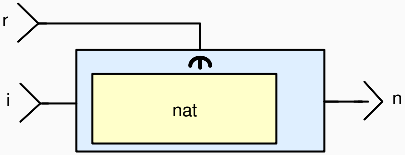


r	F	F	T
i	0	5	10
$nat(i)$	0	1	2
(**restart** nat **every** r)(i)	0	1	10

A Simpler Example: Intuitive Semantics of the Modular Reset

```
node nat(i: int)
  returns (n: int)
let
  n = i fby (n + 1);
tel
```

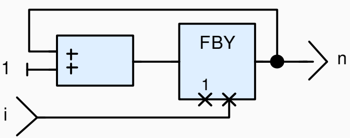


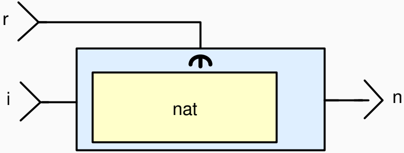


r	F	F	T	F
i	0	5	10	15
$nat(i)$	0	1	2	3
$(\texttt{restart}\ nat\ \texttt{every}\ r)(i)$	0	1	10	11

A Simpler Example: Intuitive Semantics of the Modular Reset

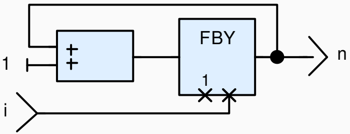
```
node nat(i: int)
  returns (n: int)
let
  n = i fby (n + 1);
tel
```

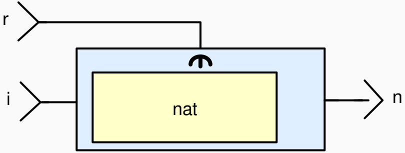




r	F	F	T	F	F
i	0	5	10	15	20
$nat(i)$	0	1	2	3	4
(**restart** nat **every** r)(i)	0	1	10	11	12

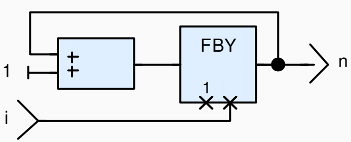
```
node nat(i: int)
  returns (n: int)
let
  n = i fby (n + 1);
tel
```

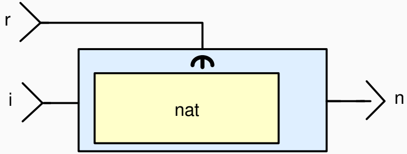




r	F	F	T	F	F	T
i	0	5	10	15	20	25
$nat(i)$	0	1	2	3	4	5
(**restart** *nat* **every** *r*)(*i*)	0	1	10	11	12	25

```
node nat(i: int)
  returns (n: int)
let
  n = i fby (n + 1);
tel
```

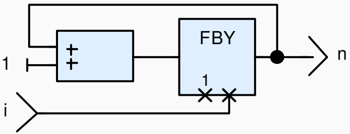


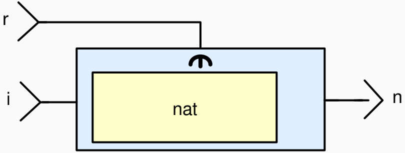


r	F	F	T	F	F	T	F
i	0	5	10	15	20	25	30
$nat(i)$	0	1	2	3	4	5	6
(**restart** *nat* **every** r)(i)	0	1	10	11	12	25	26

A Simpler Example: Intuitive Semantics of the Modular Reset

```
node nat(i: int)
  returns (n: int)
let
  n = i fby (n + 1);
tel
```

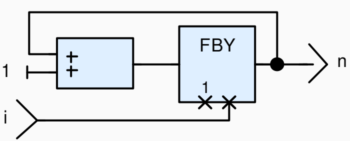


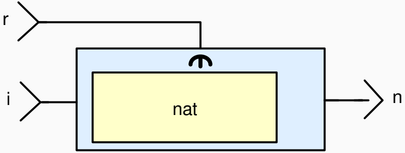


r	F	F	T	F	F	T	F	···
i	0	5	10	15	20	25	30	···
$nat(i)$	0	1	2	3	4	5	6	···
(**restart** *nat* **every** *r*)(*i*)	0	1	10	11	12	25	26	···

A Simpler Example: Intuitive Semantics of the Modular Reset

```
node nat(i: int)
  returns (n: int)
let
  n = i fby (n + 1);
tel
```





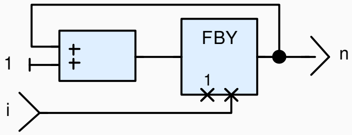
r	F	F	T	F	F	T	F	$\cdots$
i	0	5	10	15	20	25	30	$\cdots$
$nat(i)$	0	1	2	3	4	5	6	$\cdots$
(**restart** nat **every** r)(i)	0	1	10	11	12	25	26	$\cdots$

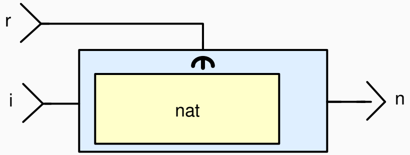
Can be implemented in a higher-order recursive language

A Simpler Example: Intuitive Semantics of the Modular Reset

Caspi 1994
Hamon and Pouzet 2000

```
node nat(i: int)
  returns (n: int)
let
  n = i fby (n + 1);
tel
```





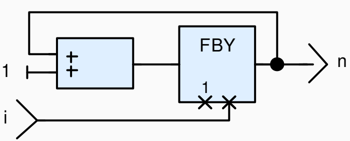
r	F	F	T	F	F	T	F	...
i	0	5	10	15	20	25	30	...
$nat(i)$	0	1	2	3	4	5	6	...
(**restart** nat **every** r)(i)	0	1	10	11	12	25	26	...

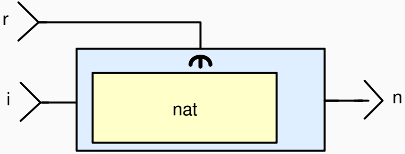
Can be implemented in a higher-order recursive language

A Simpler Example: Intuitive Semantics of the Modular Reset

Caspi 1994
Hamon and Pouzet 2000

```
node nat(i: int)
  returns (n: int)
let
  n = i fby (n + 1);
tel
```





r	F	F	T	F	F	T	F	...
i	0	5	10	15	20	25	30	...
$nat(i)$	0	1	2	3	4	5	6	...
$($**restart** nat **every** $r)(i)$	0	1	10	11	12	25	26	...

Can be implemented in a higher-order recursive language

A Simpler Example: Intuitive Semantics of the Modular Reset

r	F	F	T	F	F	T	F	$\cdots$
i	0	5	10	15	20	25	30	$\cdots$
(`restart` *nat* `every` r)(i)	0	1	10	11	12	25	26	$\cdots$

A Simpler Example: Intuitive Semantics of the Modular Reset

r	F	F	T	F	F	T	F	$\cdots$
count r	0	0	1	1	1	2	2	$\cdots$
i	0	5	10	15	20	25	30	$\cdots$
(`restart` *nat* `every` r)(i)	0	1	10	11	12	25	26	$\cdots$

A Simpler Example: Intuitive Semantics of the Modular Reset

r	F	F	T	F	F	T	F	$\cdots$
count r	0	0	1	1	1	2	2	$\cdots$
i	0	5	10	15	20	25	30	$\cdots$
$\text{mask}_r^0\ i$	0	5						$\cdots$
(`restart` *nat* `every` r)(i)	0	1	10	11	12	25	26	$\cdots$

A Simpler Example: Intuitive Semantics of the Modular Reset

r	F	F	T	F	F	T	F	$\cdots$
count r	0	0	1	1	1	2	2	$\cdots$
i	0	5	10	15	20	25	30	$\cdots$
$\text{mask}_r^0\ i$	0	5						$\cdots$
$nat(\text{ mask}_r^0\ i)$	0	1						$\cdots$
(`restart` *nat* `every` r)(i)	0	1	10	11	12	25	26	$\cdots$

A Simpler Example: Intuitive Semantics of the Modular Reset

r	F	F	T	F	F	T	F	$\cdots$
count r	0	0	1	1	1	2	2	$\cdots$
i	0	5	10	15	20	25	30	$\cdots$
$\text{mask}_r^0\ i$	0	5						$\cdots$
$nat(\text{mask}_r^0\ i)$	0	1						$\cdots$
$\text{mask}_r^1\ i$			10	15	20			$\cdots$
(`restart` nat `every` r)(i)	0	1	10	11	12	25	26	$\cdots$

A Simpler Example: Intuitive Semantics of the Modular Reset

r	F	F	T	F	F	T	F	$\cdots$
count r	0	0	1	1	1	2	2	$\cdots$
i	0	5	10	15	20	25	30	$\cdots$
$\text{mask}_r^0\ i$	0	5						$\cdots$
$nat(\ \text{mask}_r^0\ i\ )$	0	1						$\cdots$
$\text{mask}_r^1\ i$			10	15	20			$\cdots$
$nat(\ \text{mask}_r^1\ i\ )$			10	11	12			$\cdots$
(`restart` nat `every` r)(i)	0	1	10	11	12	25	26	$\cdots$

A Simpler Example: Intuitive Semantics of the Modular Reset

r	F	F	T	F	F	T	F	$\cdots$
count r	0	0	1	1	1	2	2	$\cdots$
i	0	5	10	15	20	25	30	$\cdots$
$\text{mask}_r^0\ i$	0	5						$\cdots$
$nat(\text{mask}_r^0\ i)$	0	1						$\cdots$
$\text{mask}_r^1\ i$			10	15	20			$\cdots$
$nat(\text{mask}_r^1\ i)$			10	11	12			$\cdots$
$\text{mask}_r^2\ i$						25	30	$\cdots$
(`restart` nat `every` r)(i)	0	1	10	11	12	25	26	$\cdots$

A Simpler Example: Intuitive Semantics of the Modular Reset

r	F	F	T	F	F	T	F	$\cdots$
count r	0	0	1	1	1	2	2	$\cdots$
i	0	5	10	15	20	25	30	$\cdots$
$\text{mask}_r^0\ i$	0	5						$\cdots$
$nat(\ \text{mask}_r^0\ i\ )$	0	1						$\cdots$
$\text{mask}_r^1\ i$			10	15	20			$\cdots$
$nat(\ \text{mask}_r^1\ i\ )$			10	11	12			$\cdots$
$\text{mask}_r^2\ i$						25	30	$\cdots$
$nat(\ \text{mask}_r^2\ i\ )$						25	26	$\cdots$
$(\texttt{restart}\ nat\ \texttt{every}\ r)(i)$	0	1	10	11	12	25	26	$\cdots$

A Simpler Example: Intuitive Semantics of the Modular Reset

r	F	F	T	F	F	T	F	$\cdots$
count r	0	0	1	1	1	2	2	$\cdots$
i	0	5	10	15	20	25	30	$\cdots$
$\text{mask}_r^0\ i$	0	5						$\cdots$
$nat(\text{mask}_r^0\ i)$	0	1						$\cdots$
$\text{mask}_r^1\ i$			10	15	20			$\cdots$
$nat(\text{mask}_r^1\ i)$			10	11	12			$\cdots$
$\text{mask}_r^2\ i$						25	30	$\cdots$
$nat(\text{mask}_r^2\ i)$						25	26	$\cdots$
$\vdots$								
(`restart` *nat* `every` r)(i)	0	1	10	11	12	25	26	$\cdots$

Node instantiation

$$\frac{\forall i,\ H_i \vdash \boldsymbol{e} \downarrow xs_i \quad \vdash f(xs) \Downarrow ys \quad \forall i,\ H_i(\boldsymbol{x}) = ys_i}{H \vdash \boldsymbol{x} = f(\boldsymbol{e})}$$

Node instantiation

$$\frac{\forall i,\ H_i \vdash e \downarrow xs_i \quad \vdash f(xs) \Downarrow ys \quad \forall i,\ H_i(x) = ys_i}{H \vdash x = f(e)}$$

Modular reset

$$\frac{}{H \vdash x = (\texttt{restart}\ f\ \texttt{every}\ y)(e)}$$

Node instantiation

$$\frac{\forall i,\ H_i \vdash \mathbf{e} \downarrow xs_i \quad \vdash f(xs) \Downarrow ys \quad \forall i,\ H_i(\mathbf{x}) = ys_i}{H \vdash \mathbf{x} = f(\mathbf{e})}$$

Modular reset

$$\frac{\forall i,\ H_i \vdash \mathbf{e} \downarrow xs_i \qquad \qquad \forall i,\ H_i(\mathbf{x}) = ys_i}{H \vdash \mathbf{x} = (\texttt{restart}\ f\ \texttt{every}\ y)(\mathbf{e})}$$

Node instantiation

$$\frac{\forall i,\ H_i \vdash \mathbf{e} \downarrow xs_i \quad \vdash f(xs) \Downarrow ys \quad \forall i,\ H_i(\mathbf{x}) = ys_i}{H \vdash \mathbf{x} = f(\mathbf{e})}$$

Modular reset

$$\frac{\begin{array}{c}\forall i,\ H_i(y) = rs_i \quad r = \text{bools-of } rs \\ \forall i,\ H_i \vdash \mathbf{e} \downarrow xs_i \quad \forall k,\ \vdash f\left(\text{mask}_r^k\, \mathbf{xs}\right) \Downarrow \text{mask}_r^k\, \mathbf{ys} \quad \forall i,\ H_i(\mathbf{x}) = ys_i\end{array}}{H \vdash \mathbf{x} = (\mathtt{restart}\ f\ \mathtt{every}\ y)(\mathbf{e})}$$

Node instantiation

$$\frac{\forall i,\ H_i \vdash \boldsymbol{e} \downarrow xs_i \quad \vdash f(xs) \Downarrow ys \quad \forall i,\ H_i(\boldsymbol{x}) = ys_i}{H \vdash \boldsymbol{x} = f(\boldsymbol{e})}$$

Modular reset

$$\frac{\begin{array}{c}\forall i,\ H_i(y) = rs_i \quad r = \text{bools-of } rs \\ \forall i,\ H_i \vdash \boldsymbol{e} \downarrow xs_i \quad \forall k,\ \vdash f\left(\text{mask}_r^k\, \boldsymbol{xs}\right) \Downarrow \text{mask}_r^k\, \boldsymbol{ys} \quad \forall i,\ H_i(\boldsymbol{x}) = ys_i\end{array}}{H \vdash \boldsymbol{x} = (\texttt{restart}\ f\ \texttt{every}\ y)(\boldsymbol{e})}$$

Universally quantified relation: unbounded number of constraints

Compiling the modular reset: from NLustre to Stc

A Problem with the Compilation from NLustre to Obc

```
node driver(x0, y0, u, v: double, r: bool)
  returns (x, y: double)
  var ax, ay: bool;
let
  x, ax = (restart ins every r)(x0, u);
  y, ay = (restart ins every r)(y0, v);
tel
```

```
class driver {
  instance x: ins, y: ins;

  reset() { ins(x).reset();
            ins(y).reset() }

  step(x0, y0, u, v: double, r: bool)
    returns (x, y: double)
    var ax, ay: bool
  {
    if r { ins(x).reset() };
    x, ax := ins(x).step(x0, u);
    if r { ins(y).reset() };
    y, ay := ins(y).step(y0, v)
  }
}
```

A Problem with the Compilation from NLustre to Obc

```
node driver(x0, y0, u, v: double, r: bool)
  returns (x, y: double)
  var ax, ay: bool;
let
  x, ax = (restart ins every r)(x0, u);
  y, ay = (restart ins every r)(y0, v);
tel
```

```
class driver {
  instance x: ins, y: ins;

  reset() { ins(x).reset();
            ins(y).reset() }

  step(x0, y0, u, v: double, r: bool)
    returns (x, y: double)
    var ax, ay: bool
  {
    if r { ins(x).reset() };
    x, ax := ins(x).step(x0, u);
    if r { ins(y).reset() };
    y, ay := ins(y).step(y0, v)
  }
}
```

A Problem with the Compilation from NLustre to Obc

```
node driver(x0, y0, u, v: double, r: bool)
  returns (x, y: double)
  var ax, ay: bool;
let
  x, ax = (restart ins every r)(x0, u);
  y, ay = (restart ins every r)(y0, v);
tel
```

```
class driver {
  instance x: ins, y: ins;

  reset() { ins(x).reset();
            ins(y).reset() }

  step(x0, y0, u, v: double, r: bool)
    returns (x, y: double)
    var ax, ay: bool
  {
    if r { ins(x).reset() };
    x, ax := ins(x).step(x0, u);
    if r { ins(y).reset() };
    y, ay := ins(y).step(y0, v)
  }
}
```

A Problem with the Compilation from NLustre to Obc

```
node driver(x0, y0, u, v: double, r: bool)
  returns (x, y: double)
  var ax, ay: bool;
let
  x, ax = (restart ins every r)(x0, u);
  y, ay = (restart ins every r)(y0, v);
tel
```

```
class driver {
  instance x: ins, y: ins;

  reset() { ins(x).reset();
            ins(y).reset() }

  step(x0, y0, u, v: double, r: bool)
    returns (x, y: double)
    var ax, ay: bool
  {
    if r { ins(x).reset() };
    x, ax := ins(x).step(x0, u);
    if r { ins(y).reset() };
    y, ay := ins(y).step(y0, v)
  }
}
```

A Problem with the Compilation from NLustre to Obc

```
node driver(x0, y0, u, v: double, r: bool)
  returns (x, y: double)
  var ax, ay: bool;
let
  x, ax = (restart ins every r)(x0, u);
  y, ay = (restart ins every r)(y0, v);
tel
```

```
class driver {
  instance x: ins, y: ins;

  reset() { ins(x).reset();
            ins(y).reset() }

  step(x0, y0, u, v: double, r: bool)
    returns (x, y: double)
    var ax, ay: bool
  {
    if r { ins(x).reset() };
    x, ax := ins(x).step(x0, u);
    if r { ins(y).reset() };
    y, ay := ins(y).step(y0, v)
  }
}
```

scheduling *and* introducing state

VS

introducing state *then* scheduling

Stc: Synchronous Transition Code

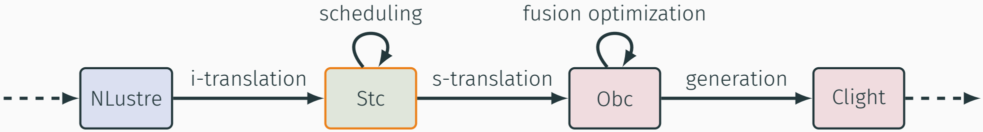
Propose a new intermediate language

- Invariant semantics under permutation
- Separate reset construct
- Explicit state variables and instances

Stc: Synchronous Transition Code

Propose a new intermediate language

- Invariant semantics under permutation
- Separate reset construct
- Explicit state variables and instances



```
node ins(gps, xv: double)
  returns (x: double, alarm: bool)
  var k: int, px: double,
      xe: double when not alarm;
let
  k = 0 fby k + 1;
  alarm = (k >= 50);
  xe = euler(gps when not alarm,
             xv when not alarm);
  x = merge alarm (px when alarm) xe;
  px = 0. fby x;
tel
```

```
system ins {
  init k = 0, px = 0.;
  sub xe: euler;

  transition(gps, xv: double)
    returns (x: double, alarm: bool)
    var xe: double when not alarm;
  {
    next k = k + 1;
    alarm = (k >= 50);
    xe = euler<xe>(gps when not alarm,
                   xv when not alarm);
    x = merge alarm (px when alarm) xe;
    next px = x;
  }
}
```

```
node ins(gps, xv: double)
  returns (x: double, alarm: bool)
  var k: int, px: double,
      xe: double when not alarm;
let
  k = 0 fby k + 1;
  alarm = (k >= 50);
  xe = euler(gps when not alarm,
             xv when not alarm);
  x = merge alarm (px when alarm) xe;
  px = 0. fby x;
tel
```

```
system ins {
  init k = 0, px = 0.;
  sub xe: euler;

  transition(gps, xv: double)
    returns (x: double, alarm: bool)
    var xe: double when not alarm;
  {
    next k = k + 1;
    alarm = (k >= 50);
    xe = euler<xe>(gps when not alarm,
                   xv when not alarm);
    x = merge alarm (px when alarm) xe;
    next px = x;
  }
}
```

NLustre | Stc

```
node ins(gps, xv: double)
  returns (x: double, alarm: bool)
  var k: int, px: double,
      xe: double when not alarm;
let
  k = 0 fby k + 1;
  alarm = (k >= 50);
  xe = euler(gps when not alarm,
             xv when not alarm);
  x = merge alarm (px when alarm) xe;
  px = 0. fby x;
tel
```

only introducing state

```
system ins {
  init k = 0, px = 0.;
  sub xe: euler;

  transition(gps, xv: double)
    returns (x: double, alarm: bool)
    var xe: double when not alarm;
  {
    next k = k + 1;
    alarm = (k >= 50);
    xe = euler<xe>(gps when not alarm,
                   xv when not alarm);
    x = merge alarm (px when alarm) xe;
    next px = x;
  }
}
```

NLustre

```
node ins(gps, xv: double)
  returns (x: double, alarm: bool)
  var k: int, px: double,
      xe: double when not alarm;
let
  k = 0 fby k + 1;
  alarm = (k >= 50);
  xe = euler(gps when not alarm,
             xv when not alarm);
  x = merge alarm (px when alarm) xe;
  px = 0. fby x;
tel
```

only introducing state

Stc

```
system ins {
  init k = 0, px = 0.;
  sub xe: euler;

  transition(gps, xv: double)
    returns (x: double, alarm: bool)
    var xe: double when not alarm;
  {
    next k = k + 1;
    alarm = (k >= 50);
    xe = euler<xe>(gps when not alarm,
                   xv when not alarm);
    x = merge alarm (px when alarm) xe;
    next px = x;
  }
}
```

```
node ins(gps, xv: double)
  returns (x: double, alarm: bool)
  var k: int, px: double,
      xe: double when not alarm;
let
  k = 0 fby k + 1;
  alarm = (k >= 50);
  xe = euler(gps when not alarm,
             xv when not alarm);
  x = merge alarm (px when alarm) xe;
  px = 0. fby x;
tel
```

only introducing state

```
system ins {
  init k = 0, px = 0.;
  sub xe: euler;

  transition(gps, xv: double)
    returns (x: double, alarm: bool)
    var xe: double when not alarm;
  {
    next k = k + 1;
    alarm = (k >= 50);
    xe = euler<xe>(gps when not alarm,
                   xv when not alarm);
    x = merge alarm (px when alarm) xe;
    next px = x;
  }
}
```

NLustre

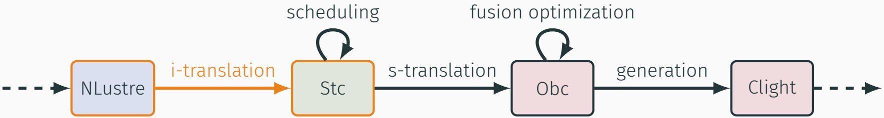
```
node ins(gps, xv: double)
  returns (x: double, alarm: bool)
  var k: int, px: double,
      xe: double when not alarm;
let
  k = 0 fby k + 1;
  alarm = (k >= 50);
  xe = euler(gps when not alarm,
             xv when not alarm);
  x = merge alarm (px when alarm) xe;
  px = 0. fby x;
tel
```

only introducing state

Stc

```
system ins {
  init k = 0, px = 0.;
  sub xe: euler;

  transition(gps, xv: double)
    returns (x: double, alarm: bool)
    var xe: double when not alarm;
  {
    next k = k + 1;
    alarm = (k >= 50);
    xe = euler<xe>(gps when not alarm,
                   xv when not alarm);
    x = merge alarm (px when alarm) xe;
    next px = x;
  }
}
```

Compilation of the Reset Example

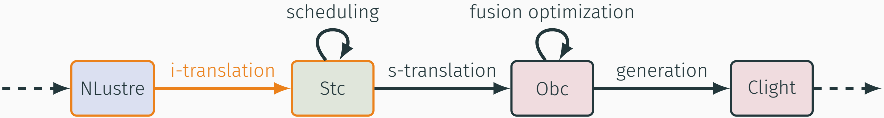



```
node driver(x0, y0, u, v: double, r: bool)
  returns (x, y: double)
  var ax, ay: bool;
let
  x, ax = (restart ins every r)(x0, u);
  y, ay = (restart ins every r)(y0, v);
tel
```

```
system driver {
  sub x: ins, y: ins;

  transition(x0, y0, u, v: double, r: bool)
    returns (x, y: double)
    var ax, ay: bool;
  {
    x, ax = ins<x>(x0, u);
    reset ins<x> every (. on r);
    y, ay = ins<y>(y0, v);
    reset ins<y> every (. on r);
  }
}
```

only introducing state

Compilation of the Reset Example

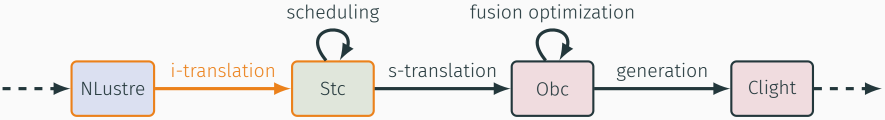



```
node driver(x0, y0, u, v: double, r: bool)
  returns (x, y: double)
  var ax, ay: bool;
let
  x, ax = (restart ins every r)(x0, u);
  y, ay = (restart ins every r)(y0, v);
tel
```

only introducing state

```
system driver {
  sub x: ins, y: ins;

  transition(x0, y0, u, v: double, r: bool)
    returns (x, y: double)
    var ax, ay: bool;
  {
    x, ax = ins<x>(x0, u);
    reset ins<x> every (. on r);
    y, ay = ins<y>(y0, v);
    reset ins<y> every (. on r);
  }
}
```

Compilation of the Reset Example

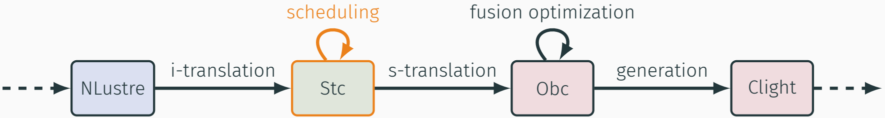



```
node driver(x0, y0, u, v: double, r: bool)
  returns (x, y: double)
  var ax, ay: bool;
let
  x, ax = (restart ins every r)(x0, u);
  y, ay = (restart ins every r)(y0, v);
tel
```

only introducing state

```
system driver {
  sub x: ins, y: ins;

  transition(x0, y0, u, v: double, r: bool)
    returns (x, y: double)
    var ax, ay: bool;
  {
    x, ax = ins<x>(x0, u);
    reset ins<x> every (. on r);
    y, ay = ins<y>(y0, v);
    reset ins<y> every (. on r);
  }
}
```

Compilation of the Reset Example




```
node driver(x0, y0, u, v: double, r: bool)
  returns (x, y: double)
  var ax, ay: bool;
let
  x, ax = (restart ins every r)(x0, u);
  y, ay = (restart ins every r)(y0, v);
tel
```

only scheduling

```
system driver {
  sub x: ins, y: ins;

  transition(x0, y0, u, v: double, r: bool)
    returns (x, y: double)
    var ax, ay: bool;
  {
    reset ins<x> every (. on r);
    reset ins<y> every (. on r);
    x, ax = ins<x>(x0, u);
    y, ay = ins<y>(y0, v);
  }
}
```

Stc Intuitive Semantics

Transition system

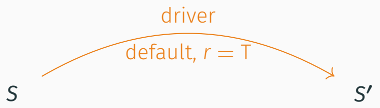
- Start state S, end state S'
- Transition constraints
- Transient state I

Stc Intuitive Semantics

Transition system

- Start state S, end state S'
- Transition constraints
- Transient state I

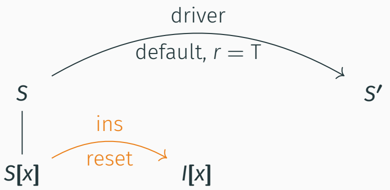
```
system driver {
  sub x: ins, y: ins;

  transition(x0, y0, u, v: double, r: bool)
    returns (x, y: double)
    var ax, ay: bool;
  {
    x, ax = ins<x>(x0, u);
    reset ins<x> every (. on r);
    y, ay = ins<y>(y0, v);
    reset ins<y> every (. on r);
  }
}
```



Transition system

- Start state S, end state S'
- Transition constraints
- Transient state I

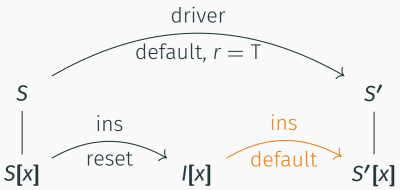
```
system driver {
  sub x: ins, y: ins;

  transition(x0, y0, u, v: double, r: bool)
    returns (x, y: double)
    var ax, ay: bool;
  {
    x, ax = ins<x>(x0, u);
    reset ins<x> every (. on r);
    y, ay = ins<y>(y0, v);
    reset ins<y> every (. on r);
  }
}
```



Transition system

- Start state S, end state S'
- Transition constraints
- Transient state I

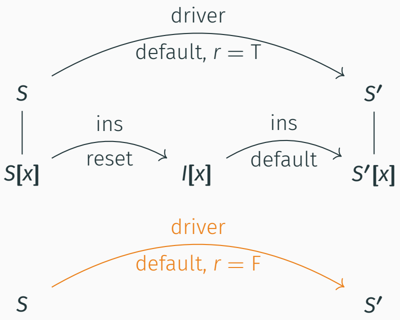
```
system driver {
  sub x: ins, y: ins;

  transition(x0, y0, u, v: double, r: bool)
    returns (x, y: double)
    var ax, ay: bool;
  {
    x, ax = ins<x>(x0, u);
    reset ins<x> every (. on r);
    y, ay = ins<y>(y0, v);
    reset ins<y> every (. on r);
  }
}
```



Transition system

- Start state S, end state S'
- Transition constraints
- Transient state I

```
system driver {
  sub x: ins, y: ins;

  transition(x0, y0, u, v: double, r: bool)
    returns (x, y: double)
    var ax, ay: bool;
  {
    x, ax = ins<x>(x0, u);
    reset ins<x> every (. on r);
    y, ay = ins<y>(y0, v);
    reset ins<y> every (. on r);
  }
}
```

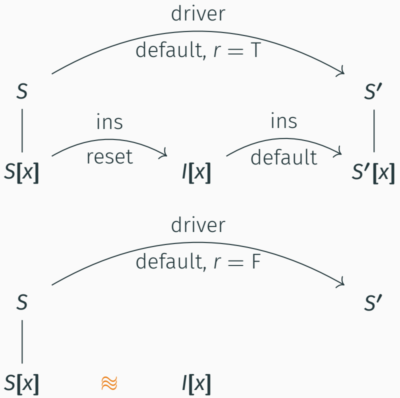


STC INTUITIVE SEMANTICS

Transition system

- Start state S, end state S'
- Transition constraints
- Transient state I

```
system driver {
  sub x: ins, y: ins;

  transition(x0, y0, u, v: double, r: bool)
    returns (x, y: double)
    var ax, ay: bool;
  {
    x, ax = ins<x>(x0, u);
    reset ins<x> every (. on r);
    y, ay = ins<y>(y0, v);
    reset ins<y> every (. on r);
  }
}
```

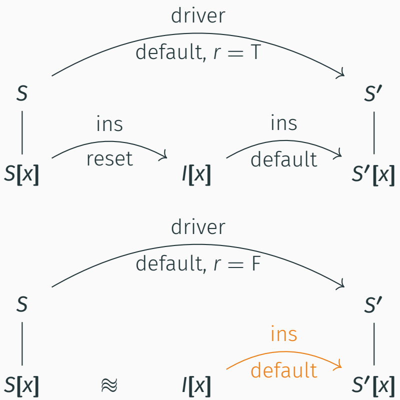


Stc Intuitive Semantics

Transition system

- Start state S, end state S'
- Transition constraints
- Transient state I

```
system driver {
  sub x: ins, y: ins;

  transition(x0, y0, u, v: double, r: bool)
    returns (x, y: double)
    var ax, ay: bool;
  {
    x, ax = ins<x>(x0, u);
    reset ins<x> every (. on r);
    y, ay = ins<y>(y0, v);
    reset ins<y> every (. on r);
  }
}
```



Basic transition constraint

$$\frac{R \vdash e \downarrow R(x)}{R, S, I, S' \vdash x \texttt{ = } e}$$

Next transition constraint

$$\frac{R \vdash e \downarrow \langle v \rangle \quad R(x) = \langle S(x) \rangle \quad S'(x) = v}{R, S, I, S' \vdash \texttt{next } x \texttt{ = } e} \qquad \frac{R \vdash e \downarrow \langle\, \rangle \quad R(x) = \langle\, \rangle \quad S'(x) = S(x)}{R, S, I, S' \vdash \texttt{next } x \texttt{ = } e}$$

Basic transition constraint

$$\frac{R \vdash e \downarrow R(x)}{R, S, I, S' \vdash x \texttt{ = } e}$$

Next transition constraint

$$\frac{R \vdash e \downarrow \langle v \rangle \quad R(x) = \langle S(x) \rangle \quad S'(x) = v}{R, S, I, S' \vdash \texttt{next } x \texttt{ = } e} \qquad \frac{R \vdash e \downarrow \langle\,\rangle \quad R(x) = \langle\,\rangle \quad S'(x) = S(x)}{R, S, I, S' \vdash \texttt{next } x \texttt{ = } e}$$

Basic transition constraint

$$\frac{R \vdash e \downarrow R(x)}{R, S, I, S' \vdash x \texttt{ = } e}$$

Next transition constraint

$$\frac{R \vdash e \downarrow \langle v \rangle \quad R(x) = \langle S(x) \rangle \quad S'(x) = v}{R, S, I, S' \vdash \texttt{next } x \texttt{ = } e} \qquad \frac{R \vdash e \downarrow \langle\, \rangle \quad R(x) = \langle\, \rangle \quad S'(x) = S(x)}{R, S, I, S' \vdash \texttt{next } x \texttt{ = } e}$$

Basic transition constraint

$$\frac{R \vdash e \downarrow R(x)}{R, S, I, S' \vdash x \texttt{ = } e}$$

Next transition constraint

$$\frac{R \vdash e \downarrow \langle v \rangle \quad R(x) = \langle S(x) \rangle \quad S'(x) = v}{R, S, I, S' \vdash \texttt{next } x \texttt{ = } e} \qquad \frac{R \vdash e \downarrow \langle\, \rangle \quad R(x) = \langle\, \rangle \quad S'(x) = S(x)}{R, S, I, S' \vdash \texttt{next } x \texttt{ = } e}$$

Basic transition constraint

$$\frac{R \vdash e \downarrow R(x)}{R, S, I, S' \vdash x \texttt{ = } e}$$

Next transition constraint

$$\frac{R \vdash e \downarrow \langle v \rangle \quad R(x) = \langle S(x) \rangle \quad S'(x) = v}{R, S, I, S' \vdash \texttt{next } x \texttt{ = } e} \qquad \frac{R \vdash e \downarrow \langle\,\rangle \quad R(x) = \langle\,\rangle \quad S'(x) = S(x)}{R, S, I, S' \vdash \texttt{next } x \texttt{ = } e}$$

Basic transition constraint

$$\frac{R \vdash e \downarrow R(x)}{R, S, I, S' \vdash x \texttt{ = } e}$$

Next transition constraint

$$\frac{R \vdash e \downarrow \langle v \rangle \quad R(x) = \langle S(x) \rangle \quad S'(x) = v}{R, S, I, S' \vdash \texttt{next } x \texttt{ = } e}$$

$$\frac{R \vdash e \downarrow \langle\,\rangle \quad R(x) = \langle\,\rangle \quad S'(x) = S(x)}{R, S, I, S' \vdash \texttt{next } x \texttt{ = } e}$$

Basic transition constraint

$$\frac{R \vdash e \downarrow R(x)}{R, S, I, S' \vdash x = e}$$

Next transition constraint

$$\frac{R \vdash e \downarrow \langle v \rangle \quad R(x) = \langle S(x) \rangle \quad S'(x) = v}{R, S, I, S' \vdash \texttt{next}\ x = e} \qquad \frac{R \vdash e \downarrow \langle\,\rangle \quad R(x) = \langle\,\rangle \quad S'(x) = S(x)}{R, S, I, S' \vdash \texttt{next}\ x = e}$$

Basic transition constraint

$$\frac{R \vdash e \downarrow R(x)}{R, S, I, S' \vdash x \texttt{ = } e}$$

Next transition constraint

$$\frac{R \vdash e \downarrow \langle v \rangle \quad R(x) = \langle S(x) \rangle \quad S'(x) = v}{R, S, I, S' \vdash \texttt{next } x \texttt{ = } e} \qquad \frac{R \vdash e \downarrow \langle\,\rangle \quad R(x) = \langle\,\rangle \quad S'(x) = S(x)}{R, S, I, S' \vdash \texttt{next } x \texttt{ = } e}$$

Default transition

$$\frac{R \vdash \boldsymbol{e} \downarrow \boldsymbol{v} \quad I[i], S'[i] \vdash f(\boldsymbol{v}) \Downarrow R(\boldsymbol{x}) \quad \text{if } (k = 0) \text{ then } I[i] \approx S[i]}{R, S, I, S' \vdash \boldsymbol{x} = f\texttt{<}i, k\texttt{>}(\boldsymbol{e})}$$

Reset transition

$$\frac{R \vdash ck \downarrow \text{true} \quad \text{initial-state}\, f\ I[i]}{R, S, I, S' \vdash \texttt{reset}\ f\texttt{<}i\texttt{>}\ \texttt{every}\ ck}
\qquad
\frac{R \vdash ck \downarrow \text{false} \quad I[i] \approx S[i]}{R, S, I, S' \vdash \texttt{reset}\ f\texttt{<}i\texttt{>}\ \texttt{every}\ ck}$$

Default transition

$$\frac{R \vdash e \downarrow v \quad I[i], S'[i] \vdash f(v) \Downarrow R(x) \quad \text{if } (k = 0) \text{ then } I[i] \approx S[i]}{R, S, I, S' \vdash x = f\texttt{<}i, k\texttt{>}(e)}$$

Reset transition

$$\frac{R \vdash ck \downarrow \text{true} \quad \text{initial-state}\, f\; I[i]}{R, S, I, S' \vdash \texttt{reset}\; f\texttt{<}i\texttt{>}\; \texttt{every}\; ck}
\qquad
\frac{R \vdash ck \downarrow \text{false} \quad I[i] \approx S[i]}{R, S, I, S' \vdash \texttt{reset}\; f\texttt{<}i\texttt{>}\; \texttt{every}\; ck}$$

Default transition

$$\frac{R \vdash \boldsymbol{e} \downarrow \boldsymbol{v} \quad I[i]\,,S'[i] \vdash f\,(\boldsymbol{v}) \Downarrow R(\boldsymbol{x}) \quad \text{if } (k = 0) \text{ then } I[i] \approx S[i]}{R, S, I, S' \vdash \boldsymbol{x} = f\texttt{<}i, k\texttt{>}(\boldsymbol{e})}$$

Reset transition

$$\frac{R \vdash ck \downarrow \text{true} \quad \text{initial-state}\, f\ I[i]}{R, S, I, S' \vdash \texttt{reset}\ f\texttt{<}i\texttt{>}\ \texttt{every}\ ck}$$

$$\frac{R \vdash ck \downarrow \text{false} \quad I[i] \approx S[i]}{R, S, I, S' \vdash \texttt{reset}\ f\texttt{<}i\texttt{>}\ \texttt{every}\ ck}$$

Default transition

$$\frac{R \vdash e \downarrow v \quad I[i], S'[i] \vdash f(v) \Downarrow R(x) \qquad \text{if } (k = 0) \text{ then } I[i] \approx S[i]}{R, S, I, S' \vdash x = f\texttt{<}i, k\texttt{>}(e)}$$

Reset transition

$$\frac{R \vdash ck \downarrow \text{true} \quad \text{initial-state}\, f\ I[i]}{R, S, I, S' \vdash \texttt{reset}\ f\texttt{<}i\texttt{>}\ \texttt{every}\ ck}
\qquad
\frac{R \vdash ck \downarrow \text{false} \quad I[i] \approx S[i]}{R, S, I, S' \vdash \texttt{reset}\ f\texttt{<}i\texttt{>}\ \texttt{every}\ ck}$$

Default transition

$$\frac{R \vdash \boldsymbol{e} \downarrow v \quad I[i]\,,S'[i] \vdash f\,(v) \Downarrow R(x) \quad \text{if } (k = 0) \text{ then } I[i] \approx S[i]}{R, S, I, S' \vdash \mathbf{x} = f\texttt{<}i, k\texttt{>}(\,\boldsymbol{e}\,)}$$

Reset transition

$$\frac{R \vdash ck \downarrow \text{true} \quad \text{initial-state}\, f\; I[i]}{R, S, I, S' \vdash \texttt{reset}\; f\texttt{<}i\texttt{>}\; \texttt{every}\; ck}$$

$$\frac{R \vdash ck \downarrow \text{false} \quad I[i] \approx S[i]}{R, S, I, S' \vdash \texttt{reset}\; f\texttt{<}i\texttt{>}\; \texttt{every}\; ck}$$

Default transition

$$\frac{R \vdash \boldsymbol{e} \downarrow \boldsymbol{v} \quad I[i]\,, S'[i] \vdash f\,(\boldsymbol{v}) \Downarrow R(\boldsymbol{x}) \quad \text{if}\ \ (k = 0)\ \ \text{then}\ I[i] \approx S[i]}{R, S, I, S' \vdash \boldsymbol{x}\ \texttt{=}\ f\texttt{<}i, k\texttt{>}\texttt{(}\,\boldsymbol{e}\,\texttt{)}}$$

Reset transition

$$\frac{R \vdash ck \downarrow \text{true} \quad \text{initial-state}\,f\ I[i]}{R, S, I, S' \vdash \texttt{reset}\ f\texttt{<}i\texttt{>}\ \texttt{every}\ ck}$$

$$\frac{R \vdash ck \downarrow \text{false} \quad I[i] \approx S[i]}{R, S, I, S' \vdash \texttt{reset}\ f\texttt{<}i\texttt{>}\ \texttt{every}\ ck}$$

Default transition

$$\frac{R \vdash \boldsymbol{e} \downarrow \boldsymbol{v} \quad I[i]\,,S'[i] \vdash f\,(\boldsymbol{v}) \Downarrow R(\boldsymbol{x}) \qquad \text{if } \ (k = 0) \ \text{ then } I[i] \approx S[i]}{R, S, I, S' \vdash \boldsymbol{x} = f\texttt{<}i, k\texttt{>}(\,\boldsymbol{e}\,)}$$

Reset transition

$$\frac{R \vdash ck \downarrow \text{true} \quad \text{initial-state}\, f \ I[i]}{R, S, I, S' \vdash \texttt{reset}\ f\texttt{<}i\texttt{>}\ \texttt{every}\ ck}$$

$$\frac{R \vdash ck \downarrow \text{false} \quad I[i] \approx S[i]}{R, S, I, S' \vdash \texttt{reset}\ f\texttt{<}i\texttt{>}\ \texttt{every}\ ck}$$

Default transition

$$\frac{R \vdash \boldsymbol{e} \downarrow \boldsymbol{v} \quad I[i]\,,S'[i] \vdash f\,(\boldsymbol{v}) \Downarrow R(\boldsymbol{x}) \qquad \text{if } \ (k = 0) \ \text{ then } I[i] \approx S[i]}{R, S, I, S' \vdash \boldsymbol{x} \texttt{ = } f\texttt{<}i, k\texttt{>}\texttt{(}\,\boldsymbol{e}\,\texttt{)}}$$

Reset transition

$$\frac{R \vdash ck \downarrow \text{true} \quad \text{initial-state}\, f\ I[i]}{R, S, I, S' \vdash \texttt{reset } f\texttt{<}i\texttt{>} \texttt{ every } ck} \qquad \frac{R \vdash ck \downarrow \text{false} \quad I[i] \approx S[i]}{R, S, I, S' \vdash \texttt{reset } f\texttt{<}i\texttt{>} \texttt{ every } ck}$$

System

$$\frac{\text{system}(P, f) = s \quad R(s.\textbf{in}) = \textbf{\textit{xs}} \quad R(s.\textbf{out}) = \textbf{\textit{ys}} \qquad \forall tc \in s.\textbf{tcs}\ , R, S, I, S' \vdash tc}{S, S' \vdash f\,(\textbf{\textit{xs}}) \Downarrow \textbf{\textit{ys}}}$$

System

$$\frac{\textsf{system}(P, f) = s \quad R(s.\textsf{in}) = \boldsymbol{xs} \quad R(s.\textsf{out}) = \boldsymbol{ys} \qquad \forall tc \in s.\textbf{tcs}\ , R, S, I, S' \vdash tc}{S, S' \vdash f(\boldsymbol{xs}) \Downarrow \boldsymbol{ys}}$$

System

$$\frac{\text{system}(P, f) = s \quad R(s.\mathbf{in}) = \mathbf{xs} \quad R(s.\mathbf{out}) = \mathbf{ys} \qquad \forall tc \in s.\mathbf{tcs}\ , R, S, I, S' \vdash tc}{S, S' \vdash f\,(\mathbf{xs}) \Downarrow \mathbf{ys}}$$

System

$$\frac{\text{system}(P, f) = s \quad R(s.\mathbf{in}) = \mathbf{xs} \quad R(s.\mathbf{out}) = \mathbf{ys} \qquad \forall tc \in s.\mathbf{tcs}\ , R, S, I, S' \vdash tc}{S, S' \vdash f\,(\mathbf{xs}) \Downarrow \mathbf{ys}}$$

System

$$\frac{\text{system}(P, f) = s \quad R(s.\textbf{in}) = \textbf{\textit{xs}} \quad R(s.\textbf{out}) = \textbf{\textit{ys}} \\ \forall tc \in s.\textbf{tcs}\ , R, S, I, S' \vdash tc}{S, S' \vdash f\,(\textbf{\textit{xs}}) \Downarrow \textbf{\textit{ys}}}$$

Loop

$$\frac{S, S' \vdash f(xs_n) \downdownarrows ys_n \qquad S' \vdash f(xs) \overset{n+1}{\circlearrowleft} ys}{S \vdash f(xs) \overset{n}{\circlearrowleft} ys}$$

Loop

$$\frac{S, S' \vdash f(xs_n) \Downarrow ys_n \qquad S' \vdash f(xs) \overset{n+1}{\circlearrowleft} ys}{S \vdash f(xs) \overset{n}{\circlearrowleft} ys}$$

Stc Formal Semantics

Loop

$$\dfrac{S, S' \vdash f(xs_n) \downdownarrows ys_n \qquad \dfrac{S', S'' \vdash f(xs_{n+1}) \downdownarrows ys_{n+1} \qquad S'' \vdash f(xs) \overset{n+2}{\circlearrowleft} ys}{S' \vdash f(xs) \overset{n+1}{\circlearrowleft} ys}}{S \vdash f(xs) \overset{n}{\circlearrowleft} ys}$$

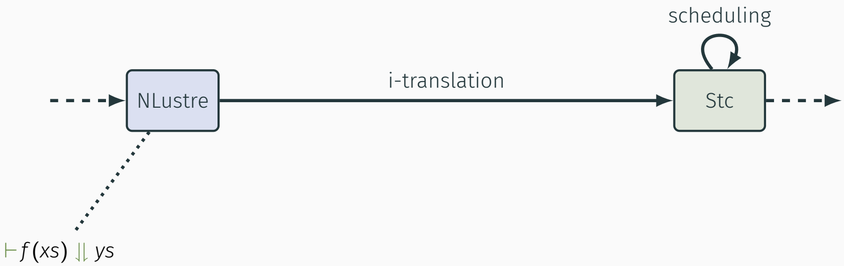
Loop

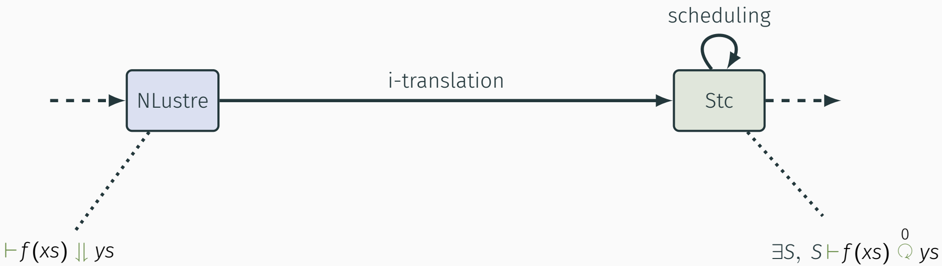
$$
\dfrac{S, S' \vdash f(xs_n) \Downarrow ys_n \qquad \dfrac{S', S'' \vdash f(xs_{n+1}) \Downarrow ys_{n+1} \qquad \dfrac{S'', S''' \vdash f(xs_{n+2}) \Downarrow ys_{n+2} \qquad S''' \vdash f(xs) \overset{n+3}{\circlearrowleft} ys}{S'' \vdash f(xs) \overset{n+2}{\circlearrowleft} ys}}{S' \vdash f(xs) \overset{n+1}{\circlearrowleft} ys}}{S \vdash f(xs) \overset{n}{\circlearrowleft} ys}
$$

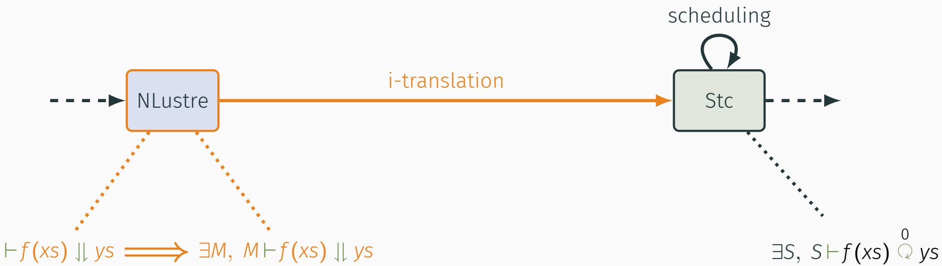
Loop

$$\dfrac{S, S' \vdash f(xs_n) \downdownarrows ys_n \qquad \dfrac{S', S'' \vdash f(xs_{n+1}) \downdownarrows ys_{n+1} \qquad \dfrac{S'', S''' \vdash f(xs_{n+2}) \downdownarrows ys_{n+2} \qquad \dfrac{\vdots}{S''' \vdash f(xs) \overset{n+3}{\circlearrowleft} ys}}{S'' \vdash f(xs) \overset{n+2}{\circlearrowleft} ys}}{S' \vdash f(xs) \overset{n+1}{\circlearrowleft} ys}}{S \vdash f(xs) \overset{n}{\circlearrowleft} ys}$$

Correctness: Preservation of the Semantics




scheduling
i-translation
NLustre
Stc
$\vdash f(xs) \Downarrow ys$
$\exists S,\ S \vdash f(xs) \overset{0}{\circlearrowleft} ys$

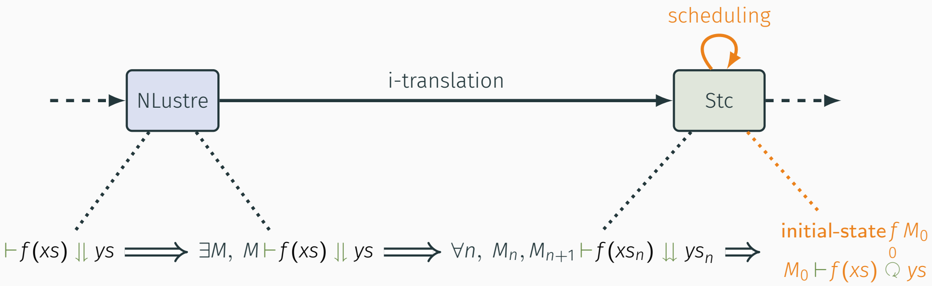

scheduling
NLustre
i-translation
Stc
$\vdash f(xs) \Downarrow ys \Longrightarrow \exists M,\ M \vdash f(xs) \Downarrow ys$
$\exists S,\ S \vdash f(xs) \overset{0}{\circlearrowleft} ys$

Correctness: Preservation of the Semantics

$$\vdash f(xs) \Downarrow ys \Longrightarrow \exists M,\ M \vdash f(xs) \Downarrow ys \Longrightarrow \forall n,\ M_n, M_{n+1} \vdash f(xs_n) \Downarrow ys_n \qquad \exists S,\ S \vdash f(xs) \overset{0}{\circlearrowleft} ys$$

$\vdash f(xs) \Downarrow ys \implies \exists M,\ M \vdash f(xs) \Downarrow ys \implies \forall n,\ M_n, M_{n+1} \vdash f(xs_n) \Downarrow ys_n \implies$ **initial-state** $f\ M_0$ / $M_0 \vdash f(xs) \overset{0}{\circlearrowleft} ys$

Correctness: Preservation of the Semantics



Producing Imperative Code: From Stc to Obc

Stc

```
system ins {
  init k = 0, px = 0.;
  sub xe: euler;

  transition(gps, xv: double)
    returns (x: double, alarm: bool)
    var xe: double when not alarm;
  {
    alarm = (k >= 50);
    next k = k + 1;
    xe = euler<xe>(gps when not alarm,
                   xv when not alarm);
    x = merge alarm (px when alarm) xe;
    next px = x;
  }
}
```

Obc

```
class ins {
  state k: int, px: double;
  instance xe: euler;

  reset() { state(k) := 0;
            state(px) := 0.;
            euler(xe).reset() }

  step(gps, xv: double)
    returns (x: double, alarm: bool)
    var xe: double
  {
    alarm := state(k) >= 50;
    state(k) := state(k) + 1;
    if alarm { }
    else { xe := euler(xe).step(gps, xv) };
    if alarm { x := state(px) }
    else { x := xe };
    state(px) := x
  }
```

Stc

```
system ins {
  init k = 0, px = 0.;
  sub xe: euler;

  transition(gps, xv: double)
    returns (x: double, alarm: bool)
    var xe: double when not alarm;
  {
    alarm = (k >= 50);
    next k = k + 1;
    xe = euler<xe>(gps when not alarm,
                   xv when not alarm);
    x = merge alarm (px when alarm) xe;
    next px = x;
  }
}
```

Obc

```
class ins {
  state k: int, px: double;
  instance xe: euler;

  reset() { state(k) := 0;
            state(px) := 0.;
            euler(xe).reset() }

  step(gps, xv: double)
    returns (x: double, alarm: bool)
    var xe: double
  {
    alarm := state(k) >= 50;
    state(k) := state(k) + 1;
    if alarm { }
    else { xe := euler(xe).step(gps, xv) };
    if alarm { x := state(px) }
    else { x := xe };
    state(px) := x
  }
```

Stc Obc

```
system ins {
  init k = 0, px = 0.;
  sub xe: euler;

  transition(gps, xv: double)
    returns (x: double, alarm: bool)
    var xe: double when not alarm;
  {
    alarm = (k >= 50);
    next k = k + 1;
    xe = euler<xe>(gps when not alarm,
                   xv when not alarm);
    x = merge alarm (px when alarm) xe;
    next px = x;
  }
}
```

```
class ins {
  state k: int, px: double;
  instance xe: euler;

  reset() { state(k) := 0;
            state(px) := 0.;
            euler(xe).reset() }

  step(gps, xv: double)
    returns (x: double, alarm: bool)
    var xe: double
  {
    alarm := state(k) >= 50;
    state(k) := state(k) + 1;
    if alarm { }
    else { xe := euler(xe).step(gps, xv) };
    if alarm { x := state(px) }
    else { x := xe };
    state(px) := x
  }
```

```
system ins {
  init k = 0, px = 0.;
  sub xe: euler;

  transition(gps, xv: double)
    returns (x: double, alarm: bool)
    var xe: double when not alarm;
  {
    alarm = (k >= 50);
    next k = k + 1;
    xe = euler<xe>(gps when not alarm,
                   xv when not alarm);
    x = merge alarm (px when alarm) xe;
    next px = x;
  }
}
```

```
class ins {
  state k: int, px: double;
  instance xe: euler;

  reset() { state(k) := 0;
            state(px) := 0.;
            euler(xe).reset() }

  step(gps, xv: double)
    returns (x: double, alarm: bool)
    var xe: double
  {
    alarm := state(k) >= 50;
    state(k) := state(k) + 1;
    if alarm { }
    else { xe := euler(xe).step(gps, xv) };
    if alarm { x := state(px) }
    else { x := xe };
    state(px) := x
  }
```

Stc

```
system ins {
  init k = 0, px = 0.;
  sub xe: euler;

  transition(gps, xv: double)
    returns (x: double, alarm: bool)
    var xe: double when not alarm;
  {
    alarm = (k >= 50);
    next k = k + 1;
    xe = euler<xe>(gps when not alarm,
                   xv when not alarm);
    x = merge alarm (px when alarm) xe;
    next px = x;
  }
}
```

Obc

```
class ins {
  state k: int, px: double;
  instance xe: euler;

  reset() { state(k) := 0;
            state(px) := 0.;
            euler(xe).reset() }

  step(gps, xv: double)
    returns (x: double, alarm: bool)
    var xe: double
  {
    alarm := state(k) >= 50;
    state(k) := state(k) + 1;
    if alarm { }
    else { xe := euler(xe).step(gps, xv) };
    if alarm { x := state(px) }
    else { x := xe };
    state(px) := x
  }
```

Stc

```
system ins {
  init k = 0, px = 0.;
  sub xe: euler;

  transition(gps, xv: double)
    returns (x: double, alarm: bool)
    var xe: double when not alarm;
  {
    alarm = (k >= 50);
    next k = k + 1;
    xe = euler<xe>(gps when not alarm,
                   xv when not alarm);
    x = merge alarm (px when alarm) xe;
    next px = x;
  }
}
```

Obc

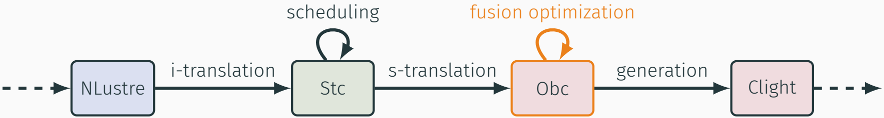
```
class ins {
  state k: int, px: double;
  instance xe: euler;

  reset() { state(k) := 0;
            state(px) := 0.;
            euler(xe).reset() }

  step(gps, xv: double)
    returns (x: double, alarm: bool)
    var xe: double
  {
    alarm := state(k) >= 50;
    state(k) := state(k) + 1;
    if alarm { }
    else { xe := euler(xe).step(gps, xv) };
    if alarm { x := state(px) }
    else { x := xe };
    state(px) := x
  }
}
```

Compilation of the Reset Example


```
system driver {
  sub x: ins, y: ins;

  transition(x0, y0, u, v: double, r: bool)
    returns (x, y: double)
    var ax, ay: bool;
  {
    reset ins<x> every (. on r);
    reset ins<y> every (. on r);
    x, ax = ins<x>(x0, u);
    y, ay = ins<y>(y0, v);
  }
}
```

```
class driver {
  instance x: ins, y: ins;

  reset() { ins(x).reset();
            ins(y).reset() }

  step(x0, y0, u, v: double, r: bool)
    returns (x, y: double)
    var ax, ay: bool
  {
    if r { ins(x).reset() };
    if r { ins(y).reset() };
    x, ax := ins(x).step(x0, u);
    y, ay := ins(y).step(y0, v)
  }
}
```

Compilation of the Reset Example




```
system driver {
  sub x: ins, y: ins;

  transition(x0, y0, u, v: double, r: bool)
    returns (x, y: double)
    var ax, ay: bool;
  {
    reset ins<x> every (. on r);
    reset ins<y> every (. on r);
    x, ax = ins<x>(x0, u);
    y, ay = ins<y>(y0, v);
  }
}
```

```
class driver {
  instance x: ins, y: ins;

  reset() { ins(x).reset();
            ins(y).reset() }

  step(x0, y0, u, v: double, r: bool)
    returns (x, y: double)
    var ax, ay: bool
  {
    if r { ins(x).reset();
           ins(y).reset() };
    x, ax := ins(x).step(x0, u);
    y, ay := ins(y).step(y0, v)
  }
}
```

Expressions

$$\frac{}{me, ve \vdash x \Downarrow ve(x)}$$

$$\frac{}{me, ve \vdash \texttt{state}(x) \Downarrow me(x)}$$

$$\frac{me, ve \vdash e_1 \Downarrow v_1 \quad me, ve \vdash e_2 \Downarrow v_2}{me, ve \vdash e_1 + e_2 \Downarrow [\![+]\!](v_1, v_2)}$$

Statements

$$\frac{me, ve \vdash e \Downarrow v}{me, ve \vdash x := e \Downarrow (me, ve\{x \mapsto v\})}$$

$$\frac{me, ve \vdash e \Downarrow v}{me, ve \vdash \texttt{state}(x) := e \Downarrow (me\{x \mapsto v\}, ve)}$$

$$\frac{me, ve \vdash s_1 \Downarrow (me_1, ve_1) \quad me_1, ve_1 \vdash s_2 \Downarrow (me_2, ve_2)}{me, ve \vdash s_1 \texttt{ ; } s_2 \Downarrow (me_2, ve_2)}$$

$$\frac{me, ve \vdash \mathbf{e} \Downarrow \mathbf{v} \quad me[i] \vdash c.f(\mathbf{v}) \overset{\mathbf{w}}{\Downarrow} me'_i}{me, ve \vdash \mathbf{x} := c(i).f(\mathbf{e}) \Downarrow (me\{i \mapsto me'_i\}, ve\{\mathbf{x} \mapsto \mathbf{w}\})}$$

Expressions

$$\frac{}{me, ve \vdash x \Downarrow ve(x)}$$

$$\frac{}{me, ve \vdash \texttt{state}(x) \Downarrow me(x)}$$

$$\frac{me, ve \vdash e_1 \Downarrow v_1 \quad me, ve \vdash e_2 \Downarrow v_2}{me, ve \vdash e_1 + e_2 \Downarrow [\![+]\!](v_1, v_2)}$$

Statements

$$\frac{me, ve \vdash e \Downarrow v}{me, ve \vdash x := e \Downarrow (me, ve\{x \mapsto v\})}$$

$$\frac{me, ve \vdash e \Downarrow v}{me, ve \vdash \texttt{state}(x) := e \Downarrow (me\{x \mapsto v\}, ve)}$$

$$\frac{\begin{array}{c} me, ve \vdash s_1 \Downarrow (me_1, ve_1) \\ me_1, ve_1 \vdash s_2 \Downarrow (me_2, ve_2) \end{array}}{me, ve \vdash s_1 \ \texttt{;} \ s_2 \Downarrow (me_2, ve_2)}$$

$$\frac{me, ve \vdash \mathbf{e} \Downarrow \mathbf{v} \quad me[i] \vdash c.f(\mathbf{v}) \overset{\mathbf{w}}{\Downarrow} me_i'}{me, ve \vdash \mathbf{x} := c(i).f(\mathbf{e}) \Downarrow (me\{i \mapsto me_i'\}, ve\{\mathbf{x} \mapsto \mathbf{w}\})}$$

Expressions

$$\frac{}{me, ve \vdash x \Downarrow ve(x)} \qquad \frac{}{me, ve \vdash \texttt{state}(x) \Downarrow me(x)} \qquad \frac{me, ve \vdash e_1 \Downarrow v_1 \quad me, ve \vdash e_2 \Downarrow v_2}{me, ve \vdash e_1 + e_2 \Downarrow [\![+]\!](v_1, v_2)}$$

Statements

$$\frac{me, ve \vdash e \Downarrow v}{me, ve \vdash x := e \Downarrow (me, ve\{x \mapsto v\})} \qquad \frac{me, ve \vdash e \Downarrow v}{me, ve \vdash \texttt{state}(x) := e \Downarrow (me\{x \mapsto v\}, ve)}$$

$$\frac{me, ve \vdash s_1 \Downarrow (me_1, ve_1) \quad me_1, ve_1 \vdash s_2 \Downarrow (me_2, ve_2)}{me, ve \vdash s_1 \,;\, s_2 \Downarrow (me_2, ve_2)} \qquad \frac{me, ve \vdash \mathbf{e} \Downarrow \mathbf{v} \quad me[i] \vdash c.f(\mathbf{v}) \overset{\mathbf{w}}{\Downarrow} me'_i}{me, ve \vdash \mathbf{x} := c(i).f(\mathbf{e}) \Downarrow (me\{i \mapsto me'_i\}, ve\{\mathbf{x} \mapsto \mathbf{w}\})}$$

Semantic Rules

Expressions

$$\frac{}{me, ve \vdash x \Downarrow ve(x)} \qquad \frac{}{me, ve \vdash \texttt{state}(x) \Downarrow me(x)} \qquad \frac{me, ve \vdash e_1 \Downarrow v_1 \quad me, ve \vdash e_2 \Downarrow v_2}{me, ve \vdash e_1 + e_2 \Downarrow [\![+]\!](v_1, v_2)}$$

Statements

$$\frac{me, ve \vdash e \Downarrow v}{me, ve \vdash x \mathrel{\texttt{:=}} e \Downarrow (me, ve\{x \mapsto v\})} \qquad \frac{me, ve \vdash e \Downarrow v}{me, ve \vdash \texttt{state}(x) \mathrel{\texttt{:=}} e \Downarrow (me\{x \mapsto v\}, ve)}$$

$$\frac{\begin{array}{c} me, ve \vdash s_1 \Downarrow (me_1, ve_1) \\ me_1, ve_1 \vdash s_2 \Downarrow (me_2, ve_2) \end{array}}{me, ve \vdash s_1 \texttt{ ; } s_2 \Downarrow (me_2, ve_2)} \qquad \frac{me, ve \vdash \mathbf{e} \Downarrow \mathbf{v} \quad me[i] \vdash c.f(\mathbf{v}) \overset{\mathbf{w}}{\Downarrow} me_i'}{me, ve \vdash \mathbf{x} \mathrel{\texttt{:=}} c(i)\texttt{.}f(\mathbf{e}) \Downarrow (me\{i \mapsto me_i'\}, ve\{\mathbf{x} \mapsto \mathbf{w}\})}$$

Expressions

$$\frac{}{me, ve \vdash x \Downarrow ve(x)} \qquad \frac{}{me, ve \vdash \texttt{state}(x) \Downarrow me(x)} \qquad \frac{me, ve \vdash e_1 \Downarrow v_1 \quad me, ve \vdash e_2 \Downarrow v_2}{me, ve \vdash e_1 + e_2 \Downarrow [\![+]\!](v_1, v_2)}$$

Statements

$$\frac{me, ve \vdash e \Downarrow v}{me, ve \vdash x := e \Downarrow (me, ve\{x \mapsto v\})} \qquad \frac{me, ve \vdash e \Downarrow v}{me, ve \vdash \texttt{state}(x) := e \Downarrow (me\{x \mapsto v\}, ve)}$$

$$\frac{\begin{array}{c} me, ve \vdash s_1 \Downarrow (me_1, ve_1) \\ me_1, ve_1 \vdash s_2 \Downarrow (me_2, ve_2) \end{array}}{me, ve \vdash s_1 \texttt{;} s_2 \Downarrow (me_2, ve_2)} \qquad \frac{me, ve \vdash \mathbf{e} \Downarrow \mathbf{v} \quad me[i] \vdash c.f(\mathbf{v}) \overset{\mathbf{w}}{\Downarrow} me_i'}{me, ve \vdash \mathbf{x} := c(i).f(\mathbf{e}) \Downarrow (me\{i \mapsto me_i'\}, ve\{\mathbf{x} \mapsto \mathbf{w}\})}$$

Semantic Rules

Expressions

$$\frac{}{me, ve \vdash x \Downarrow ve(x)} \qquad \frac{}{me, ve \vdash \texttt{state}(x) \Downarrow me(x)} \qquad \frac{me, ve \vdash e_1 \Downarrow v_1 \quad me, ve \vdash e_2 \Downarrow v_2}{me, ve \vdash e_1 + e_2 \Downarrow [\![+]\!](v_1, v_2)}$$

Statements

$$\frac{me, ve \vdash e \Downarrow v}{me, ve \vdash x \texttt{ := } e \Downarrow (me, ve\{x \mapsto v\})} \qquad \frac{me, ve \vdash e \Downarrow v}{me, ve \vdash \texttt{state}(x) \texttt{ := } e \Downarrow (me\{x \mapsto v\}, ve)}$$

$$\frac{me, ve \vdash s_1 \Downarrow (me_1, ve_1) \quad me_1, ve_1 \vdash s_2 \Downarrow (me_2, ve_2)}{me, ve \vdash s_1 \texttt{ ; } s_2 \Downarrow (me_2, ve_2)} \qquad \frac{me, ve \vdash \mathbf{e} \Downarrow \mathbf{v} \quad me[i] \vdash c.f(\mathbf{v}) \overset{\mathbf{w}}{\Downarrow} me_i'}{me, ve \vdash \mathbf{x} \texttt{ := } c\texttt{(}i\texttt{).}f\texttt{(}\mathbf{e}\texttt{)} \Downarrow (me\{i \mapsto me_i'\}, ve\{\mathbf{x} \mapsto \mathbf{w}\})}$$

Semantic Rules

Expressions

$$\frac{}{me, ve \vdash x \Downarrow ve(x)} \qquad \frac{}{me, ve \vdash \texttt{state}(x) \Downarrow me(x)} \qquad \frac{me, ve \vdash e_1 \Downarrow v_1 \quad me, ve \vdash e_2 \Downarrow v_2}{me, ve \vdash e_1 + e_2 \Downarrow [\![+]\!](v_1, v_2)}$$

Statements

$$\frac{me, ve \vdash e \Downarrow v}{me, ve \vdash x \texttt{ := } e \Downarrow (me, ve\{x \mapsto v\})} \qquad \frac{me, ve \vdash e \Downarrow v}{me, ve \vdash \texttt{state}(x) \texttt{ := } e \Downarrow (me\{x \mapsto v\}, ve)}$$

$$\frac{\begin{array}{c} me, ve \vdash s_1 \Downarrow (me_1, ve_1) \\ me_1, ve_1 \vdash s_2 \Downarrow (me_2, ve_2) \end{array}}{me, ve \vdash s_1 \texttt{ ; } s_2 \Downarrow (me_2, ve_2)} \qquad \frac{me, ve \vdash \mathbf{e} \Downarrow \mathbf{v} \quad me[i] \vdash c.f(\mathbf{v}) \overset{\mathbf{w}}{\Downarrow} me_i'}{me, ve \vdash \mathbf{x} \texttt{ := } c\texttt{(}i\texttt{).}f\texttt{(}\mathbf{e}\texttt{)} \Downarrow (me\{i \mapsto me_i'\}, ve\{\mathbf{x} \mapsto \mathbf{w}\})}$$

Expressions

$$\frac{}{me, ve \vdash x \Downarrow ve(x)}$$

$$\frac{}{me, ve \vdash \texttt{state}(x) \Downarrow me(x)}$$

$$\frac{me, ve \vdash e_1 \Downarrow v_1 \quad me, ve \vdash e_2 \Downarrow v_2}{me, ve \vdash e_1 + e_2 \Downarrow [\![+]\!](v_1, v_2)}$$

Statements

$$\frac{me, ve \vdash e \Downarrow v}{me, ve \vdash x \texttt{ := } e \Downarrow (me, ve\{x \mapsto v\})}$$

$$\frac{me, ve \vdash e \Downarrow v}{me, ve \vdash \texttt{state}(x) \texttt{ := } e \Downarrow (me\{x \mapsto v\}, ve)}$$

$$\frac{me, ve \vdash s_1 \Downarrow (me_1, ve_1) \quad me_1, ve_1 \vdash s_2 \Downarrow (me_2, ve_2)}{me, ve \vdash s_1 \texttt{ ; } s_2 \Downarrow (me_2, ve_2)}$$

$$\frac{me, ve \vdash \mathbf{e} \Downarrow \mathbf{v} \quad me[i] \vdash c.f(\mathbf{v}) \overset{\mathbf{w}}{\Downarrow} me_i'}{me, ve \vdash \mathbf{x} \texttt{ := } c\texttt{(}i\texttt{).}f\texttt{(}\mathbf{e}\texttt{)} \Downarrow (me\{i \mapsto me_i'\}, ve\{\mathbf{x} \mapsto \mathbf{w}\})}$$

Expressions

$$\frac{}{me, ve \vdash x \Downarrow ve(x)} \qquad \frac{}{me, ve \vdash \texttt{state}(x) \Downarrow me(x)} \qquad \frac{me, ve \vdash e_1 \Downarrow v_1 \quad me, ve \vdash e_2 \Downarrow v_2}{me, ve \vdash e_1 + e_2 \Downarrow [\![+]\!](v_1, v_2)}$$

Statements

$$\frac{me, ve \vdash e \Downarrow v}{me, ve \vdash x \texttt{ := } e \Downarrow (me, ve\{x \mapsto v\})} \qquad \frac{me, ve \vdash e \Downarrow v}{me, ve \vdash \texttt{state}(x) \texttt{ := } e \Downarrow (me\{x \mapsto v\}, ve)}$$

$$\frac{\begin{array}{c} me, ve \vdash s_1 \Downarrow (me_1, ve_1) \\ me_1, ve_1 \vdash s_2 \Downarrow (me_2, ve_2) \end{array}}{me, ve \vdash s_1 \texttt{ ; } s_2 \Downarrow (me_2, ve_2)} \qquad \frac{me, ve \vdash \mathbf{e} \Downarrow \mathbf{v} \quad me[i] \vdash c.f(\mathbf{v}) \overset{\mathbf{w}}{\Downarrow} me_i'}{me, ve \vdash \mathbf{x} \texttt{ := } c\texttt{(}i\texttt{)}\texttt{.}f\texttt{(}\mathbf{e}\texttt{)} \Downarrow (me\{i \mapsto me_i'\}, ve\{\mathbf{x} \mapsto \mathbf{w}\})}$$

Expressions

$$\frac{}{me, ve \vdash x \Downarrow ve(x)} \qquad \frac{}{me, ve \vdash \texttt{state}(x) \Downarrow me(x)} \qquad \frac{me, ve \vdash e_1 \Downarrow v_1 \quad me, ve \vdash e_2 \Downarrow v_2}{me, ve \vdash e_1 + e_2 \Downarrow [\![+]\!](v_1, v_2)}$$

Statements

$$\frac{me, ve \vdash e \Downarrow v}{me, ve \vdash x := e \Downarrow (me, ve\{x \mapsto v\})} \qquad \frac{me, ve \vdash e \Downarrow v}{me, ve \vdash \texttt{state}(x) := e \Downarrow (me\{x \mapsto v\}, ve)}$$

$$\frac{me, ve \vdash s_1 \Downarrow (me_1, ve_1) \quad me_1, ve_1 \vdash s_2 \Downarrow (me_2, ve_2)}{me, ve \vdash s_1 \texttt{;}\ s_2 \Downarrow (me_2, ve_2)} \qquad \frac{me, ve \vdash \mathbf{e} \Downarrow \mathbf{v} \quad me[i] \vdash c.f(\mathbf{v}) \overset{\mathbf{w}}{\Downarrow} me_i'}{me, ve \vdash \mathbf{x} := c(i).f(\mathbf{e}) \Downarrow (me\{i \mapsto me_i'\}, ve\{\mathbf{x} \mapsto \mathbf{w}\})}$$

Expressions

$$\frac{}{me, ve \vdash x \Downarrow ve(x)} \qquad \frac{}{me, ve \vdash \texttt{state}(x) \Downarrow me(x)} \qquad \frac{me, ve \vdash e_1 \Downarrow v_1 \quad me, ve \vdash e_2 \Downarrow v_2}{me, ve \vdash e_1 + e_2 \Downarrow [\![+]\!](v_1, v_2)}$$

Statements

$$\frac{me, ve \vdash e \Downarrow v}{me, ve \vdash x \texttt{ := } e \Downarrow (me, ve\{x \mapsto v\})} \qquad \frac{me, ve \vdash e \Downarrow v}{me, ve \vdash \texttt{state}(x) \texttt{ := } e \Downarrow (me\{x \mapsto v\}, ve)}$$

$$\frac{\begin{array}{c} me, ve \vdash s_1 \Downarrow (me_1, ve_1) \\ me_1, ve_1 \vdash s_2 \Downarrow (me_2, ve_2) \end{array}}{me, ve \vdash s_1 \texttt{ ; } s_2 \Downarrow (me_2, ve_2)} \qquad \frac{me, ve \vdash \mathbf{e} \Downarrow \mathbf{v} \quad me[i] \vdash c.f(\mathbf{v}) \overset{w}{\Downarrow} me'_i}{me, ve \vdash \mathbf{x} \texttt{ := } c(i)\texttt{.}f(\mathbf{e}) \Downarrow (me\{i \mapsto me'_i\}, ve\{\mathbf{x} \mapsto \mathbf{w}\})}$$

Expressions

$$\frac{}{me, ve \vdash x \Downarrow ve(x)} \qquad \frac{}{me, ve \vdash \texttt{state}(x) \Downarrow me(x)} \qquad \frac{me, ve \vdash e_1 \Downarrow v_1 \quad me, ve \vdash e_2 \Downarrow v_2}{me, ve \vdash e_1 + e_2 \Downarrow [\![+]\!](v_1, v_2)}$$

Statements

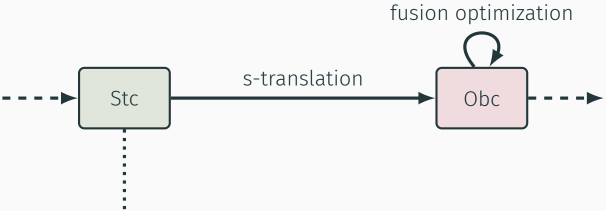
$$\frac{me, ve \vdash e \Downarrow v}{me, ve \vdash x \texttt{ := } e \Downarrow (me, ve\{x \mapsto v\})} \qquad \frac{me, ve \vdash e \Downarrow v}{me, ve \vdash \texttt{state}(x) \texttt{ := } e \Downarrow (me\{x \mapsto v\}, ve)}$$

$$\frac{\begin{array}{c} me, ve \vdash s_1 \Downarrow (me_1, ve_1) \\ me_1, ve_1 \vdash s_2 \Downarrow (me_2, ve_2) \end{array}}{me, ve \vdash s_1 \texttt{ ; } s_2 \Downarrow (me_2, ve_2)} \qquad \frac{me, ve \vdash \boldsymbol{e} \Downarrow \mathbf{v} \quad me[i] \vdash c.f(\mathbf{v}) \overset{\mathbf{w}}{\Downarrow} me_i'}{me, ve \vdash \mathbf{x} \texttt{ := } c(i)\texttt{.}f(\boldsymbol{e}) \Downarrow (me\{i \mapsto me_i'\}, ve\{\mathbf{x} \mapsto \mathbf{w}\})}$$

Loop

$$\frac{me \vdash c.f(xs_n) \overset{ys_n}{\Downarrow} me' \quad me' \vdash c.f(xs) \overset{n+1}{\circlearrowleft} ys}{me \vdash c.f(xs) \overset{n}{\circlearrowleft} ys}$$

Correctness



initial-state $f\ S$

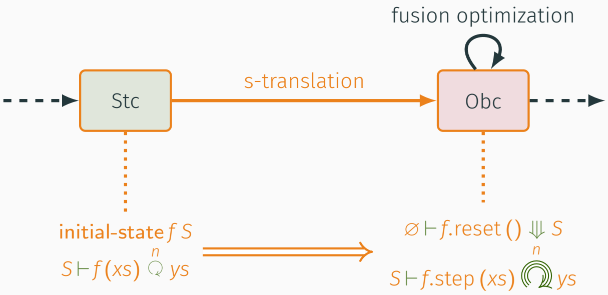
$S \vdash f(xs) \overset{n}{\circlearrowleft} ys$

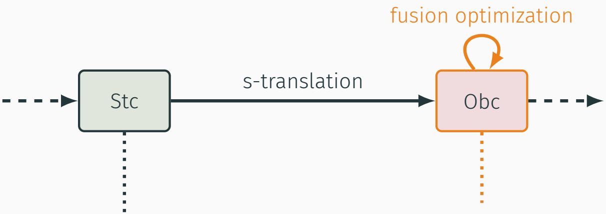
CORRECTNESS

initial-state $f\ S$

$S \vdash f\,(xs) \overset{n}{\circlearrowleft} ys$

$\exists me,\ me \vdash f.\mathsf{step}\,(xs) \overset{n}{\circlearrowleft} ys$


fusion optimization
Stc
s-translation
Obc
initial-state f S
S ⊢ f(xs) n ys
∅ ⊢ f.reset () ⇓ S
n
S ⊢ f.step (xs) ys



$$\frac{\text{initial-state}\ f\ S \qquad S \vdash f\,(xs) \overset{n}{\circlearrowleft} ys}{} \implies \frac{\emptyset \vdash f.\text{reset}\,() \Downarrow S \qquad S \vdash f.\text{step}\,(xs) \overset{n}{\circlearrowleft} ys}{}$$

Generating Clight code

CompCert

Mechanization in Coq of the syntax, the semantics and the compilation algorithms of the C language.

Clight

- CompCert intermediate language
- very similar to C
- low-level operations (addresses, structures,...)

Obc

```
class ins {
  state k: int, px: double;
  instance xe: euler;

  reset() { state(k) := 0;
            state(px) := 0.;
            euler(xe).reset() }

  step(gps, xv: double)
    returns (x: double, alarm: bool)
    var xe: double
  {
    alarm := state(k) >= 50;
    state(k) := state(k) + 1;
    if alarm { x := state(px) }
    else {
      xe := euler(xe).step(gps, xv);
      x := xe };
    state(px) := x
  }
```

Clight

```
struct ins {
  int k;
  double px;
  struct euler xe;
};

void fun$ins$reset(struct ins *self) {
  self->k = 0;
  self->px = 0;
  fun$euler$reset(&(self->xe));
  return;
}
```

Obc

```
class ins {
  state k: int, px: double;
  instance xe: euler;

  reset() { state(k) := 0;
            state(px) := 0.;
            euler(xe).reset() }

  step(gps, xv: double)
    returns (x: double, alarm: bool)
    var xe: double
  {
    alarm := state(k) >= 50;
    state(k) := state(k) + 1;
    if alarm { x := state(px) }
    else {
      xe := euler(xe).step(gps, xv);
      x := xe };
    state(px) := x
  }
}
```

Clight

```
struct ins {
  int k;
  double px;
  struct euler xe;
};

void fun$ins$reset(struct ins *self) {
  self->k = 0;
  self->px = 0;
  fun$euler$reset(&(self->xe));
  return;
}
```

Obc

```
class ins {
  state k: int, px: double;
  instance xe: euler;

  reset() { state(k) := 0;
            state(px) := 0.;
            euler(xe).reset() }

  step(gps, xv: double)
    returns (x: double, alarm: bool)
    var xe: double
  {
    alarm := state(k) >= 50;
    state(k) := state(k) + 1;
    if alarm { x := state(px) }
    else {
      xe := euler(xe).step(gps, xv);
      x := xe };
    state(px) := x
  }
```

Clight

```
struct ins {
  int k;
  double px;
  struct euler xe;
};

void fun$ins$reset(struct ins *self) {
  self->k = 0;
  self->px = 0;
  fun$euler$reset(&(self->xe));
  return;
}
```

Obc

```
class ins {
  state k: int, px: double;
  instance xe: euler;

  reset() { state(k) := 0;
            state(px) := 0.;
            euler(xe).reset() }

  step(gps, xv: double)
    returns (x: double, alarm: bool)
    var xe: double
  {
    alarm := state(k) >= 50;
    state(k) := state(k) + 1;
    if alarm { x := state(px) }
    else {
      xe := euler(xe).step(gps, xv);
      x := xe };
    state(px) := x
  }
```

Clight

```
struct fun$ins$step {
  double x;
  bool alarm;
};
void fun$ins$step(struct ins *self,
                  struct fun$ins$step *out,
                  double gps, double xv) {
  register double step$x;
  register double xe;
  out->alarm = self->k >= 50;
  self->k = self->k + 1;
  if (out->alarm) { out->x = self->px; }
  else {
    step$x = fun$euler$step(&(self->xe), gps, xv);
    xe = step$x;
    out->x = xe;
  }
  self->px = out->x;
  return;
}
```

Obc

```
class ins {
  state k: int, px: double;
  instance xe: euler;

  reset() { state(k) := 0;
            state(px) := 0.;
            euler(xe).reset() }

  step(gps, xv: double)
    returns (x: double, alarm: bool)
    var xe: double
  {
    alarm := state(k) >= 50;
    state(k) := state(k) + 1;
    if alarm { x := state(px) }
    else {
      xe := euler(xe).step(gps, xv);
      x := xe };
    state(px) := x
  }
```

Clight

```
struct fun$ins$step {
  double x;
  bool alarm;
};
void fun$ins$step(struct ins *self,
                  struct fun$ins$step *out,
                  double gps, double xv) {
  register double step$x;
  register double xe;
  out->alarm = self->k >= 50;
  self->k = self->k + 1;
  if (out->alarm) { out->x = self->px; }
  else {
    step$x = fun$euler$step(&(self->xe), gps, xv);
    xe = step$x;
    out->x = xe;
  }
  self->px = out->x;
  return;
}
```

Main Loop

```
struct nav {
  bool c;
  bool r;
  struct ins insr;
};

struct fun$nav$step {
  double x;
  bool alarm;
};

struct nav self$;
double volatile gps$;
double volatile xv$;
bool volatile s$;
double volatile x$;
bool volatile alarm$;
```

```
int main(void) {
  struct fun$nav$step out$step;
  register double gps;
  register double xv;
  register bool s;

  fun$nav$reset(&self$);

  while (true) {
    gps = volatile_load(&gps$);
    xv = volatile_load(&xv$);
    s = volatile_load(&s$);

    fun$nav$step(&self$, &out$step, gps, xv, s);

    volatile_store(&x$, out$step.x);
    volatile_store(&alarm$, out$step.alarm);
  }
}
```

Main Loop

```
struct nav {
  bool c;
  bool r;
  struct ins insr;
};

struct fun$nav$step {
  double x;
  bool alarm;
};

struct nav self$;
double volatile gps$;
double volatile xv$;
bool volatile s$;
double volatile x$;
bool volatile alarm$;
```

```
int main(void) {
  struct fun$nav$step out$step;
  register double gps;
  register double xv;
  register bool s;

  fun$nav$reset(&self$);

  while (true) {
    gps = volatile_load(&gps$);
    xv = volatile_load(&xv$);
    s = volatile_load(&s$);

    fun$nav$step(&self$, &out$step, gps, xv, s);

    volatile_store(&x$, out$step.x);
    volatile_store(&alarm$, out$step.alarm);
  }
}
```

Main Loop

```c
struct nav {
  bool c;
  bool r;
  struct ins insr;
};

struct fun$nav$step {
  double x;
  bool alarm;
};

struct nav self$;
double volatile gps$;
double volatile xv$;
bool volatile s$;
double volatile x$;
bool volatile alarm$;
```

```c
int main(void) {
  struct fun$nav$step out$step;
  register double gps;
  register double xv;
  register bool s;

  fun$nav$reset(&self$);

  while (true) {
    gps = volatile_load(&gps$);
    xv = volatile_load(&xv$);
    s = volatile_load(&s$);

    fun$nav$step(&self$, &out$step, gps, xv, s);

    volatile_store(&x$, out$step.x);
    volatile_store(&alarm$, out$step.alarm);
  }
}
```

Main Loop

```
struct nav {
  bool c;
  bool r;
  struct ins insr;
};

struct fun$nav$step {
  double x;
  bool alarm;
};

struct nav self$;
double volatile gps$;
double volatile xv$;
bool volatile s$;
double volatile x$;
bool volatile alarm$;
```

```
int main(void) {
  struct fun$nav$step out$step;
  register double gps;
  register double xv;
  register bool s;

  fun$nav$reset(&self$);

  while (true) {
    gps = volatile_load(&gps$);
    xv = volatile_load(&xv$);
    s = volatile_load(&s$);

    fun$nav$step(&self$, &out$step, gps, xv, s);

    volatile_store(&x$, out$step.x);
    volatile_store(&alarm$, out$step.alarm);
  }
}
```

Main Loop

```c
struct nav {
  bool c;
  bool r;
  struct ins insr;
};

struct fun$nav$step {
  double x;
  bool alarm;
};

struct nav self$;
double volatile gps$;
double volatile xv$;
bool volatile s$;
double volatile x$;
bool volatile alarm$;
```

```c
int main(void) {
  struct fun$nav$step out$step;
  register double gps;
  register double xv;
  register bool s;

  fun$nav$reset(&self$);

  while (true) {
    gps = volatile_load(&gps$);
    xv = volatile_load(&xv$);
    s = volatile_load(&s$);

    fun$nav$step(&self$, &out$step, gps, xv, s);

    volatile_store(&x$, out$step.x);
    volatile_store(&alarm$, out$step.alarm);
  }
}
```

Main Loop

```
struct nav {
  bool c;
  bool r;
  struct ins insr;
};

struct fun$nav$step {
  double x;
  bool alarm;
};

struct nav self$;
double volatile gps$;
double volatile xv$;
bool volatile s$;
double volatile x$;
bool volatile alarm$;
```

```
int main(void) {
  struct fun$nav$step out$step;
  register double gps;
  register double xv;
  register bool s;

  fun$nav$reset(&self$);

  while (true) {
    gps = volatile_load(&gps$);
    xv = volatile_load(&xv$);
    s = volatile_load(&s$);

    fun$nav$step(&self$, &out$step, gps, xv, s);

    volatile_store(&x$, out$step.x);
    volatile_store(&alarm$, out$step.alarm);
  }
}
```

```
struct nav {
  bool c;
  bool r;
  struct ins insr;
};

struct fun$nav$step {
  double x;
  bool alarm;
};

struct nav self$;
double volatile gps$;
double volatile xv$;
bool volatile s$;
double volatile x$;
bool volatile alarm$;
```

```
int main(void) {
  struct fun$nav$step out$step;
  register double gps;
  register double xv;
  register bool s;

  fun$nav$reset(&self$);

  while (true) {
    gps = volatile_load(&gps$);
    xv = volatile_load(&xv$);
    s = volatile_load(&s$);

    fun$nav$step(&self$, &out$step, gps, xv, s);

    volatile_store(&x$, out$step.x);
    volatile_store(&alarm$, out$step.alarm);
  }
}
```

Main Loop

```
struct nav {
  bool c;
  bool r;
  struct ins insr;
};

struct fun$nav$step {
  double x;
  bool alarm;
};

struct nav self$;
double volatile gps$;
double volatile xv$;
bool volatile s$;
double volatile x$;
bool volatile alarm$;
```

```
int main(void) {
  struct fun$nav$step out$step;
  register double gps;
  register double xv;
  register bool s;

  fun$nav$reset(&self$);

  while (true) {
    gps = volatile_load(&gps$);
    xv = volatile_load(&xv$);
    s = volatile_load(&s$);

    fun$nav$step(&self$, &out$step, gps, xv, s);

    volatile_store(&x$, out$step.x);
    volatile_store(&alarm$, out$step.alarm);
  }
}
```

Main Loop

```
struct nav {
  bool c;
  bool r;
  struct ins insr;
};

struct fun$nav$step {
  double x;
  bool alarm;
};

struct nav self$;
double volatile gps$;
double volatile xv$;
bool volatile s$;
double volatile x$;
bool volatile alarm$;
```

```
int main(void) {
  struct fun$nav$step out$step;
  register double gps;
  register double xv;
  register bool s;

  fun$nav$reset(&self$);

  while (true) {
    gps = volatile_load(&gps$);
    xv = volatile_load(&xv$);
    s = volatile_load(&s$);

    fun$nav$step(&self$, &out$step, gps, xv, s);

    volatile_store(&x$, out$step.x);
    volatile_store(&alarm$, out$step.alarm);
  }
}
```

Main Loop

```c
struct nav {
  bool c;
  bool r;
  struct ins insr;
};

struct fun$nav$step {
  double x;
  bool alarm;
};

struct nav self$;
double volatile gps$;
double volatile xv$;
bool volatile s$;
double volatile x$;
bool volatile alarm$;
```

```c
int main(void) {
  struct fun$nav$step out$step;
  register double gps;
  register double xv;
  register bool s;

  fun$nav$reset(&self$);

  while (true) {
    gps = volatile_load(&gps$);
    xv = volatile_load(&xv$);
    s = volatile_load(&s$);

    fun$nav$step(&self$, &out$step, gps, xv, s);

    volatile_store(&x$, out$step.x);
    volatile_store(&alarm$, out$step.alarm);
  }
}
```

Clight Semantic Model

- contiguous blocks memory model
- variables and registers
- semantic state (E, L, M)

E	variable environment: maps identifiers to locations
L	register environment: maps identifiers to values
M	memory: maps locations to bytes

Correctness: Correspondence Between Obc and Clight

Consequences of CompCert's memory model

- aliasing
- alignment
- permissions
- type sizes

Manipulation of structures and pointers

Correctness: Correspondence Between Obc and Clight

Consequences of CompCert's memory model

- aliasing
- alignment
- permissions
- type sizes

Manipulation of structures and pointers

Solution: use Separation Logic assertions

Separation Logic

An extension of Hoare logic to reason about programs that manipulate pointers and structures.

Separation Logic

An extension of Hoare logic to reason about programs that manipulate pointers and structures.

The predicate $M \models P * Q$ asserts that M can be partitioned into two distinct areas on which P and Q hold respectively.

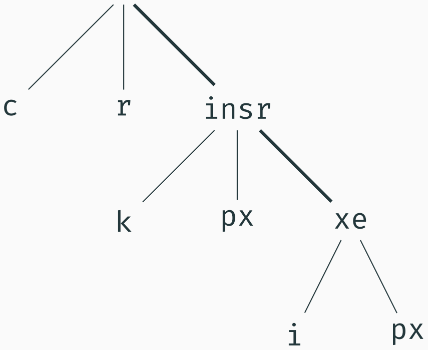
Separation Logic

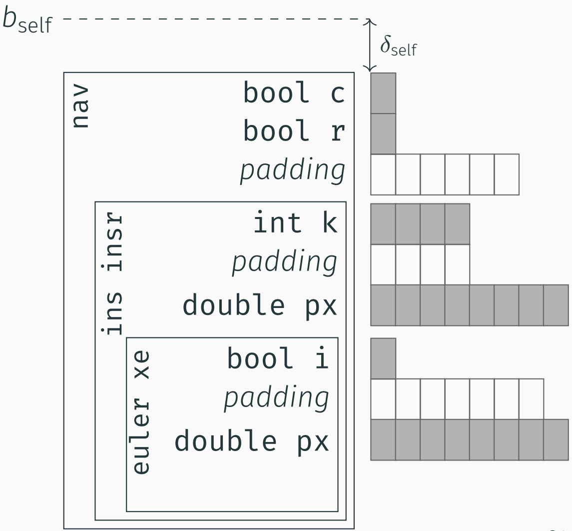
An extension of Hoare logic to reason about programs that manipulate pointers and structures.

The predicate $M \vDash P * Q$ asserts that M can be partitioned into two distinct areas on which P and Q hold respectively.

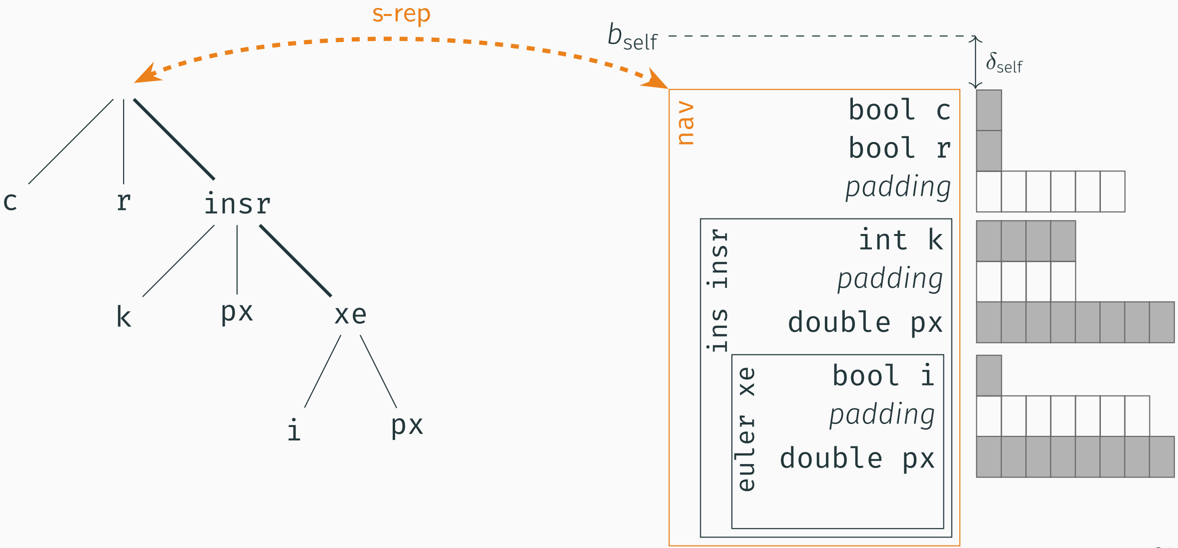
CompCert already uses a small Separation Logic library for one of its passes.

State Correspondence Predicate

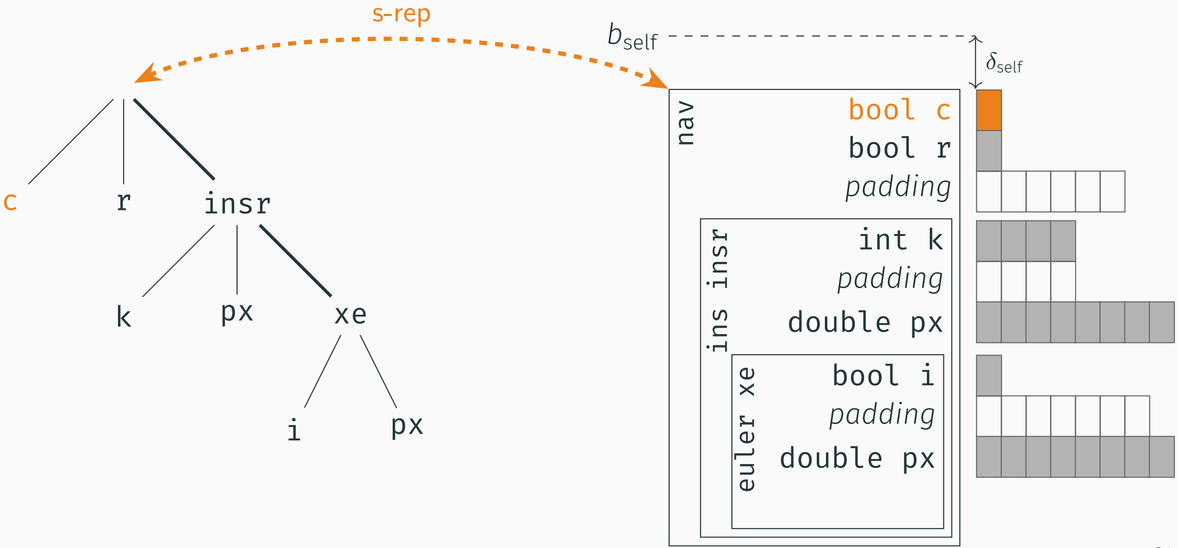




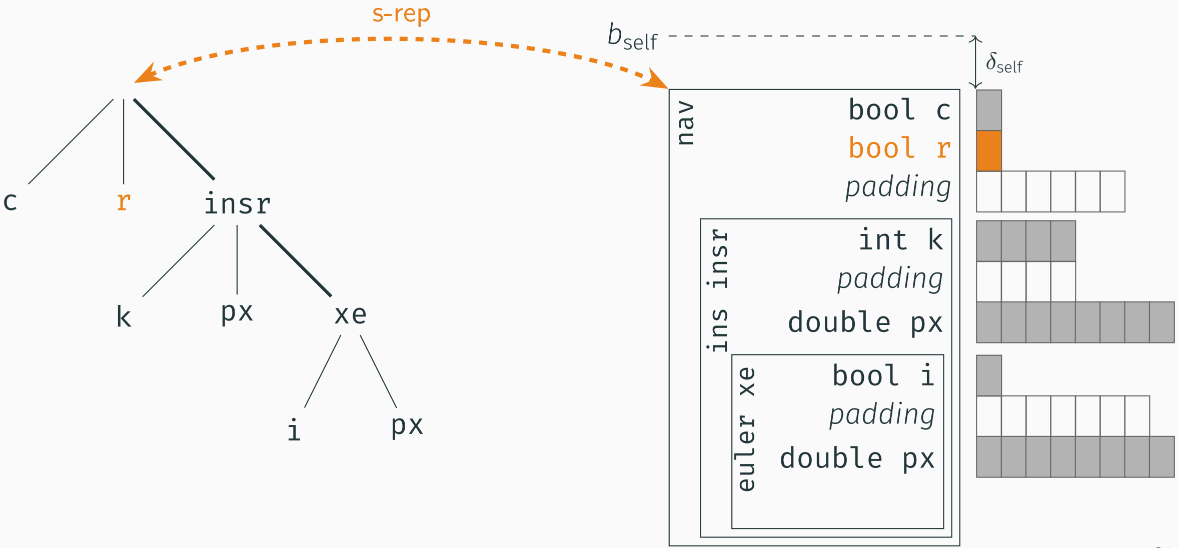
State Correspondence Predicate



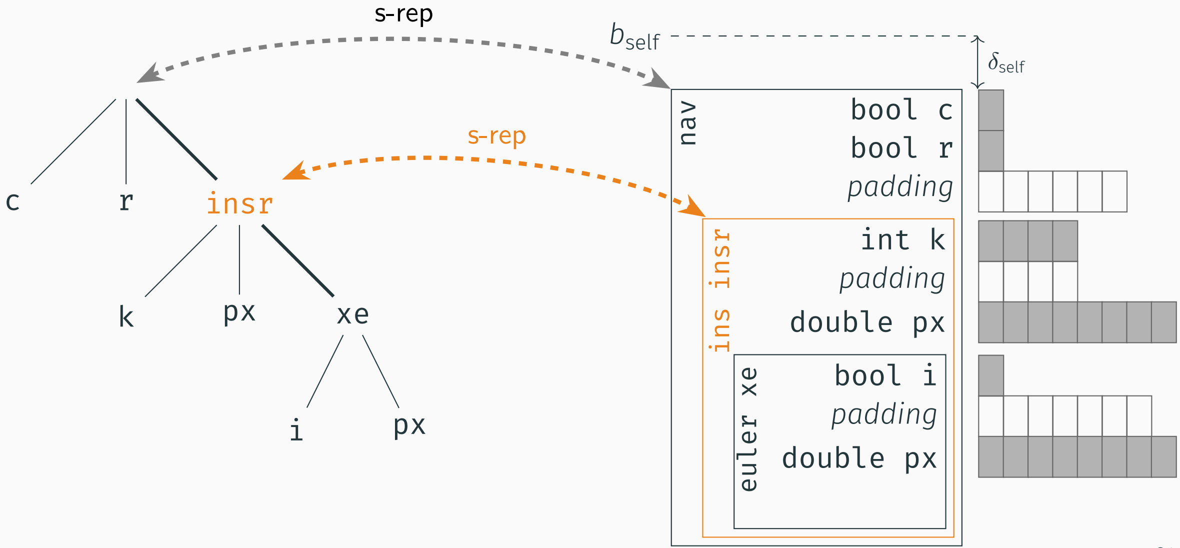
State Correspondence Predicate



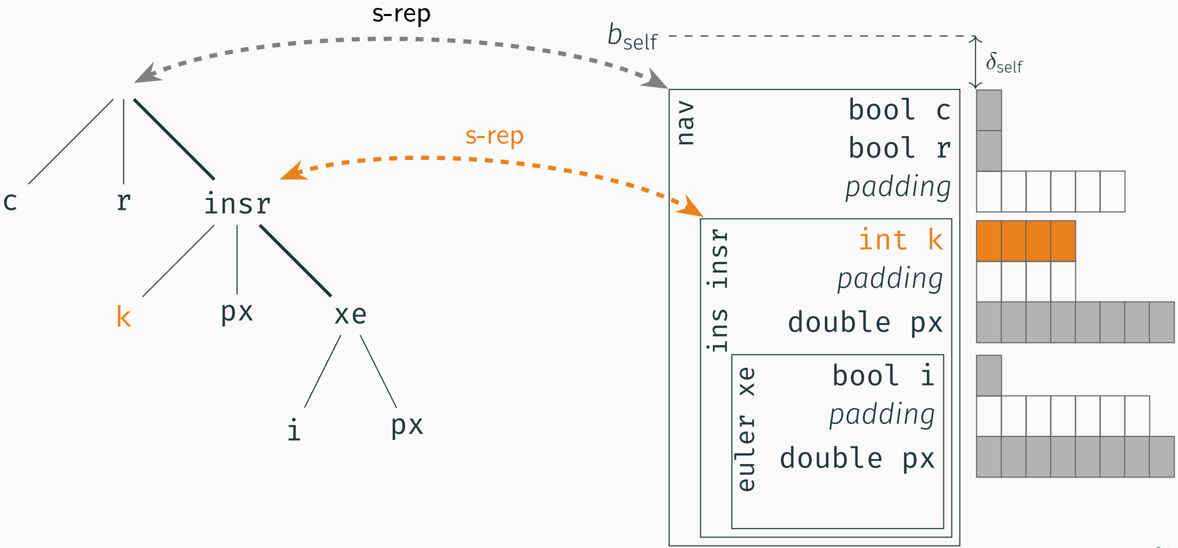
State Correspondence Predicate



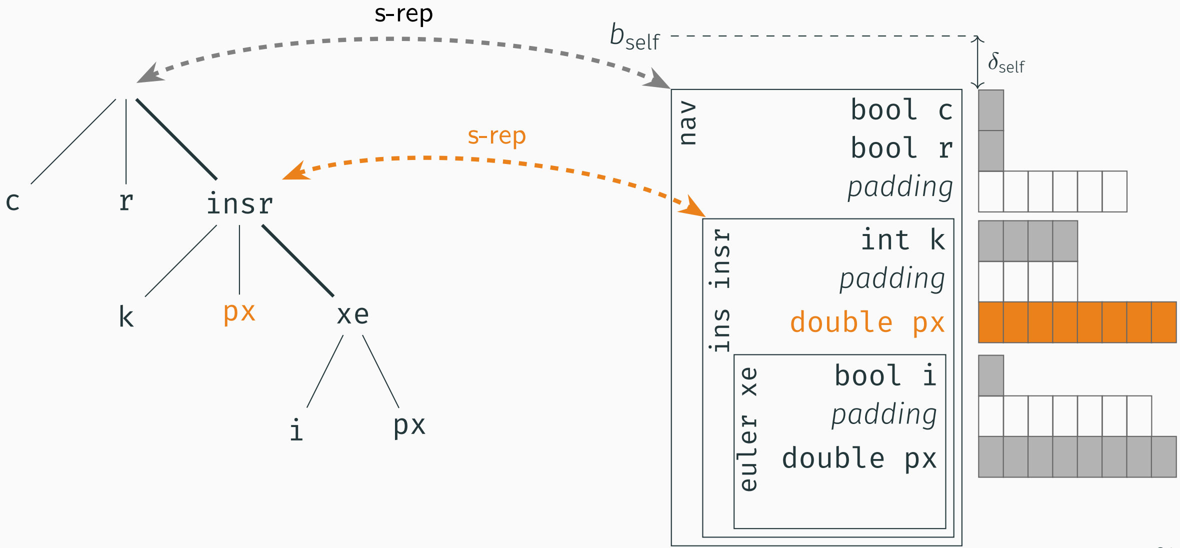
State Correspondence Predicate



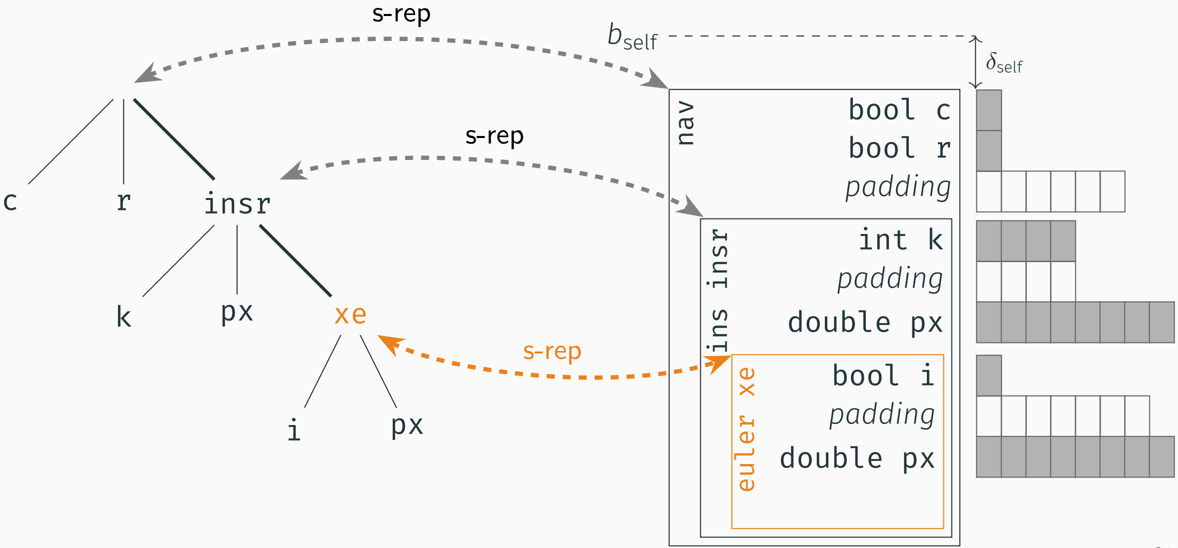
State Correspondence Predicate



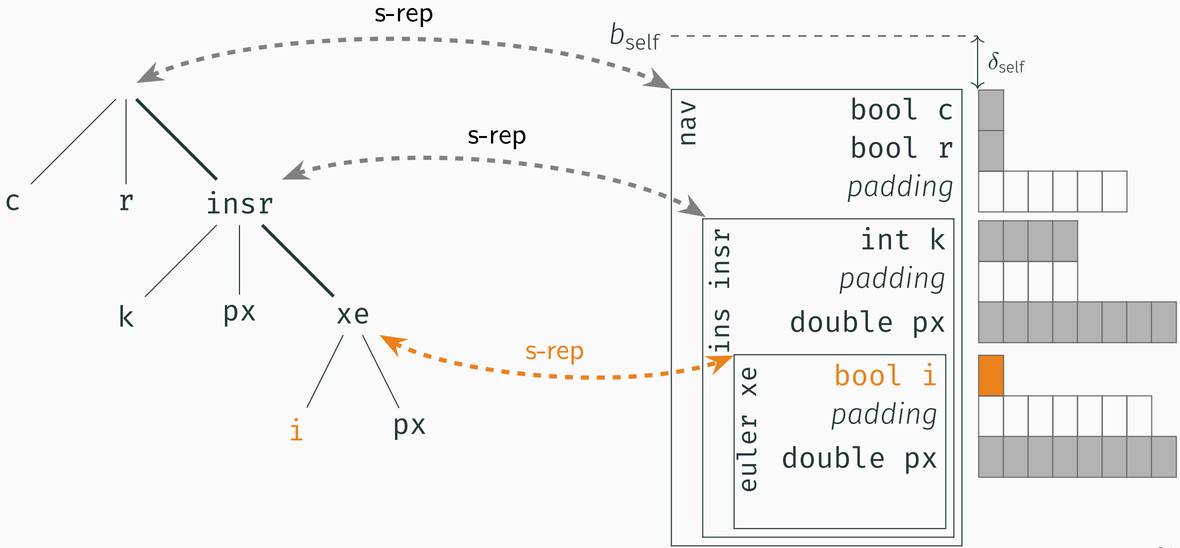
State Correspondence Predicate



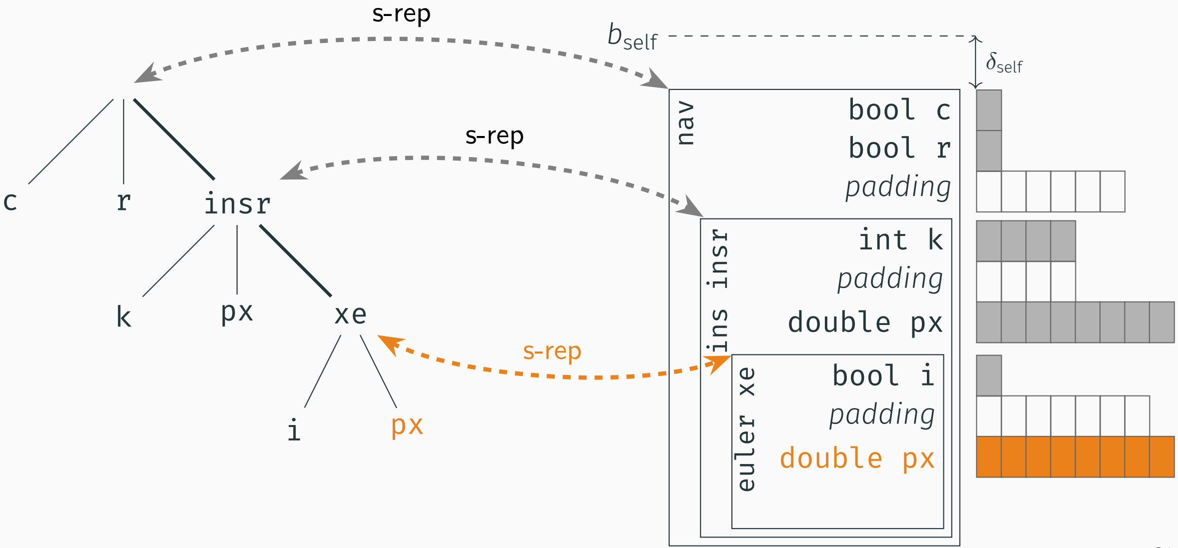
State Correspondence Predicate

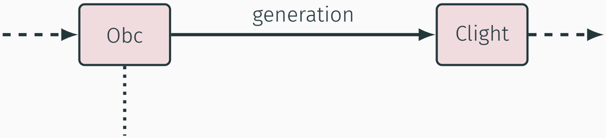


State Correspondence Predicate



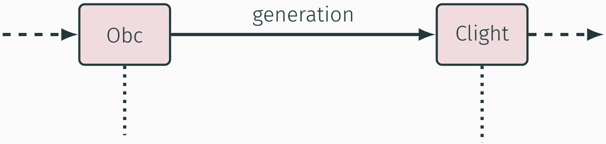
State Correspondence Predicate





$$\varnothing \vdash f.\mathrm{reset}\,() \Downarrow me_0$$

$$me_0 \vdash f.\mathrm{step}\,(xs) \overset{0}{\circlearrowleft} ys$$



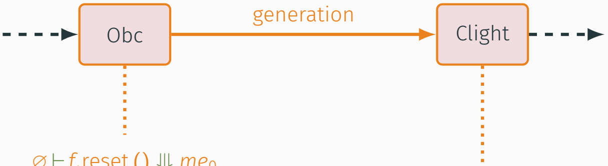
$\varnothing \vdash f.\mathrm{reset}\,() \Downarrow me_0$

$me_0 \vdash f.\mathrm{step}\,(xs) \overset{0}{\circlearrowleft} ys$

$\exists T,\ P \Downarrow \mathbf{Reacts}(T)$

T and (xs, ys) are the “same”

Correctness



$$\frac{\varnothing \vdash f.\mathsf{reset}\,() \Downarrow me_0 \qquad me_0 \vdash f.\mathsf{step}\,(xs) \overset{0}{\circlearrowleft} ys}{} \Longrightarrow P \Downarrow \mathbf{Reacts}(\mathbf{trace\text{-}step}\; f\; xs\; ys)$$

Conclusion

Ultimate Theorem

Theorem (Vélus correctness)

Given a list of declarations D, a name f, lists of streams of values $\mathbf{xs}$ *and* $\mathbf{ys}$*, an NLustre program G and an assembly program P such that* $\mathsf{compile}\, D\, f = \mathrm{OK}\,(G, P)$ *and* $G \vdash f\,(\mathbf{xs}) \Downarrow \mathbf{ys}$*, then there exists an infinite trace of events T such that*

$$P \Downarrow_{ASM} \mathrm{Reacts}(T) \quad \textit{and} \quad \textsf{bisim-IO}^{G}\, f\, \mathbf{xs}\, \mathbf{ys}\, T$$

Theorem (Vélus correctness)

Given a list of declarations D, a name f, lists of streams of values $\mathbf{xs}$ *and* $\mathbf{ys}$*, an NLustre program G and an assembly program P such that* $\mathsf{compile}\,D\,f = \mathsf{OK}\,(G, P)$ *and* $G \vdash f\,(\mathbf{xs}) \Downarrow \mathbf{ys}$*, then there exists an infinite trace of events T such that*

$$P \Downarrow_{ASM} \mathsf{Reacts}(T) \quad \text{and} \quad \mathsf{bisim\text{-}IO}^{G}\,f\,\mathbf{xs}\,\mathbf{ys}\,T$$

Theorem (Vélus correctness)

Given a list of declarations D, a name f, lists of streams of values $\mathbf{xs}$ *and* $\mathbf{ys}$, *an NLustre program G and an assembly program P such that* $\mathsf{compile}\, D\, f = \mathsf{OK}\,(G, P)$ *and* $G \vdash f\,(\mathbf{xs}) \Downarrow \mathbf{ys}$, *then there exists an infinite trace of events T such that*

$$P \Downarrow_{ASM} \mathsf{Reacts}(T) \quad \textit{and} \quad \mathsf{bisim\text{-}IO}^{G}\, f\, \mathbf{xs}\, \mathbf{ys}\, T$$

Theorem (Vélus correctness)

Given a list of declarations D, a name f, lists of streams of values $\mathbf{xs}$ *and* $\mathbf{ys}$*, an NLustre program G and an assembly program P such that* $\mathsf{compile}\, D\, f = \mathsf{OK}\,(G, P)$ *and* $G \vdash f\,(\mathbf{xs}) \Downarrow \mathbf{ys}$*, then there exists an infinite trace of events T such that*

$$P \Downarrow_{ASM} \mathsf{Reacts}(T) \quad \textit{and} \quad \mathsf{bisim\text{-}IO}^G\, f\, \mathbf{xs}\, \mathbf{ys}\, T$$

Theorem (Vélus correctness)

Given a list of declarations D, a name f, lists of streams of values $\mathbf{xs}$ *and* $\mathbf{ys}$, *an NLustre program G and an assembly program P such that* $\mathsf{compile}\, D\, f = \mathrm{OK}\,(G, P)$ *and* $G \vdash f\,(\mathbf{xs}) \Downarrow \mathbf{ys}$, *then there exists an infinite trace of events T such that*

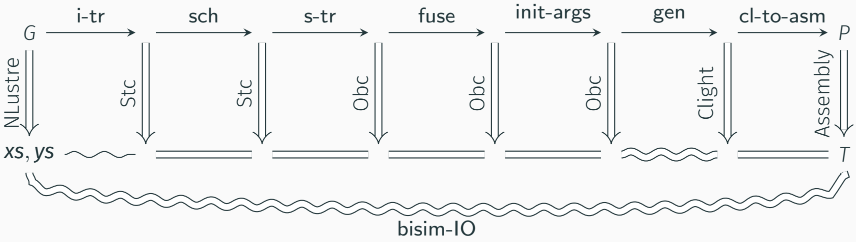
$$P \Downarrow_{ASM} \mathrm{Reacts}(T) \quad \textit{and} \quad \mathsf{bisim\text{-}IO}^{G}\, f\, \mathbf{xs}\, \mathbf{ys}\, T$$

Ultimate Theorem

Theorem (Vélus correctness)

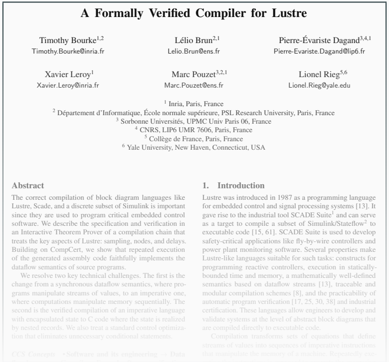
Given a list of declarations D, a name f, lists of streams of values $\mathbf{xs}$ *and* $\mathbf{ys}$*, an NLustre program G and an assembly program P such that* $\mathsf{compile}\ D\ f = \mathsf{OK}\ (G, P)$ *and* $G \vdash f\ (\mathbf{xs}) \Downarrow \mathbf{ys}$*, then there exists an infinite trace of events T such that*

$$P \Downarrow_{ASM} \mathsf{Reacts}(T) \quad \textit{and} \quad \mathsf{bisim\text{-}IO}^{G}\ f\ \mathbf{xs}\ \mathbf{ys}\ T$$



Contributions

PLDI'17



A Formally Verified Compiler for Lustre

Timothy Bourke[1,2] Timothy.Bourke@inria.fr
Lélio Brun[2,1] Lelio.Brun@ens.fr
Pierre-Évariste Dagand[3,4,1] Pierre-Evariste.Dagand@lip6.fr
Xavier Leroy[1] Xavier.Leroy@inria.fr
Marc Pouzet[3,2,1] Marc.Pouzet@ens.fr
Lionel Rieg[5,6] Lionel.Rieg@yale.edu

[1] Inria, Paris, France
[2] Département d'Informatique, École normale supérieure, PSL Research University, Paris, France
[3] Sorbonne Universités, UPMC Univ Paris 06, France
[4] CNRS, LIP6 UMR 7606, Paris, France
[5] Collège de France, Paris, France
[6] Yale University, New Haven, Connecticut, USA

Abstract

The correct compilation of block diagram languages like Lustre, Scade, and a discrete subset of Simulink is important since they are used to program critical embedded control software. We describe the specification and verification in an Interactive Theorem Prover of a compilation chain that treats the key aspects of Lustre: sampling, nodes, and delays. Building on CompCert, we show that repeated execution of the generated assembly code faithfully implements the dataflow semantics of source programs.

We resolve two key technical challenges. The first is the change from a synchronous dataflow semantics, where programs manipulate streams of values, to an imperative one, where computations manipulate memory sequentially. The second is the verified compilation of an imperative language with encapsulated state to C code where the state is realized by nested records. We also treat a standard control optimization that eliminates unnecessary conditional statements.

1. Introduction

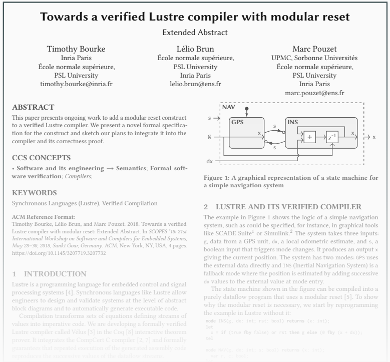
Lustre was introduced in 1987 as a programming language for embedded control and signal processing systems [13]. It gave rise to the industrial tool SCADE Suite[1] and can serve as a target to compile a subset of Simulink/Stateflow[2] to executable code [15, 61]. SCADE Suite is used to develop safety-critical applications like fly-by-wire controllers and power plant monitoring software. Several properties make Lustre-like languages suitable for such tasks: constructs for programming reactive controllers, execution in statically-bounded time and memory, a mathematically well-defined semantics based on dataflow streams [13], traceable and modular compilation schemes [8], and the practicality of automatic program verification [17, 25, 30, 38] and industrial certification. These languages allow engineers to develop and validate systems at the level of abstract block diagrams that are compiled directly to executable code.

Compilation transforms sets of equations that define streams of values into sequences of imperative instructions that manipulate the memory of a machine. Repeatedly exe-

1st version of Vélus

Obc to Clight pass

SCOPES'18



Towards a verified Lustre compiler with modular reset

Extended Abstract

Timothy Bourke, Inria Paris, École normale supérieure, PSL University, timothy.bourke@inria.fr
Lélio Brun, École normale supérieure, PSL University, Inria Paris, lelio.brun@ens.fr
Marc Pouzet, UPMC, Sorbonne Universités, École normale supérieure, PSL University, Inria Paris, marc.pouzet@ens.fr

ABSTRACT

This paper presents ongoing work to add a modular reset construct to a verified Lustre compiler. We present a novel formal specification for the construct and sketch our plans to integrate it into the compiler and its correctness proof.

CCS CONCEPTS

• Software and its engineering → Semantics; Formal software verification; Compilers;

KEYWORDS

Synchronous Languages (Lustre), Verified Compilation

ACM Reference Format:
Timothy Bourke, Lélio Brun, and Marc Pouzet. 2018. Towards a verified Lustre compiler with modular reset: Extended Abstract. In *SCOPES '18: 21st International Workshop on Software and Compilers for Embedded Systems, May 28–30, 2018, Sankt Goar, Germany.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3207719.3207732

1 INTRODUCTION

Lustre is a programming language for embedded control and signal processing systems [4]. Synchronous languages like Lustre allow engineers to design and validate systems at the level of abstract block diagrams and to automatically generate executable code.

Compilation transforms sets of equations defining streams of values into imperative code. We are developing a formally verified Lustre compiler called Vélus [3] in the Coq [8] interactive theorem prover. It integrates the CompCert C compiler [2, 7] and formally guarantees that repeated execution of the generated assembly code reproduces the successive values of the dataflow streams.

Figure 1: A graphical representation of a state machine for a simple navigation system

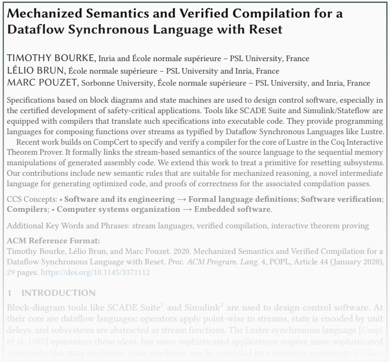
2 LUSTRE AND ITS VERIFIED COMPILER

The example in Figure 1 shows the logic of a simple navigation system, such as could be specified, for instance, in graphical tools like SCADE Suite[1] or Simulink.[2] The system takes three inputs: g, data from a GPS unit, dx, a local odometric estimate, and s, a boolean input that triggers mode changes. It produces an output x giving the current position. The system has two modes: GPS uses the external data directly and INS (Inertial Navigation System) is a fallback mode where the position is estimated by adding successive dx values to the external value at mode entry.

The state machine shown in the figure can be compiled into a purely dataflow program that uses a modular reset [5]. To show why the modular reset is necessary, we start by reprogramming the example in Lustre without it:

Reset formal semantics

POPL'20



Mechanized Semantics and Verified Compilation for a Dataflow Synchronous Language with Reset

TIMOTHY BOURKE, Inria and École normale supérieure – PSL University, France
LÉLIO BRUN, École normale supérieure – PSL University and Inria, France
MARC POUZET, Sorbonne University, École normale supérieure – PSL University, and Inria, France

Specifications based on block diagrams and state machines are used to design control software, especially in the certified development of safety-critical applications. Tools like SCADE Suite and Simulink/Stateflow are equipped with compilers that translate such specifications into executable code. They provide programming languages for composing functions over streams as typified by Dataflow Synchronous Languages like Lustre.

Recent work builds on CompCert to specify and verify a compiler for the core of Lustre in the Coq Interactive Theorem Prover. It formally links the stream-based semantics of the source language to the sequential memory manipulations of generated assembly code. We extend this work to treat a primitive for resetting subsystems. Our contributions include new semantic rules that are suitable for mechanized reasoning, a novel intermediate language for generating optimized code, and proofs of correctness for the associated compilation passes.

CCS Concepts: • **Software and its engineering → Formal language definitions**; **Software verification**; **Compilers**; • **Computer systems organization → Embedded software**.

Additional Key Words and Phrases: stream languages, verified compilation, interactive theorem proving

ACM Reference Format:
Timothy Bourke, Lélio Brun, and Marc Pouzet. 2020. Mechanized Semantics and Verified Compilation for a Dataflow Synchronous Language with Reset. *Proc. ACM Program. Lang.* 4, POPL, Article 44 (January 2020), 29 pages. https://doi.org/10.1145/3371112

1 INTRODUCTION

Block-diagram tools like SCADE Suite[1] and Simulink[2] are used to design control software. At their core are dataflow languages: operators apply point-wise to streams, state is encoded by unit delays, and subsystems are abstracted as stream functions. The Lustre synchronous language [Caspi et al. 1987] epitomizes these ideas, but more sophisticated applications require more sophisticated constructs like state machines. State machines can be compiled into primitive constructs [Colaço

2nd version of Vélus

Stc and reset compilation

Summary

- A verified compiler from Lustre to Assembly
- A single additional semantic rule for the reset
- An intermediate transition system language: Stc

VÉLUS

`velus.inria.fr`

`github.com/INRIA/velus`

Future Work

- Normalization
- State machines
- *Refinement*
- Optimizations

Perspectives and discussion

- 42 000 loc and only 3% of functional code
- Extensibility
- Maintenance
- Axioms

References I

- Paul Caspi, Daniel Pilaud, Nicolas Halbwachs, and John Alexander Plaice (1987). "LUSTRE: A Declarative Language for Programming Synchronous Systems". In: *In 14th Symposium on Principles of Programming Languages (POPL'87). ACM.*
- Nicolas Halbwachs, Paul Caspi, Pascal Raymond, and Daniel Pilaud (Sept. 1991). "The Synchronous Data Flow Programming Language LUSTRE". In: *Proceedings of the IEEE 79.9*, pp. 1305–1320.
- Paul Caspi (Jan. 1, 1994). "Towards Recursive Block Diagrams". In: *Annual Review in Automatic Programming* 18, pp. 81–85.
- Grégoire Hamon and Marc Pouzet (2000). "Modular Resetting of Synchronous Data-Flow Programs". In: *Proceedings of the 2Nd ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming*. PPDP '00. New York, NY, USA: ACM, pp. 289–300.
- John C. Reynolds (2002). "Separation Logic: A Logic for Shared Mutable Data Structures". In: *Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science*. LICS '02. Washington, DC, USA: IEEE Computer Society, pp. 55–74.

References II

- Jean-Louis Colaço, Bruno Pagano, and Marc Pouzet (2005). "A Conservative Extension of Synchronous Data-Flow with State Machines". In: *Proceedings of the 5th ACM International Conference on Embedded Software*. EMSOFT '05. New York, NY, USA: ACM, pp. 173–182.
- Dariusz Biernacki, Jean-Louis Colaço, Gregoire Hamon, and Marc Pouzet (2008). "Clock-Directed Modular Code Generation for Synchronous Data-Flow Languages". In: *Proceedings of the 2008 ACM SIGPLAN-SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*. LCTES '08. New York, NY, USA: ACM, pp. 121–130.
- Sandrine Blazy and Xavier Leroy (Oct. 1, 2009). "Mechanized Semantics for the Clight Subset of the C Language". In: *Journal of Automated Reasoning* 43.3, pp. 263–288.
- Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood (Oct. 11, 2009). "seL4: Formal Verification of an OS Kernel". In: *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*. SOSP '09. Big Sky, Montana, USA: Association for Computing Machinery, pp. 207–220.

References III

- Xavier Leroy (July 2009). “Formal Verification of a Realistic Compiler”. In: *Communications of the ACM* 52.7, pp. 107–115.
- Jacques-Henri Jourdan, François Pottier, and Xavier Leroy (2012). “Validating LR(1) Parsers”. In: *Programming Languages and Systems*. Ed. by Helmut Seidl. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 397–416.
- Ramana Kumar, Magnus O. Myreen, Michael Norrish, and Scott Owens (Jan. 2014). “CakeML: A Verified Implementation of ML”. In: *Principles of Programming Languages (POPL)*. ACM Press, pp. 179–191.
- Timothy Bourke, Lélio Brun, Pierre-Évariste Dagand, Xavier Leroy, Marc Pouzet, and Lionel Rieg (June 2017). “A Formally Verified Compiler for Lustre”. In: *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI 2017. New York, NY, USA: ACM, pp. 586–601.
- Jean-Louis Colaço, Bruno Pagano, and Marc Pouzet (Sept. 2017). “SCADE 6: A Formal Language for Embedded Critical Software Development”. In: *2017 International Symposium on Theoretical Aspects of Software Engineering (TASE)*, pp. 1–11.

- Timothy Bourke, Lélio Brun, and Marc Pouzet (Jan. 2020). "Mechanized Semantics and Verified Compilation for a Dataflow Synchronous Language with Reset". In: *Proceedings of the 47th ACM SIGPLAN Symposium on Principles of Programming Languages*. Principles Of Programming Languages. Vol. 4. POPL'20. New Orleans, LA, USA: Association for Computing Machinery, p. 29.