From Lustre to C: a Proof-carrying Approach to Verified Compilation

AID Meeting – *Essaim de drones*

Lélio Brun[1]

October 12, 2021

[1]ISAE-SUPAERO – DISC – IpSC

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

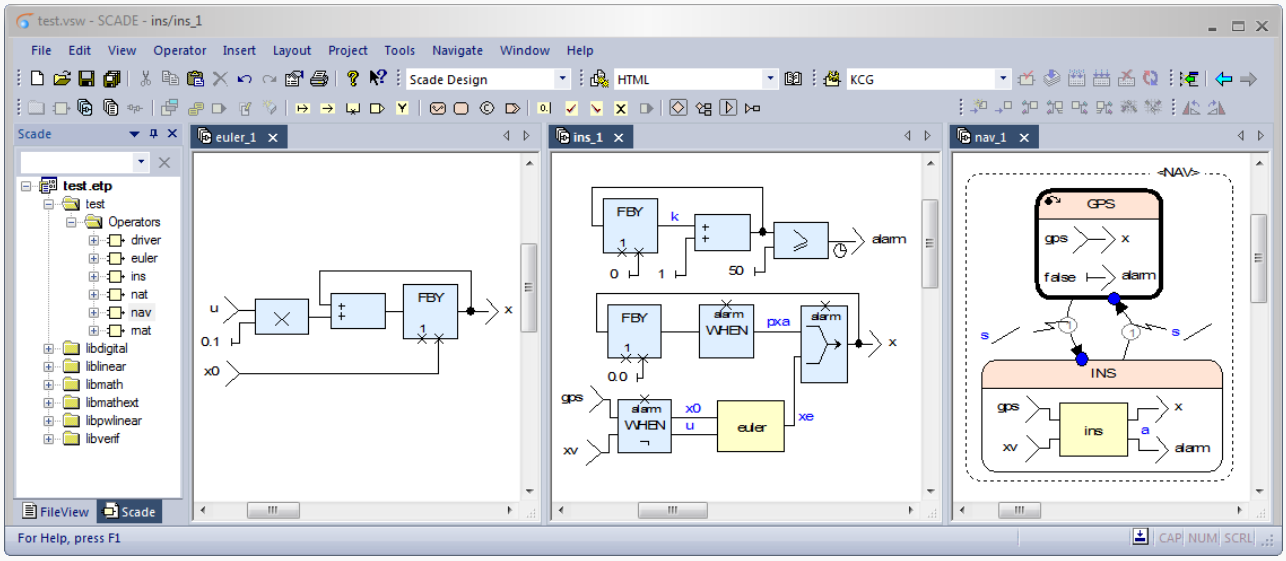
- computer systems within physical systems that interact with the real world, often with real-time constraints
- software usually written in low-level languages: C, Ada, Assembly

Model-Based Design

Executable high-level abstract specifications

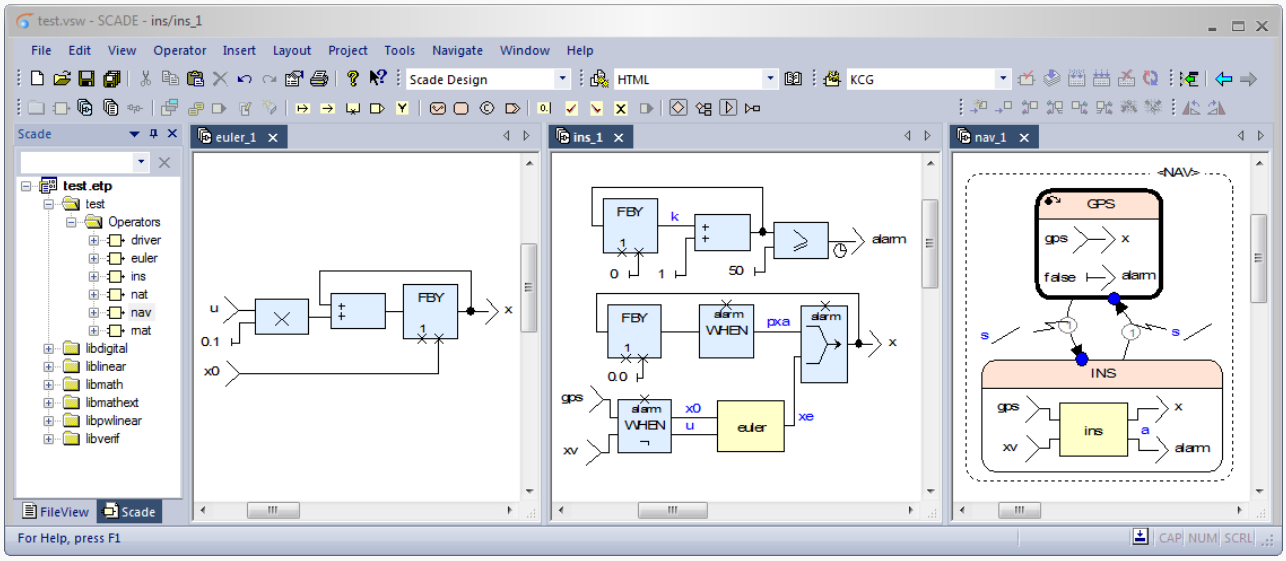
Model-Based Design in SCADE Suite

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



Model-Based Design in SCADE Suite

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

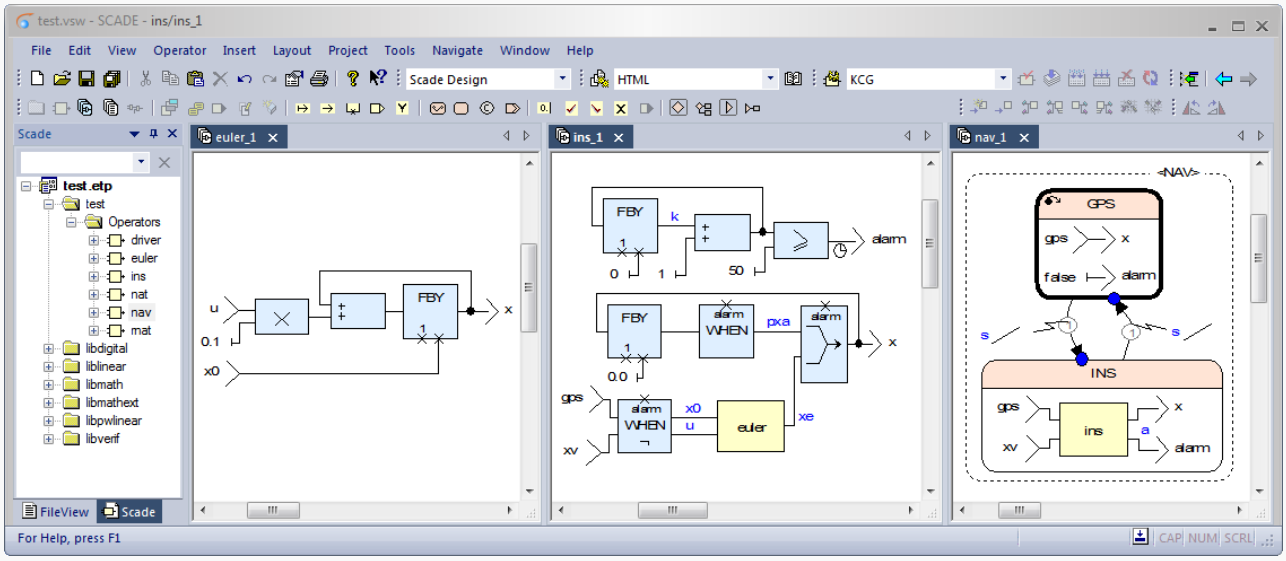


block / node = system

line = signal

Model-Based Design in SCADE Suite

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

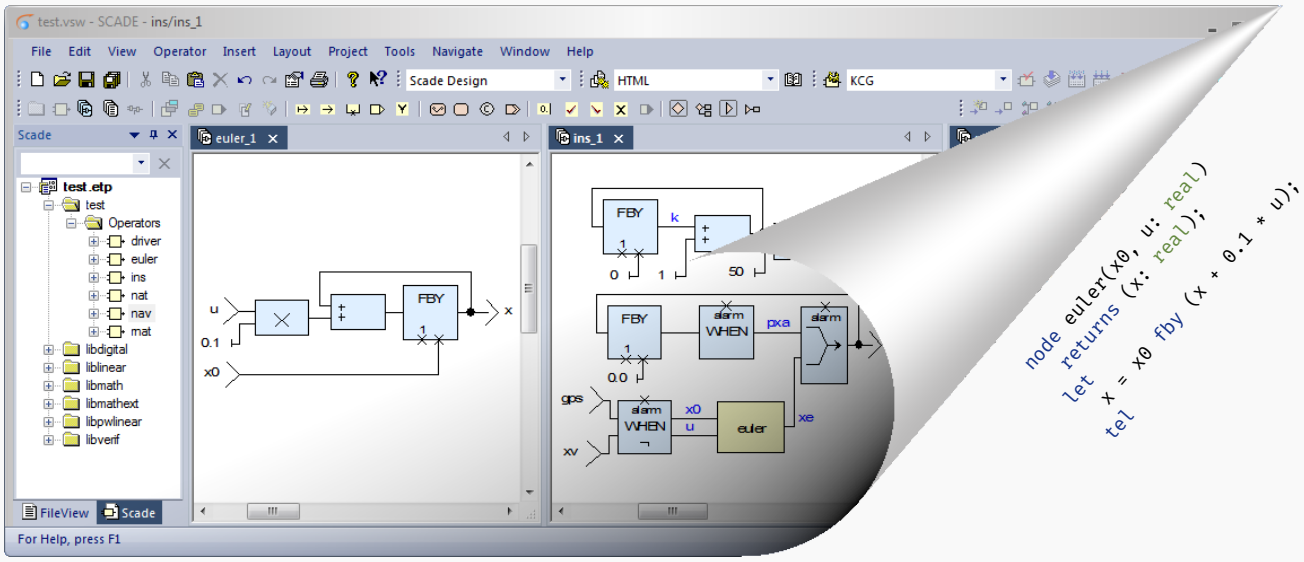


block / node = system = stream function

line = signal = stream of values

Model-Based Design in SCADE Suite

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

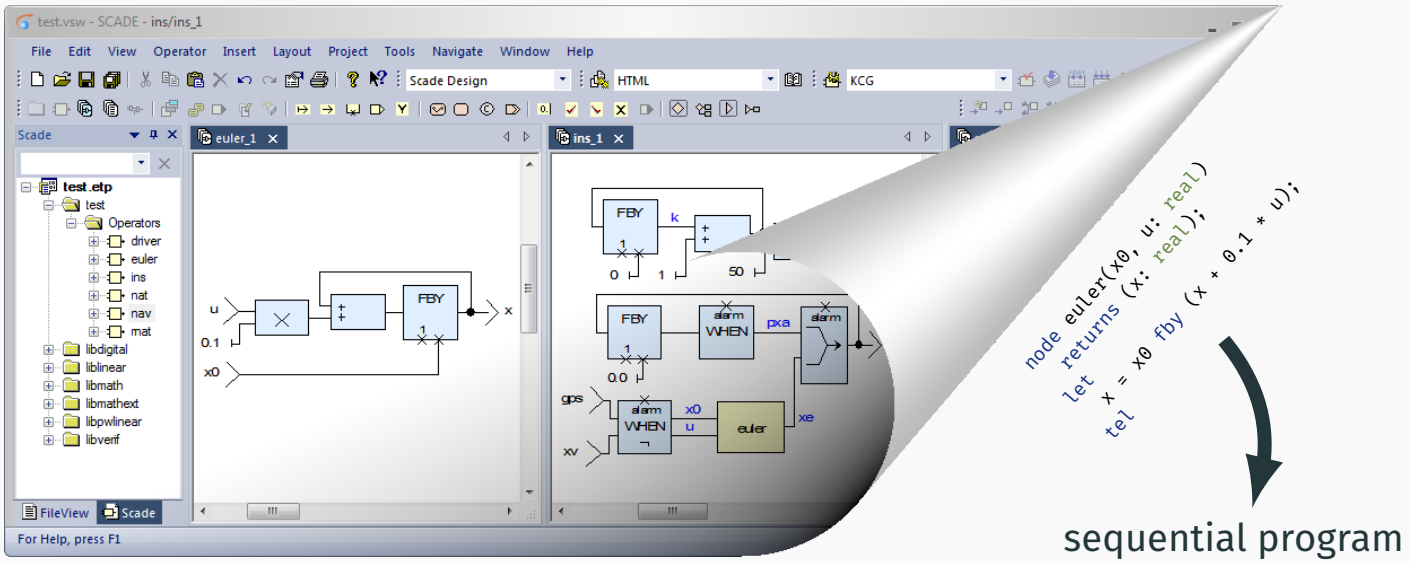


block / node = system = stream function

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

Criticality

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 questions:

- Can we produce an end-to-end correctness proof?
- Can verification performed at high-level be transported and replayed at low-level?

Formal Verification

Leroy 2009
Appel 2011
Klein et al. 2009

Kumar, Myreen, Norrish, and Owens 2014
Bohrer, Tan, Mitsch, Myreen, and Platzer 2018

CompCert a verified C compiler in Coq

VST a C verification toolset based on CompCert

seL4 a verified micro-kernel in Isabelle

CakeML a verified compiler for a functional language in HOL

VeriPhy a verified pipeline from CPS to controllers, using KeYmaera X, HOL4 and Isabelle

Formal Verification

Leroy 2009
Appel 2011
Klein et al. 2009
Kumar, Myreen, Norrish, and Owens 2014
Bohrer, Tan, Mitsch, Myreen, and Platzer 2018
Bourke, Brun, Dagand, Leroy, Pouzet, and Rieg 2017

CompCert a verified C compiler in Coq

VST a C verification toolset based on CompCert

seL4 a verified micro-kernel in Isabelle

CakeML a verified compiler for a functional language in HOL

VeriPhy a verified pipeline from CPS to controllers, using KeYmaera X, HOL4 and Isabelle

Vélus a verified Lustre compiler in Coq, based on CompCert

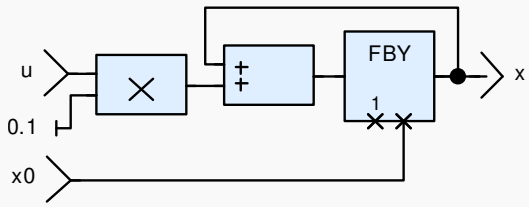
The LustreC Project

An academic Lustre compiler with several backends and verification abilities

- Generation of ACSL specification, along with the C code
- Encoding of the correctness result, to be proven by solvers
- Automatic contract translation

Software Analyzers

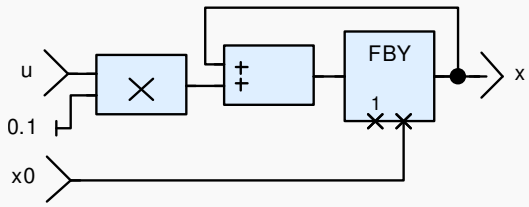
Example




```
node euler(x0, u: real)
  returns (x: real);
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	$\cdots$

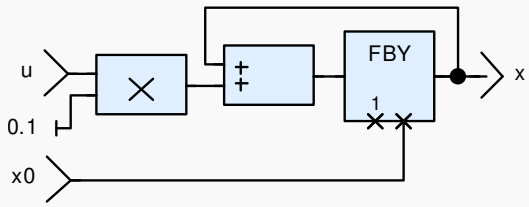
Example




```
node euler(x0, u: real)
  returns (x: real);
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	$\cdots$

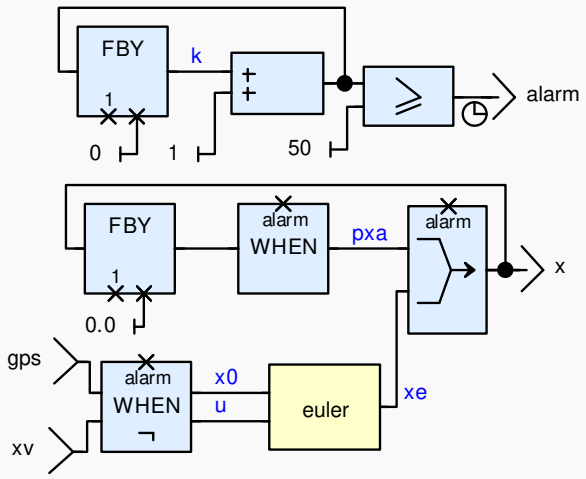
Example




```
node euler(x0, u: real)
  returns (x: real);
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	$\cdots$

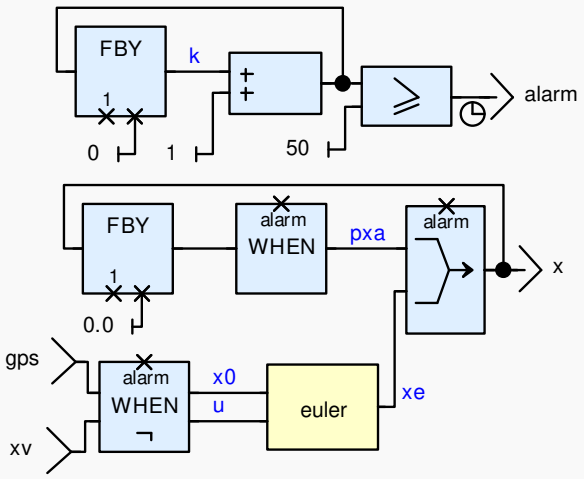
Example




```
node ins(gps, xv: real)
  returns (x: real, alarm: bool)
  var pxa, xe: real; 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	···

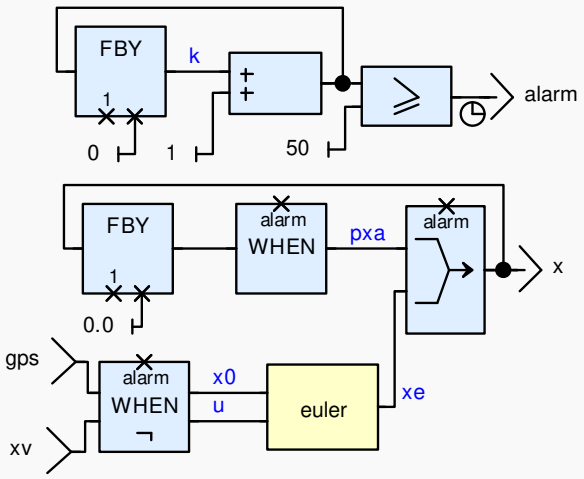
Example




```
node ins(gps, xv: real)
  returns (x: real, alarm: bool)
  var pxa, xe: real; 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	···

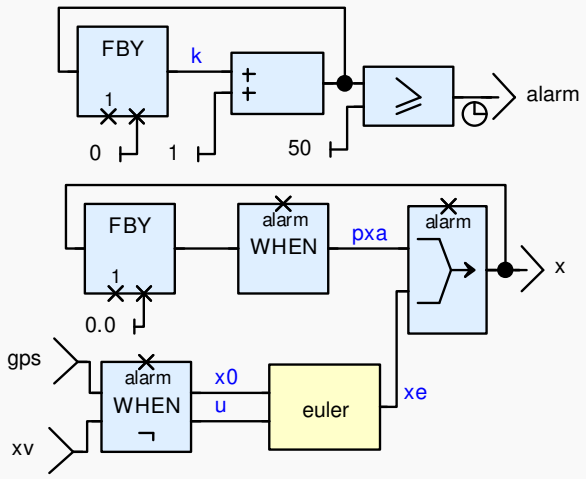
Example




```
node ins(gps, xv: real)
  returns (x: real, alarm: bool)
  var pxa, xe: real; 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	···

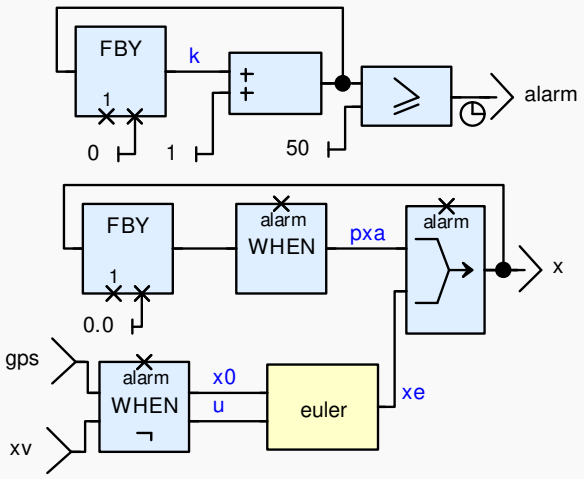
Example




```
node ins(gps, xv: real)
  returns (x: real, alarm: bool)
  var pxa, xe: real; 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

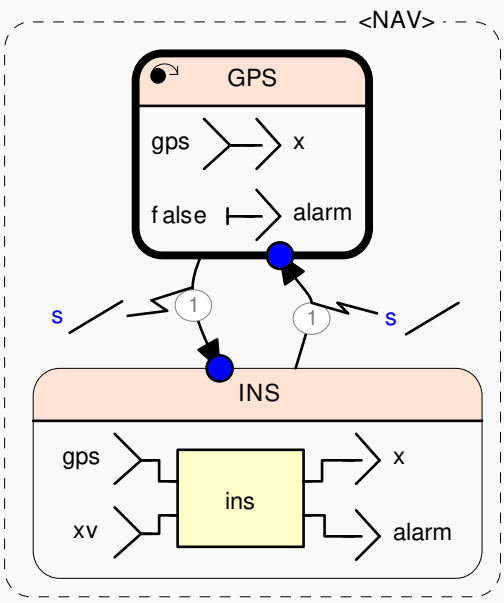



```
node ins(gps, xv: real)
  returns (x: real, alarm: bool)
  var pxa, xe: real; k: int;
let
  x = merge alarm pxa xe;
  k = 0 fby (k + 1);
  pxa = (0. fby x) when alarm;
  xe = euler((gps, xv) when not alarm);
  alarm = (k >= 50);
tel
```

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	···

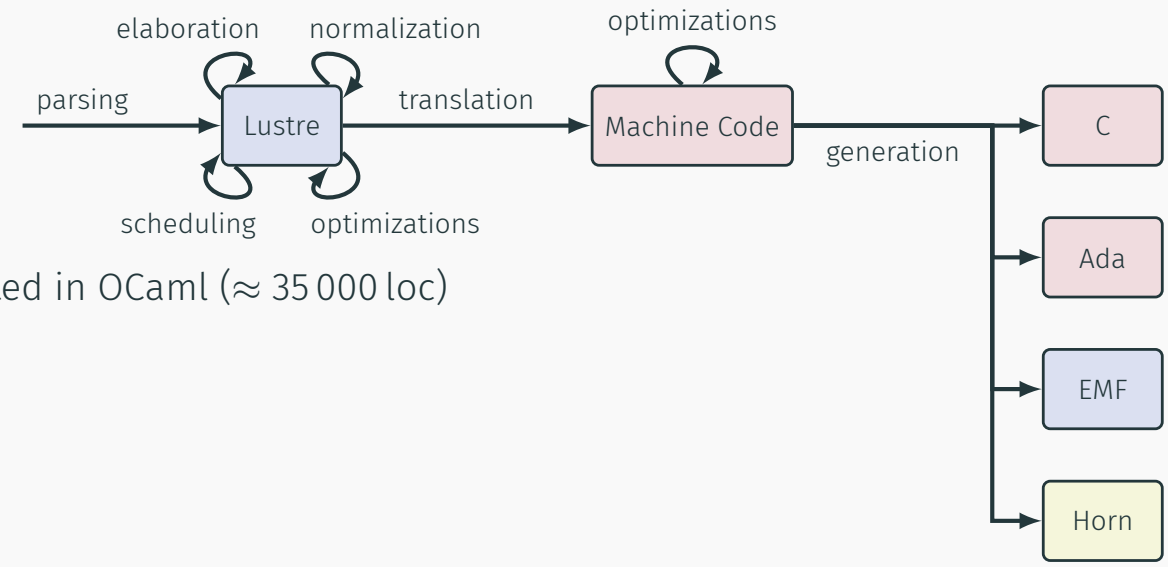
Example

Colaço, Pagano, and Pouzet 2005



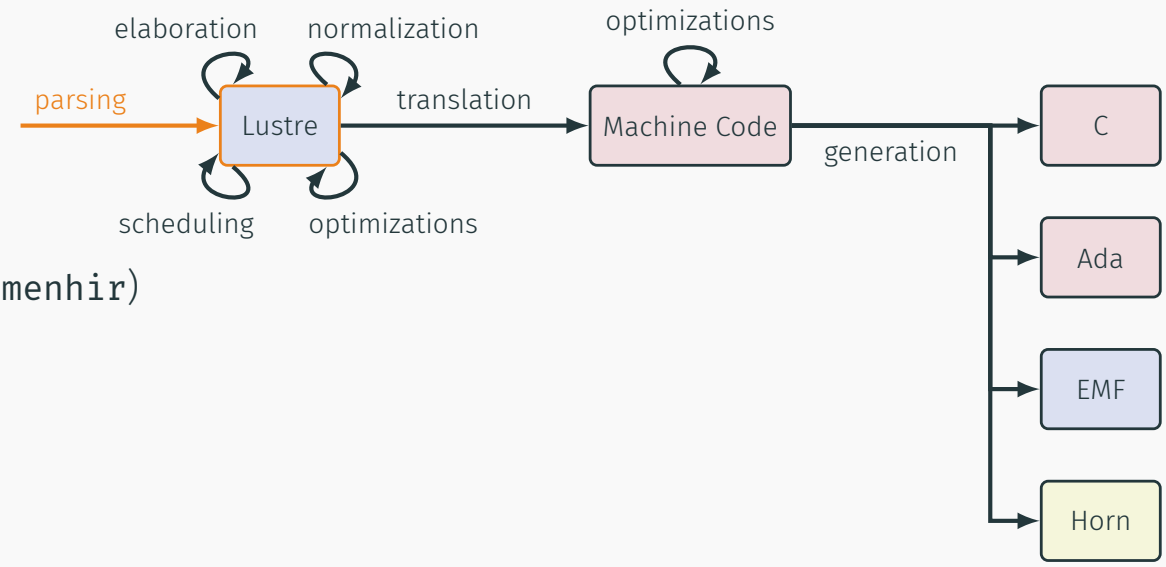

```
node nav(gps, xv: real, s: bool)
  returns (x: real, alarm: bool)
let
  automaton a
    state GPS:
      let
        x = gps;
        alarm = false;
      tel until s restart INS
    state INS:
      let
        (x, alarm) = ins(gps, xv);
      tel until s resume GPS
tel
```

LustreC: a Lustre Compiler



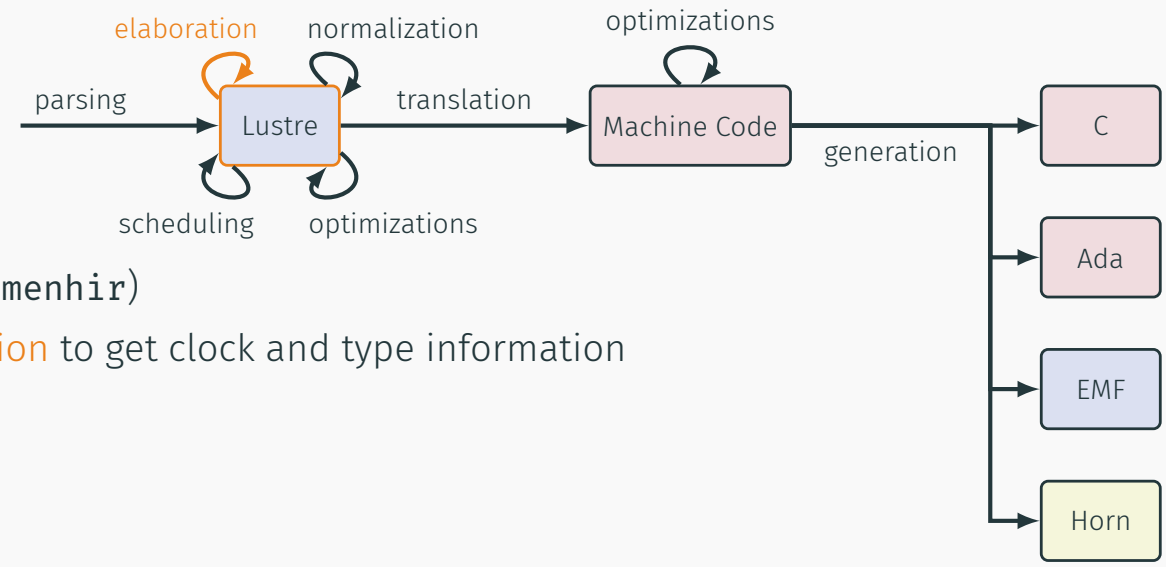
Implemented in OCaml (≈ 35 000 loc)

LustreC: a Lustre Compiler



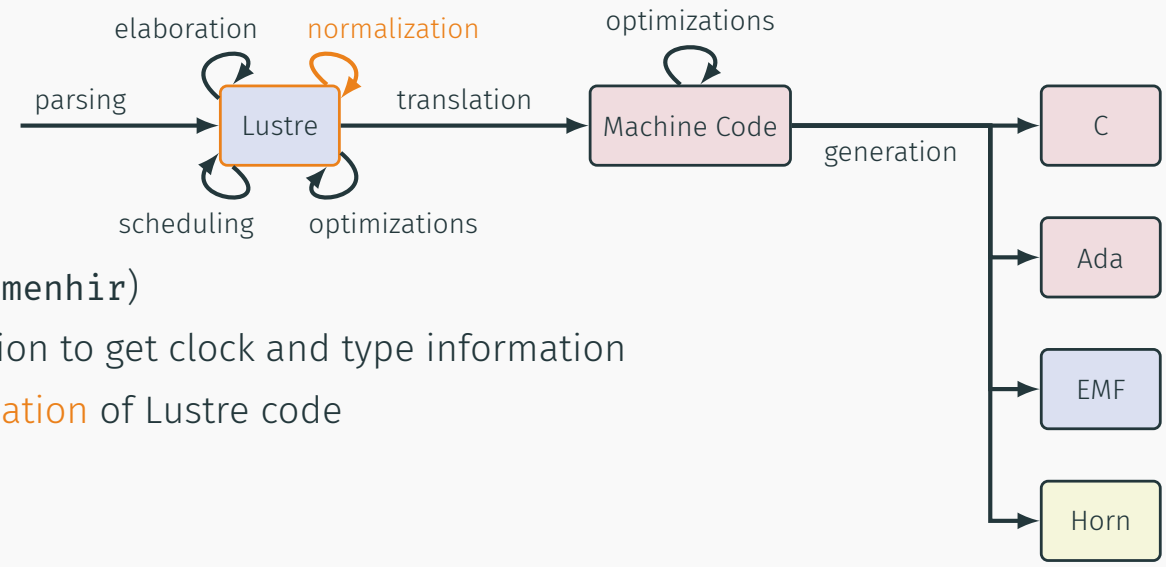
- parsing (`menhir`)

LustreC: a Lustre Compiler



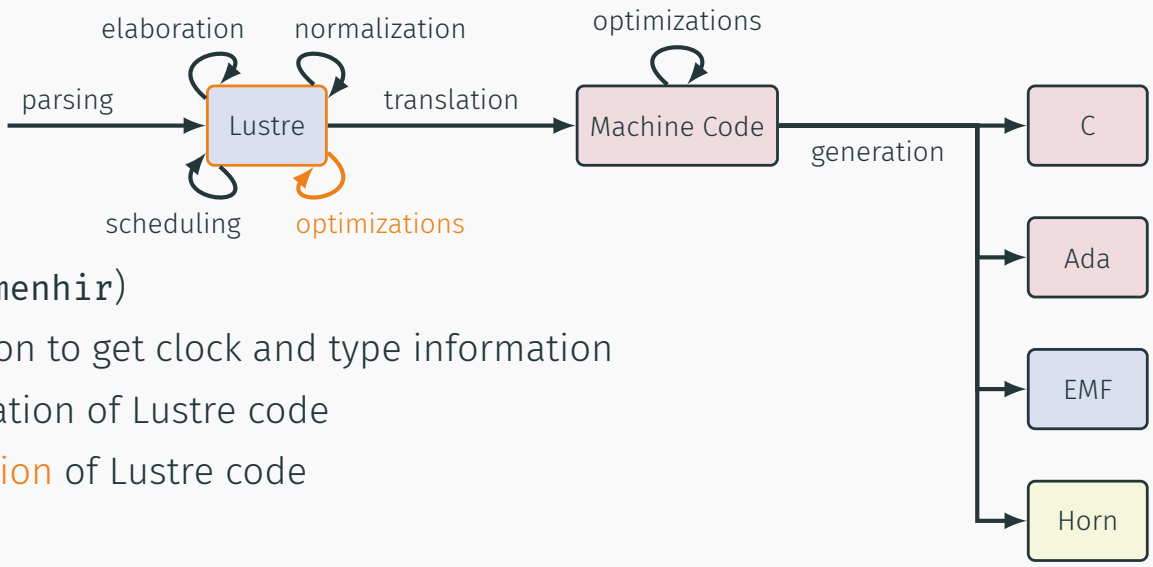
- parsing (`menhir`)
- elaboration to get clock and type information

LustreC: a Lustre Compiler



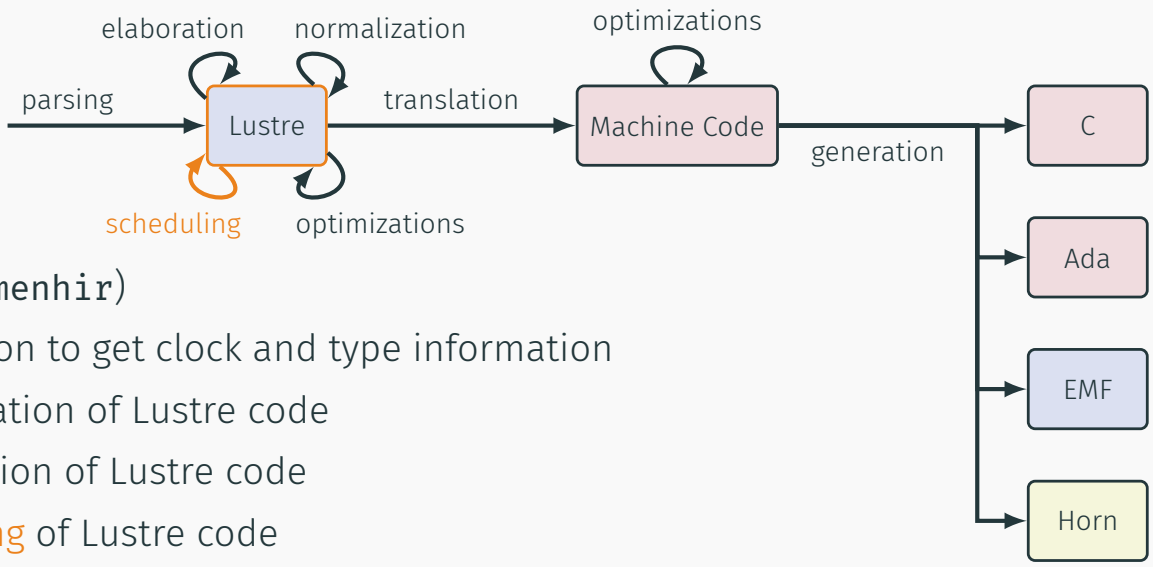
- parsing (`menhir`)
- elaboration to get clock and type information
- normalization of Lustre code

LustreC: a Lustre Compiler



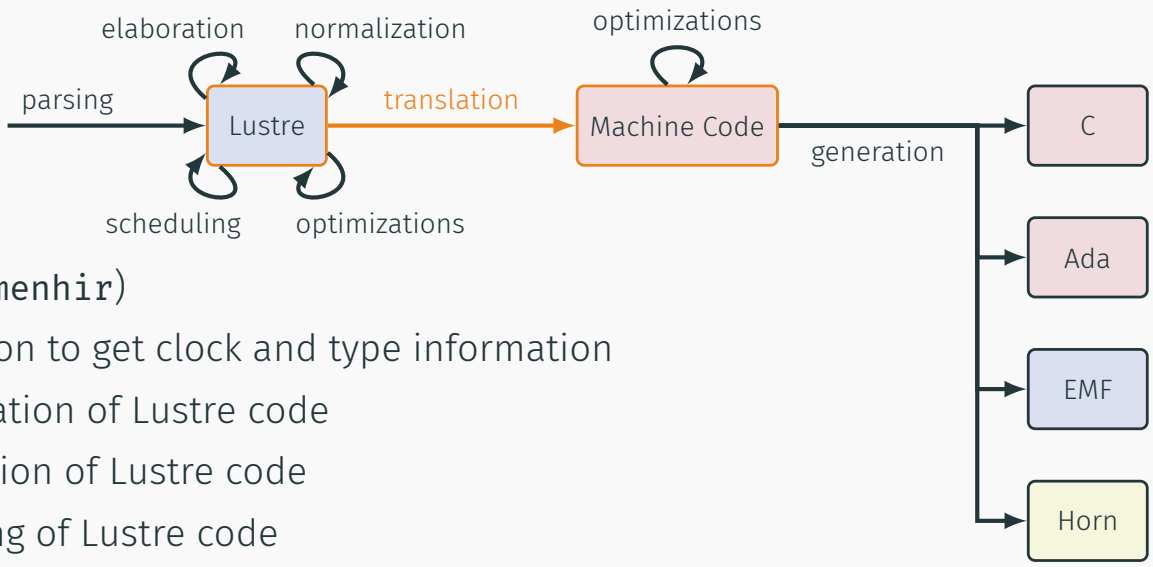
- parsing (`menhir`)
- elaboration to get clock and type information
- normalization of Lustre code
- optimization of Lustre code

LustreC: a Lustre Compiler



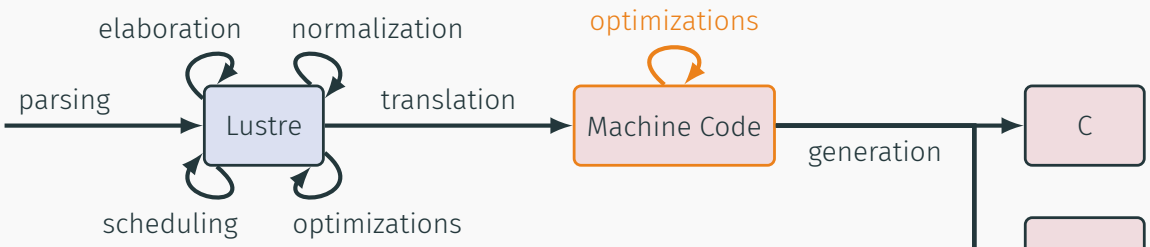
- parsing (`menhir`)
- elaboration to get clock and type information
- normalization of Lustre code
- optimization of Lustre code
- scheduling of Lustre code

LustreC: a Lustre Compiler



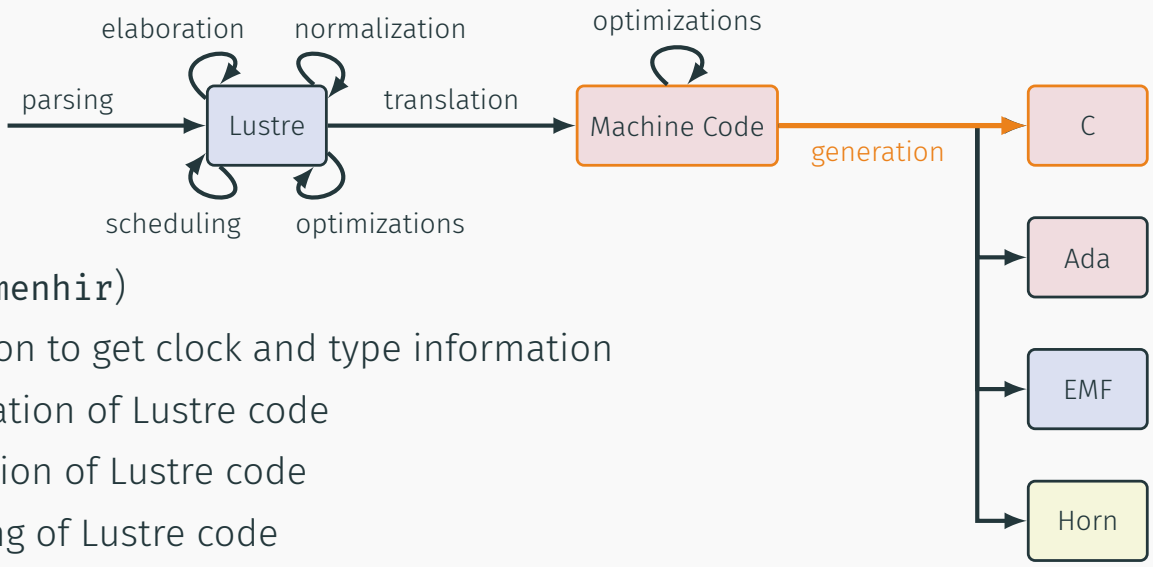
- parsing (`menhir`)
- elaboration to get clock and type information
- normalization of Lustre code
- optimization of Lustre code
- scheduling of Lustre code
- translation to Machine code

LustreC: a Lustre Compiler



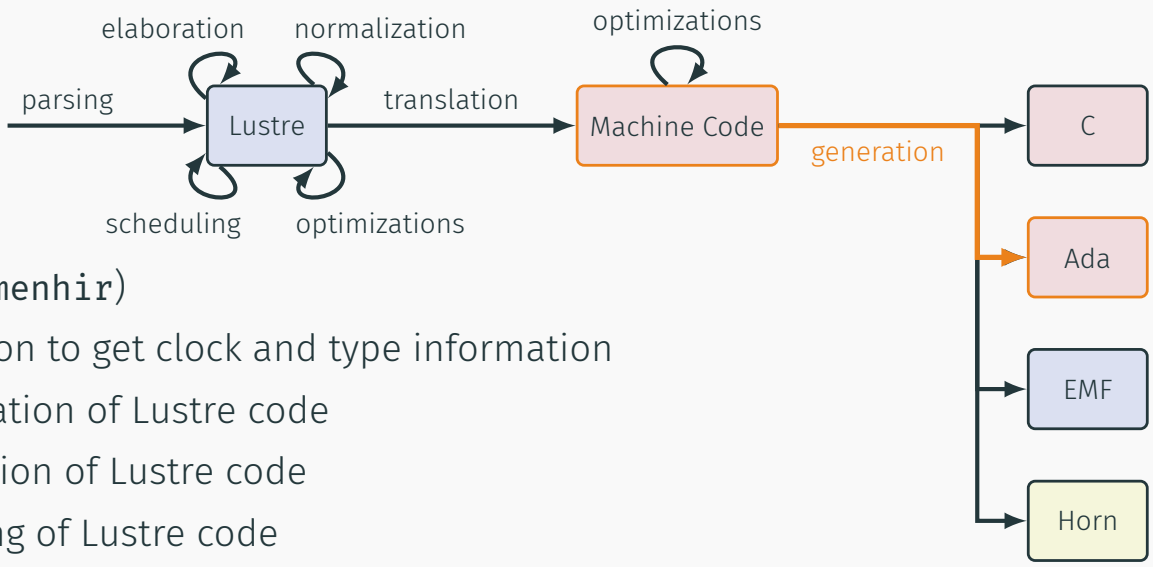
- parsing (`menhir`)
- elaboration to get clock and type information
- normalization of Lustre code
- optimization of Lustre code
- scheduling of Lustre code
- translation to Machine code
- optimisation of Machine code

LustreC: a Lustre Compiler



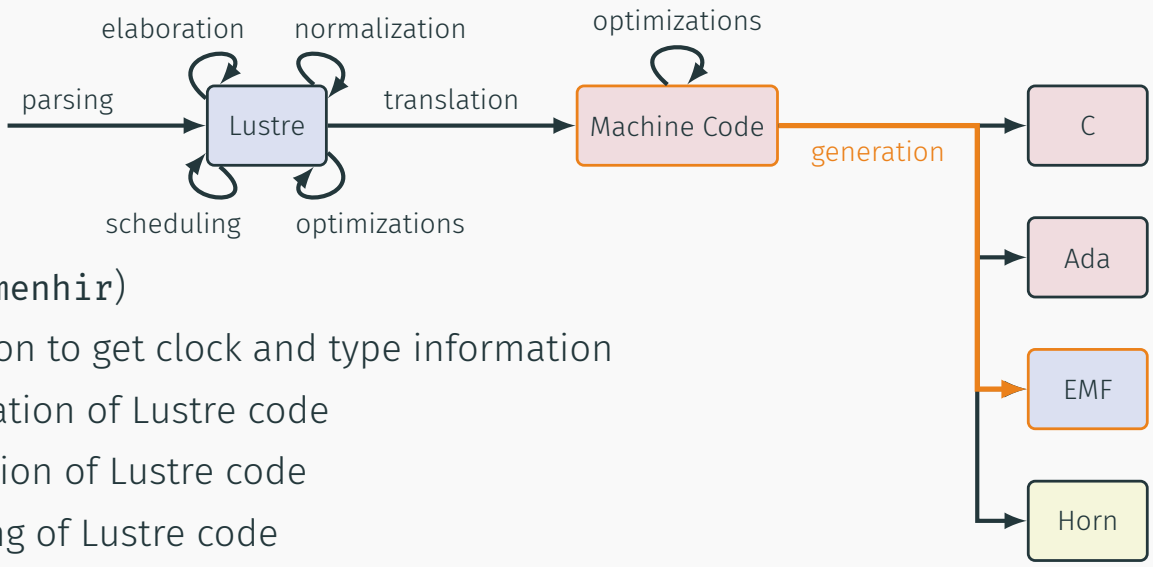
- parsing (`menhir`)
- elaboration to get clock and type information
- normalization of Lustre code
- optimization of Lustre code
- scheduling of Lustre code
- translation to Machine code
- optimisation of Machine code
- generation of C code

LustreC: a Lustre Compiler



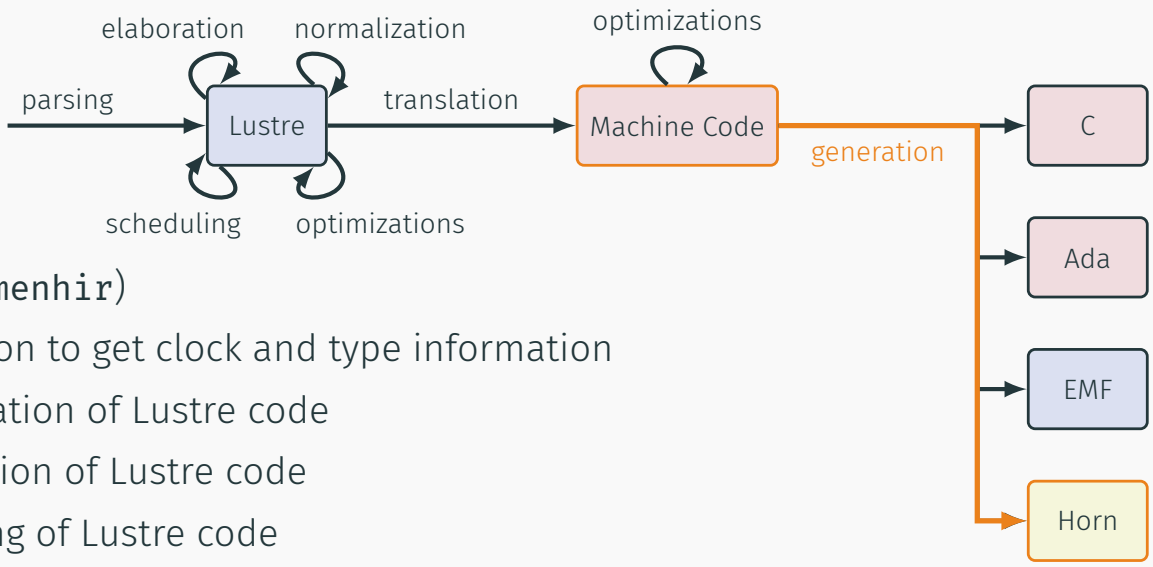
- parsing (`menhir`)
- elaboration to get clock and type information
- normalization of Lustre code
- optimization of Lustre code
- scheduling of Lustre code
- translation to Machine code
- optimisation of Machine code
- generation of Ada code

LustreC: a Lustre Compiler



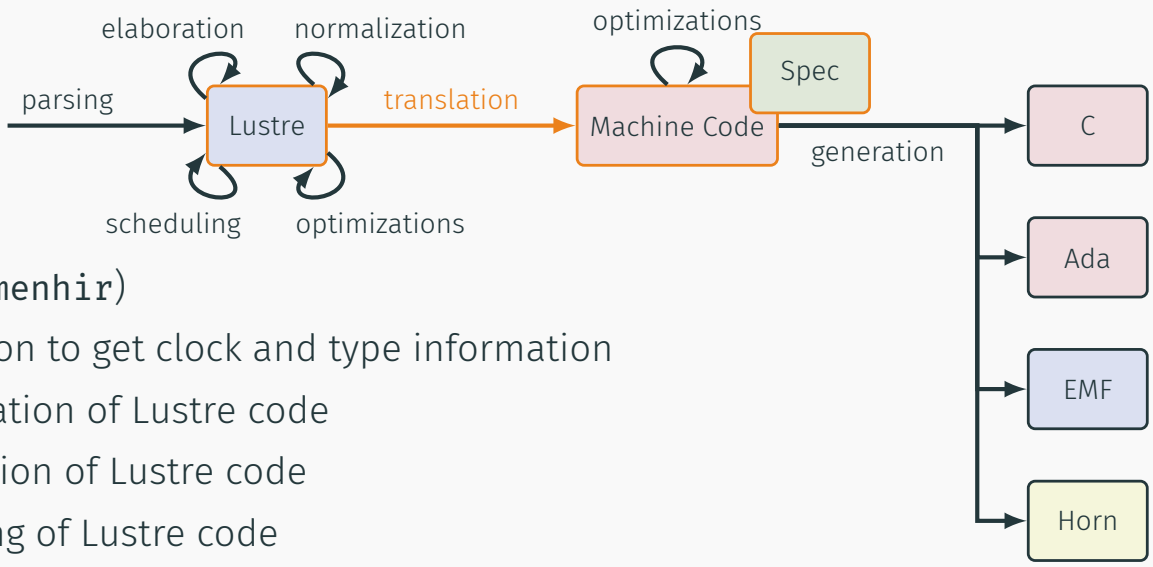
- parsing (`menhir`)
- elaboration to get clock and type information
- normalization of Lustre code
- optimization of Lustre code
- scheduling of Lustre code
- translation to Machine code
- optimisation of Machine code
- generation of EMF code

LustreC: a Lustre Compiler



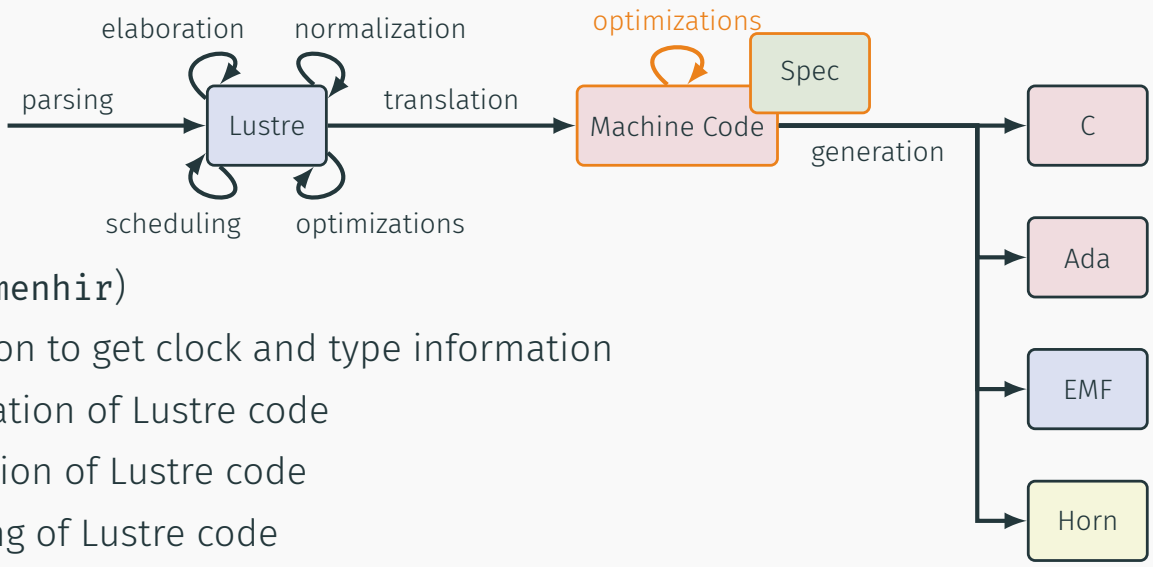
- parsing (`menhir`)
- elaboration to get clock and type information
- normalization of Lustre code
- optimization of Lustre code
- scheduling of Lustre code
- translation to Machine code
- optimisation of Machine code
- generation of Horn clauses

LustreC: a Certifying Lustre Compiler



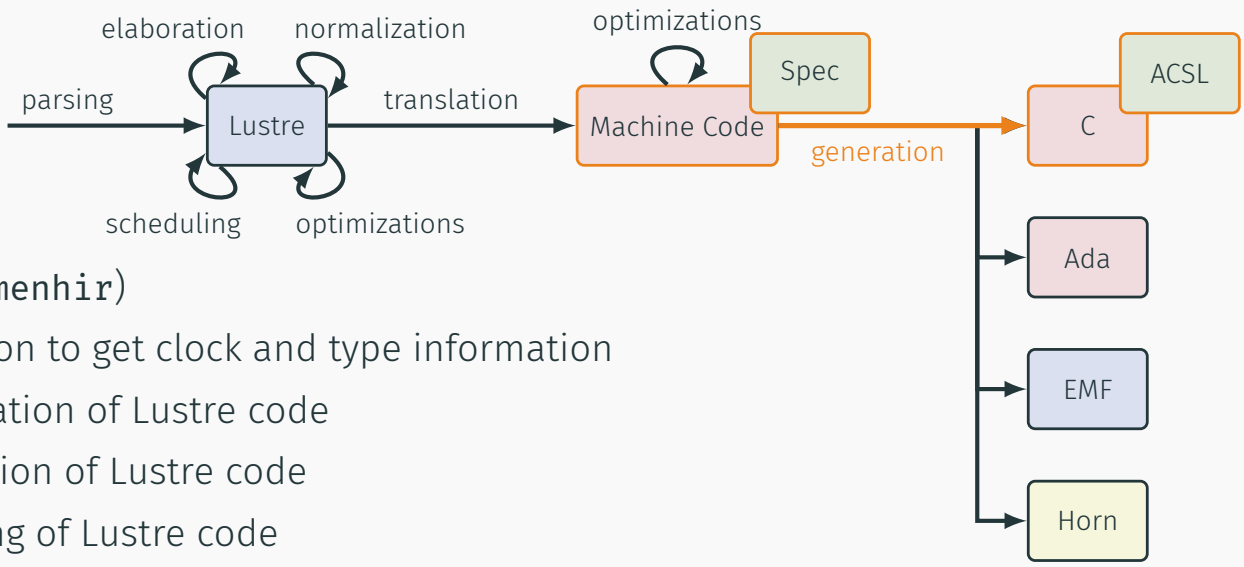
- parsing (`menhir`)
- elaboration to get clock and type information
- normalization of Lustre code
- optimization of Lustre code
- scheduling of Lustre code
- translation to Machine code
- optimisation of Machine code
- generation of C code

LustreC: a Certifying Lustre Compiler



- parsing (`menhir`)
- elaboration to get clock and type information
- normalization of Lustre code
- optimization of Lustre code
- scheduling of Lustre code
- translation to Machine code
- optimisation of Machine code
- generation of C code

LustreC: a Certifying Lustre Compiler



- parsing (`menhir`)
- elaboration to get clock and type information
- normalization of Lustre code
- optimization of Lustre code
- scheduling of Lustre code
- translation to Machine code
- optimisation of Machine code
- generation of C code + ACSL specification

Lustre

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

node ins(gps, xv: real)
  returns (x: real, alarm: bool)
  var pxa, xe: real; 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, xv: real, s: bool)
  returns (x: real, alarm: bool)
let
  automaton a
    state GPS:
      let
        x = gps;
        alarm = false;
      tel until s restart INS
    state INS:
      let
        (x, alarm) = ins(gps, xv);
      tel until s resume GPS
tel
```

Lustre

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

node ins(gps, xv: real)
  returns (x: real, alarm: bool)
  var pxa, xe: real; 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, xv: real, s: bool)
  returns (x: real, alarm: bool)
let
  automaton a
    state GPS:
      let
        x = gps;
        alarm = false;
      tel until s restart INS
    state INS:
      let
        (x, alarm) = ins(gps, xv);
      tel until s resume GPS
tel
```

C

Lustre

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

node ins(gps, xv: real)
  returns (x: real, alarm: bool)
  var pxa, xe: real; 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, xv: real, s: bool)
  returns (x: real, alarm: bool)
let
  automaton a
    state GPS:
      let
        x = gps;
        alarm = false;
      tel until s restart INS
    state INS:
      let
        (x, alarm) = ins(gps, xv);
      tel until s resume GPS
tel
```

C

+ separated code for the main loop

Lustre

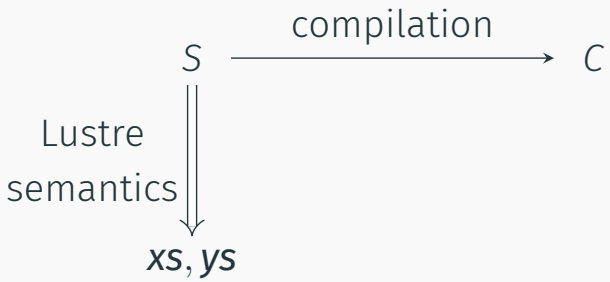
```
node euler(x0, u: real)
  returns (x: real);
let
  x = x0 fby (x + 0.1 * u);
tel

node ins(gps, xv: real)
  returns (x: real, alarm: bool)
  var pxa, xe: real; 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, xv: real, s: bool)
  returns (x: real, alarm: bool)
let
  automaton a
    state GPS:
      let
        x = gps;
        alarm = false;
      tel until s restart INS
    state INS:
      let
        (x, alarm) = ins(gps, xv);
      tel until s resume GPS
tel
```

C + ACSL

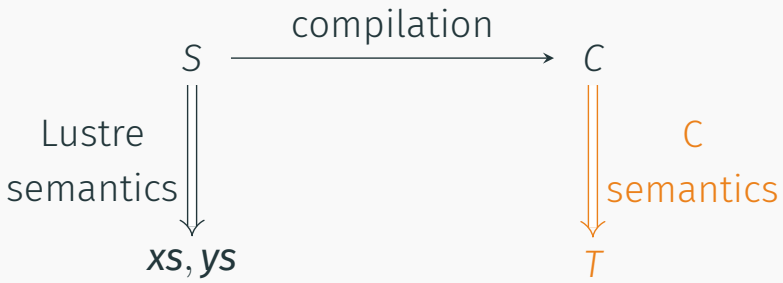
Correctness?

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

Correctness?



Correctness?



Correctness?

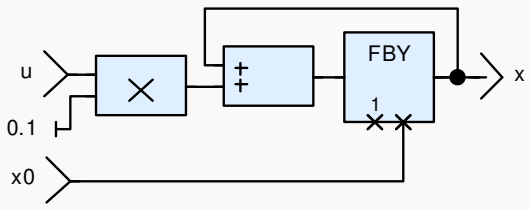
Correctness?

LustreC:

- no easy way to model streams in ACSL
- encoding of a state / transition semantics
- the simulation result is expressed as contracts

The Compilation Scheme

Example




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

Example: Normalization

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

```
node euler (x0: real; u: real)
  returns (x: real)
var euler_2: real; euler_1: bool;
let
  x = if euler_1 then x0 else euler_2;
  euler_2 = pre (x + 0.1 * u);
  euler_1 = true -> false;
tel
```

Example: Scheduling

```
node euler (x0: real; u: real)
  returns (x: real)
var euler_2: real; euler_1: bool;
let
  x = if euler_1 then x0 else euler_2;
  euler_2 = pre (x + 0.1 * u);
  euler_1 = true -> false;
tel
```

```
node euler (x0: real; u: real)
  returns (x: real)
var euler_2: real; euler_1: bool;
let
  euler_1 = true -> false;
  x = if euler_1 then x0 else euler_2;
  euler_2 = pre (x + 0.1 * u);
tel
```

Example: Machine code generation

```
node euler (x0: real; u: real)
  returns (x: real)
var euler_2: real; euler_1: bool;
let
  euler_1 = true -> false;
  x = if euler_1 then x0 else euler_2;
  euler_2 = pre (x + 0.1 * u);
tel
```

```
machine euler {
  mem euler_2: real;
  instance ni_0: arrow<>;

  step(x0, u: real) returns (x: double)
  var euler_1: bool;
  {
    euler_1 := ni_0(true, false);
    if euler_1 {
      x := x0
    } else {
      x := euler_2
    };
    euler_2 := x + 0.1 * u;
  }
}
```

Example: C code generation

```
machine euler {
  mem euler_2: real;
  instance ni_0: arrow<>;

  step(x0, u: real) returns (x: double)
  var euler_1: bool;
  {
    euler_1 := ni_0(true, false);
    if euler_1 {
      x := x0
    } else {
      x := euler_2
    };
    euler_2 := x + 0.1 * u;
  }
}
```

```
struct _arrow_mem {
  struct _arrow_reg {_Bool _first; } _reg;
};

#define _arrow_reset(self) \
  {(self)->_reg._first = 1;}

_Bool _arrow_step(struct _arrow_mem *self) {
  if (self->_reg._first) {
    self->_reg._first = 0;
    return 1;
  }
  return 0;
}
```

Example: C code generation

```
machine euler {
  mem euler_2: real;
  instance ni_0: arrow<>;

  step(x0, u: real) returns (x: double)
  var euler_1: bool;
  {
    euler_1 := ni_0(true, false);
    if euler_1 {
      x := x0
    } else {
      x := euler_2
    };
    euler_2 := x + 0.1 * u;
  }
}
```

```
struct euler_mem {
  _Bool _reset;
  struct euler_reg {double euler_2;} _reg;
  struct _arrow_mem *ni_0;
};
```

Example: C code generation

```
machine euler {
  mem euler_2: real;
  instance ni_0: arrow<>;

  step(x0, u: real) returns (x: double)
  var euler_1: bool;
  {
    euler_1 := ni_0(true, false);
    if euler_1 {
      x := x0
    } else {
      x := euler_2
    };
    euler_2 := x + 0.1 * u;
  }
}
```

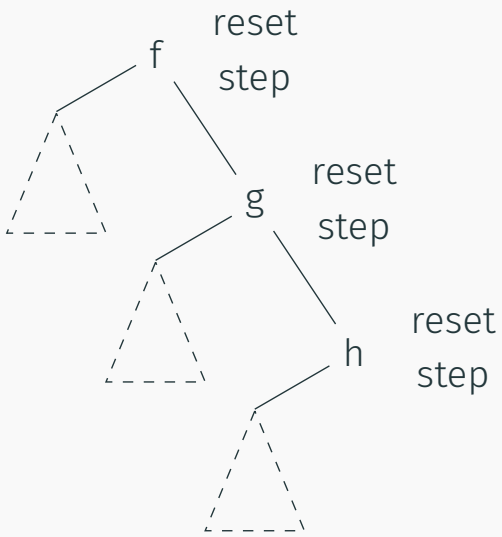
```
void euler_clear_reset
  (struct euler_mem *self)
{
  if (self->_reset) {
    self->_reset = 0;
    _arrow_reset(self->ni_0);
  }
  return;
}

void euler_set_reset
  (struct euler_mem *self)
{
  self->_reset = 1;
  return;
}
```

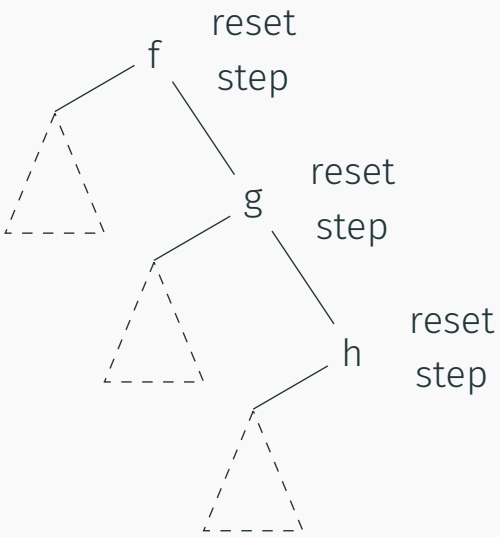
Example: C code generation

```
machine euler {
  mem euler_2: real;
  instance ni_0: arrow<>;

  step(x0, u: real) returns (x: double)
  var euler_1: bool;
  {
    euler_1 := ni_0(true, false);
    if euler_1 {
      x := x0
    } else {
      x := euler_2
    };
    euler_2 := x + 0.1 * u;
  }
}
```

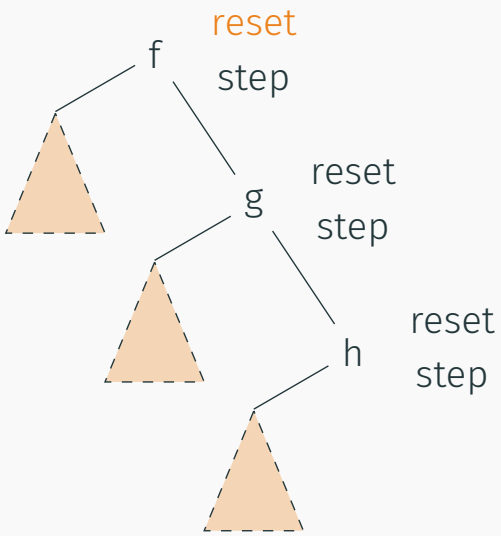
```
void euler_step(double x0, double u,
                double *x,
                struct euler_mem *self) {
  _Bool euler_1;
  euler_clear_reset(self);
  Reset:
  euler_1 = _arrow_step(self->ni_0);
  if (euler_1) {
    *x = x0;
  } else {
    *x = self->_reg.euler_2;
  }
  self->_reg.euler_2 = *x + 0.1 * u;
  return;
}
```

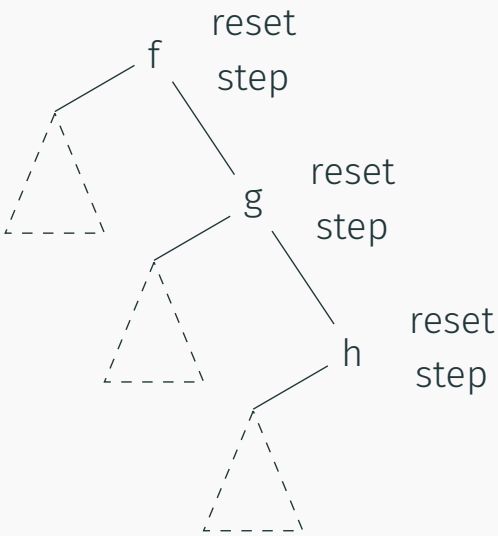
SCADE-like reset compilation scheme



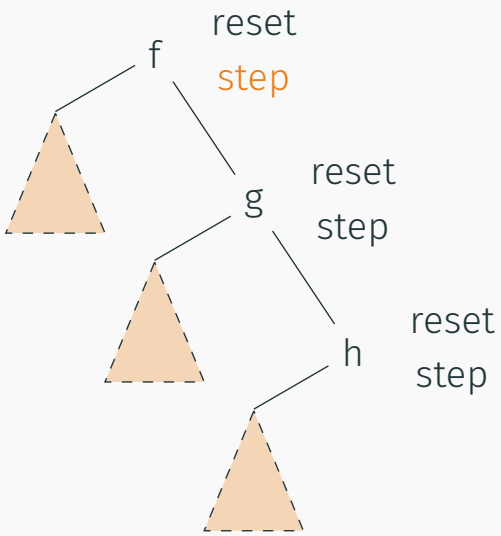


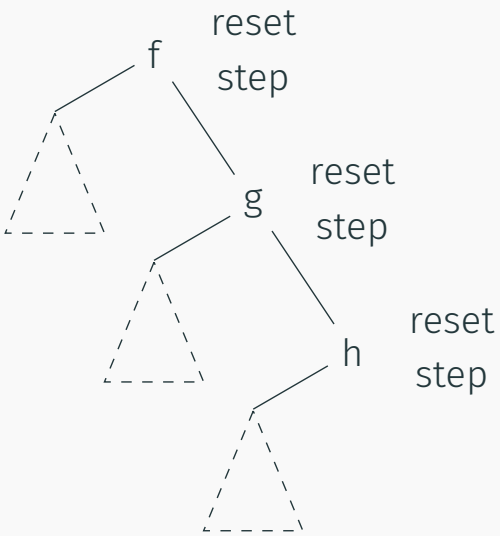
SCADE-like reset compilation scheme



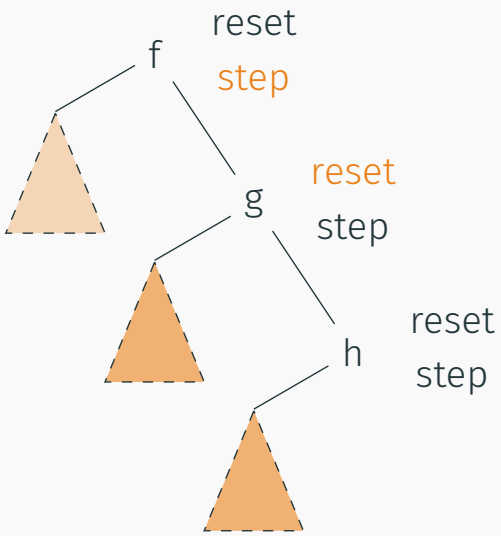


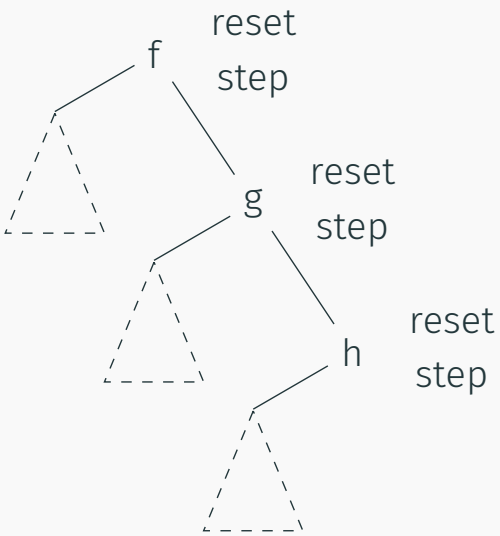
SCADE-like reset compilation scheme



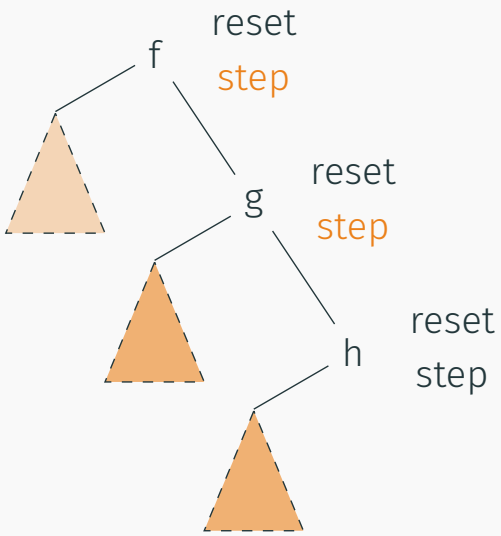


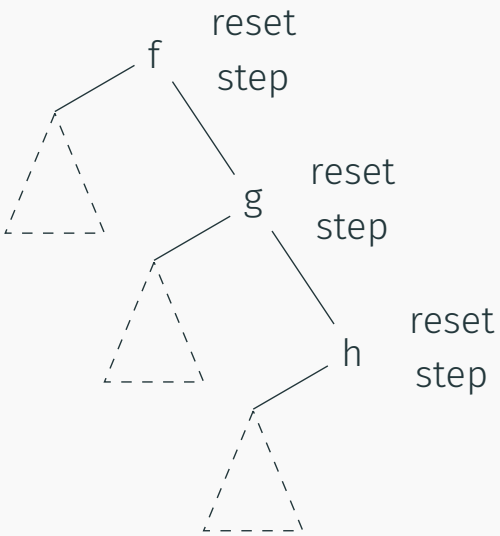
SCADE-like reset compilation scheme



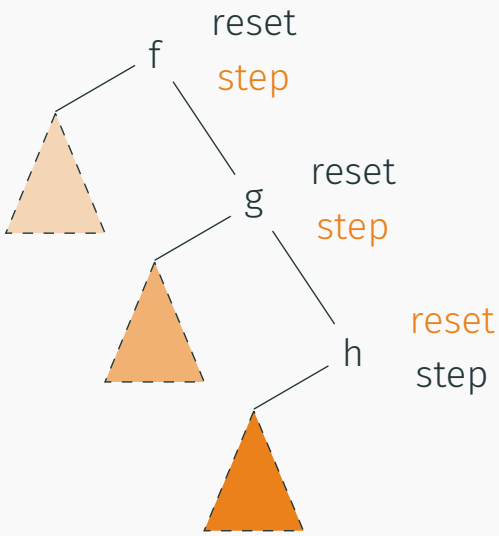


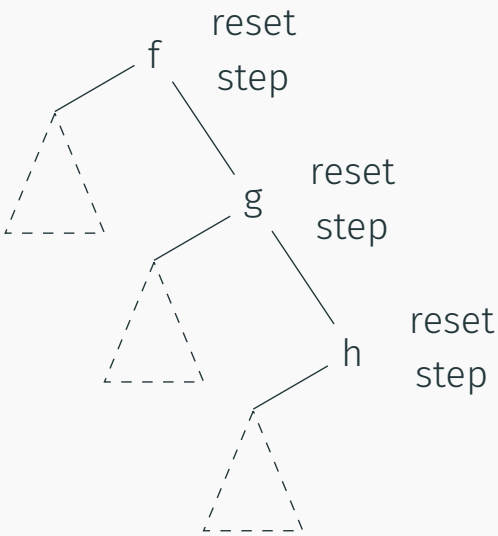
SCADE-like reset compilation scheme



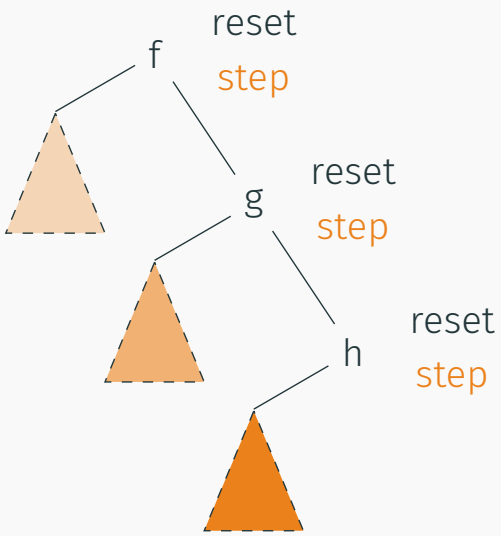


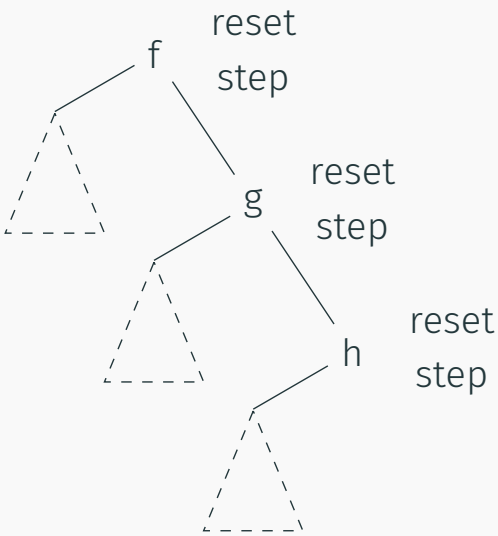
SCADE-like reset compilation scheme



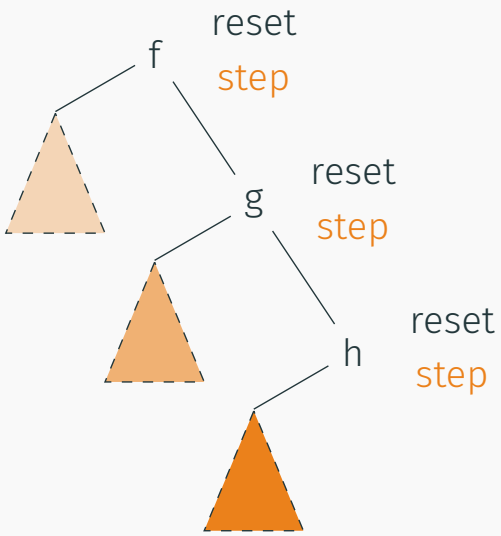


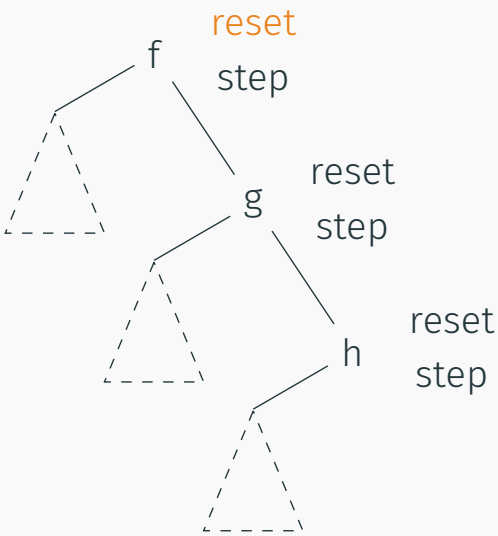
SCADE-like reset compilation scheme



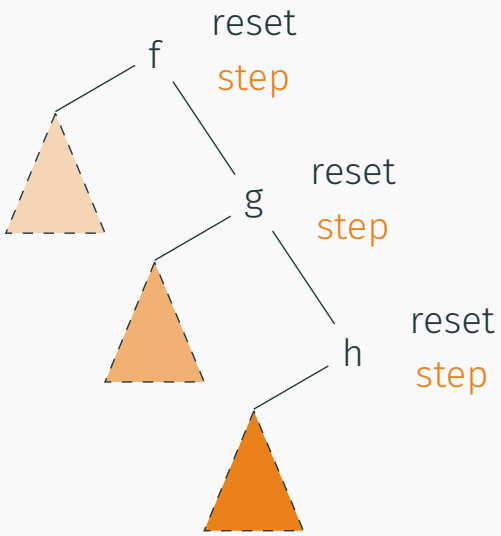


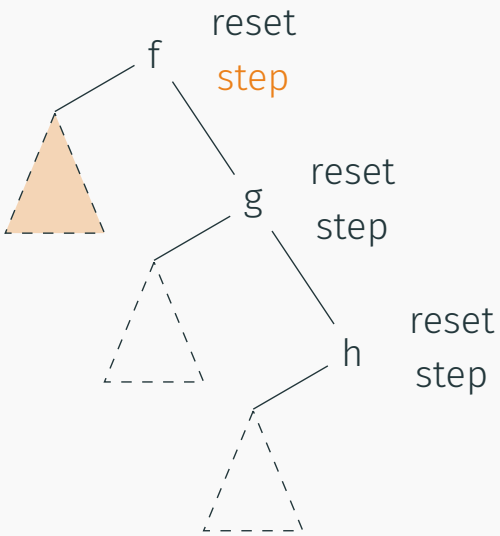
SCADE-like reset compilation scheme



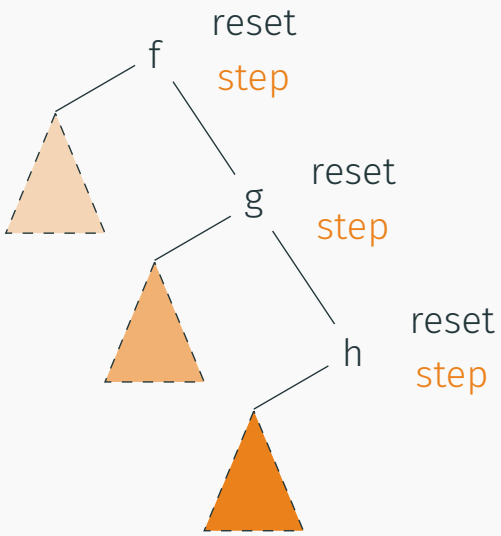


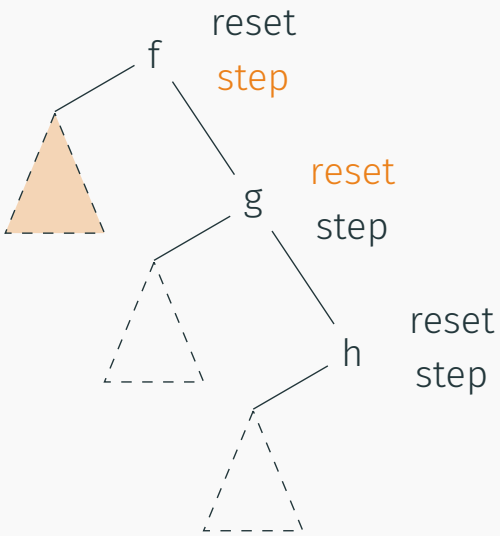
SCADE-like reset compilation scheme



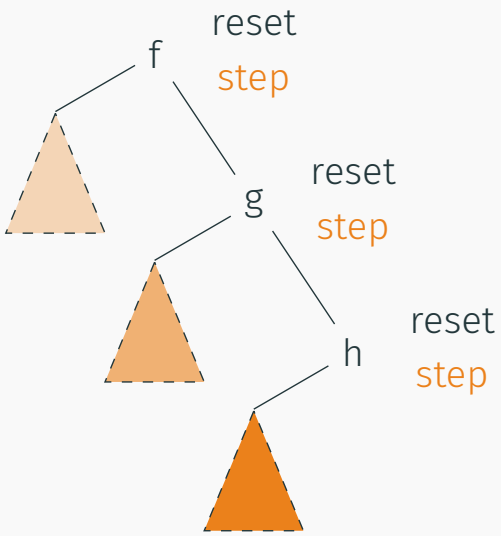


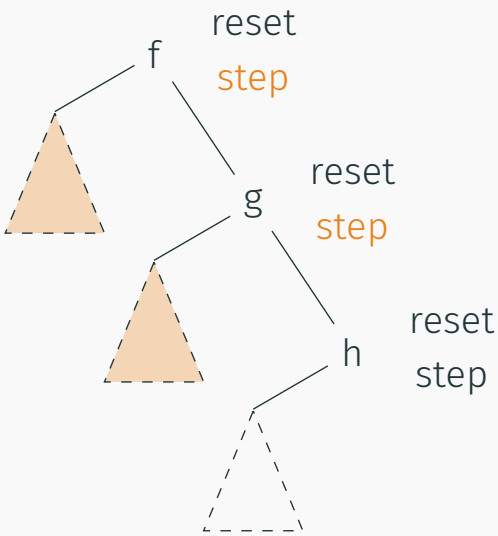
SCADE-like reset compilation scheme



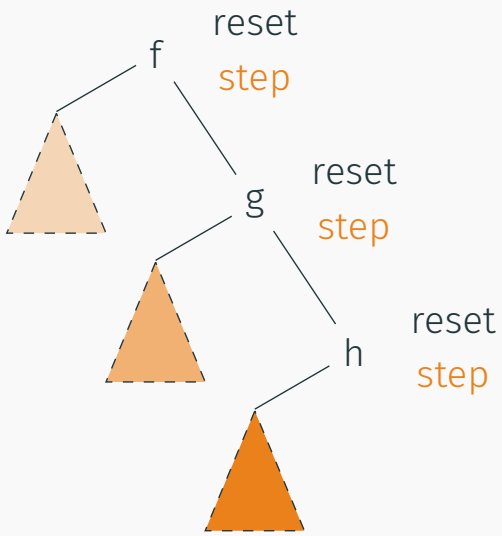


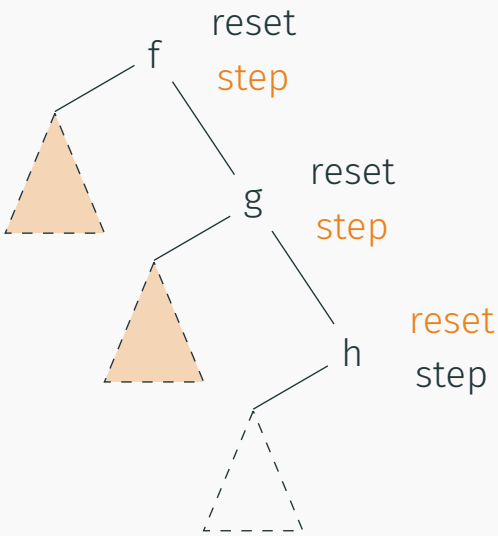
SCADE-like reset compilation scheme



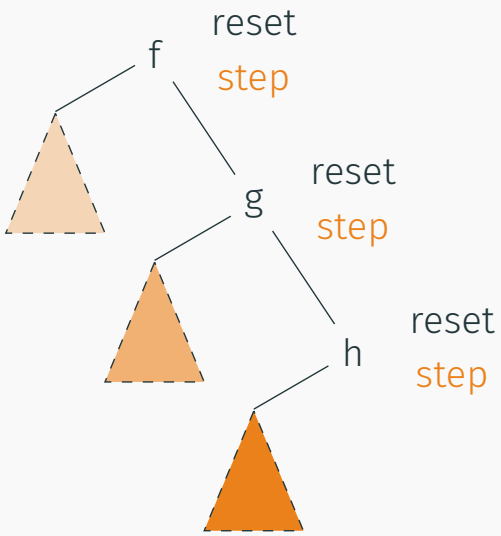


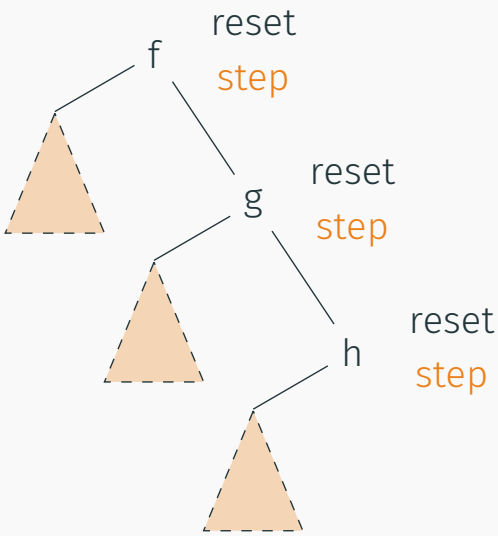
SCADE-like reset compilation scheme





SCADE-like reset compilation scheme





Encoding of the Semantics

State / Transition Semantics

```
node euler (x0: real; u: real)
  returns (x: real)
var euler_2: real; euler_1: bool;
let
  euler_1 = true -> false;
  x = if euler_1 then x0 else euler_2;
  euler_2 = pre (x + 0.1 * u);
tel
```

$$euler\_init(S) = arrow\_init(S[ni_0])$$

$$euler\_reset(S, S') = S(\_reset) \rightarrow S'(\_reset) = false \wedge euler\_init(S') \wedge \neg S(\_reset) \rightarrow S' = S$$

State / Transition Semantics

```
node euler (x0: real; u: real)
  returns (x: real)
var euler_2: real; euler_1: bool;
let
  euler_1 = true -> false;
  x = if euler_1 then x0 else euler_2;
  euler_2 = pre (x + 0.1 * u);
tel
```

$$
\begin{aligned}
&euler\_tr\left(S, x_0, u, x, S'\right) = \\
&\quad \exists S_r, \\
&\qquad euler\_reset\left(S, S_r\right) \\
&\qquad \wedge S'(\_reset) = S_r(\_reset) \\
&\qquad \wedge \exists euler_1, \\
&\qquad\qquad arrow\_tr\left(S_r[ni_0], euler_1, S'[ni_0]\right) \\
&\qquad\qquad \wedge euler_1 \rightarrow x = x_0 \\
&\qquad\qquad \wedge \neg euler_1 \rightarrow x = S_r(euler_2) \\
&\qquad\qquad \wedge S'(euler_2) = x + 0.1 \times u
\end{aligned}
$$

Machine code specification generation

- Internal AST for an ad-hoc first order logic
- Generation from each Lustre equation
- Optimizations

ACSL GENERATION: MEMORY CORRESPONDENCE

Notion of state in ACSL: same C structures, *flattened*

```
struct _arrow_mem {
  struct _arrow_reg {_Bool _first; } _reg;
};

struct euler_mem {
  _Bool _reset;
  struct euler_reg {double euler_2;} _reg;
  struct _arrow_mem *ni_0;
};
```

```
/*@ ghost struct _arrow_mem_ghost {
              struct _arrow_reg _reg;
    };
*/

/*@ ghost struct euler_mem_ghost {
              _Bool _reset;
              struct euler_reg _reg;
              struct _arrow_mem_ghost ni_0;
    };
*/
```

ACSL generation: memory correspondence

```
/*@ predicate euler_pack0(struct euler_mem_ghost mem, struct euler_mem *self) =
      mem._reset == self->_reset
      && mem._reset == 0;
*/
/*@ predicate euler_pack1(struct euler_mem_ghost mem, struct euler_mem *self) =
      euler_pack0(mem, self)
      && _arrow_pack(mem.ni_0, self->ni_0);
*/
/*@ predicate euler_pack2(struct euler_mem_ghost mem, struct euler_mem *self) =
      euler_pack1(mem, self);
*/
/*@ predicate euler_pack3(struct euler_mem_ghost mem, struct euler_mem *self) =
      euler_pack2(mem, self)
      && !_arrow_initialization(mem.ni_0) ==> mem._reg.euler_2 == self->_reg.euler_2;
*/
/*@ predicate euler_pack(struct euler_mem_ghost mem, struct euler_mem *self) =
      (mem._reset ? mem._reset == self->_reset : euler_pack3(mem, self));
*/
```

ACSL generation: transition relation

$$euler\_init\,(S) =$$
$$\quad arrow\_init\,(S[ni_0])$$

$$euler\_reset\,(S, S') =$$
$$\quad S(\_reset) \rightarrow S'(\_reset) = false$$
$$\qquad\qquad \wedge\, euler\_init(S')$$
$$\quad \wedge\, \neg S(\_reset) \rightarrow S' = S$$

```
/*@ predicate euler_init(struct euler_mem_ghost mem_in) =
      _arrow_init(mem_in.ni_0);
*/

/*@ predicate euler_reset(struct euler_mem_ghost mem_in,
                          struct euler_mem_ghost mem_out) =
      mem_in._reset ? mem_out._reset == 0
                      && euler_init(mem_out)
                    : mem_out == mem_in;
*/
```

ACSL generation: transition relation

$$
\begin{aligned}
&\mathit{euler\_tr}\,(S, x_0, u, x, S') = \\
&\quad \exists S_r, \\
&\qquad \mathit{euler\_reset}\,(S, S_r) \\
&\qquad \wedge\, S'(\_reset) = S_r(\_reset) \\
&\qquad \wedge\, \exists \mathit{euler}_1, \\
&\qquad\quad \wedge\, \mathit{arrow\_tr}\,(S_r[ni_0], \mathit{euler}_1, S'[ni_0]) \\
&\qquad\quad \wedge\, \mathit{euler}_1 \rightarrow x = x_0 \\
&\qquad\quad \wedge\, \neg \mathit{euler}_1 \rightarrow x = S_r(\mathit{euler}_2) \\
&\qquad\quad \wedge\, S'(\mathit{euler}_2) = x + 0.1 \times u
\end{aligned}
$$

```
/*@ predicate euler_tr(struct euler_mem_ghost mem_in,
                       double x0, double u, double x,
                       struct euler_mem_ghost mem_out) =
      \exists struct euler_mem_ghost mem_reset;
        euler_reset(mem_in, mem_reset)
        && euler_tr3(mem_reset, x0, u, x, mem_out);
*/
```

ACSL generation: transition relation

$$
\begin{aligned}
&\mathit{euler\_tr}\left(S, x_0, u, x, S'\right) = \\
&\quad \exists S_r, \\
&\qquad \mathit{euler\_reset}\left(S, S_r\right) \\
&\qquad \wedge\, S'(\mathit{\_reset}) = S_r(\mathit{\_reset}) \\
&\qquad \wedge\, \exists \mathit{euler}_1, \\
&\qquad\quad \wedge\, \mathit{arrow\_tr}\left(S_r[ni_0], \mathit{euler}_1, S'[ni_0]\right) \\
&\qquad\quad \wedge\, \mathit{euler}_1 \rightarrow x = x_0 \\
&\qquad\quad \wedge\, \neg \mathit{euler}_1 \rightarrow x = S_r(\mathit{euler}_2) \\
&\qquad\quad \wedge\, S'(\mathit{euler}_2) = x + 0.1 \times u
\end{aligned}
$$

```
/*@ predicate euler_tr3(struct euler_mem_ghost mem_in,
                        double x0, double u, double x,
                        struct euler_mem_ghost mem_out) =
      euler_tr2(mem_in, x0, u, x, mem_out)
      && mem_out._reg.euler_2 == x + 0.1 * u;
*/
```

ACSL generation: transition relation

$$
\begin{aligned}
&\mathit{euler\_tr}\left(S, x_0, u, x, S'\right) = \\
&\quad \exists S_r, \\
&\qquad \mathit{euler\_reset}\left(S, S_r\right) \\
&\qquad \wedge S'(\mathit{\_reset}) = S_r(\mathit{\_reset}) \\
&\qquad \wedge \exists \mathit{euler}_1, \\
&\qquad\quad \wedge \mathit{arrow\_tr}\left(S_r[\mathit{ni}_0], \mathit{euler}_1, S'[\mathit{ni}_0]\right) \\
&\qquad\quad \wedge \mathit{euler}_1 \rightarrow x = x_0 \\
&\qquad\quad \wedge \neg \mathit{euler}_1 \rightarrow x = S_r(\mathit{euler}_2) \\
&\qquad\quad \wedge S'(\mathit{euler}_2) = x + 0.1 \times u
\end{aligned}
$$

```
/*@ predicate euler_tr2(struct euler_mem_ghost mem_in,
                        double x0, double u, double x,
                        struct euler_mem_ghost mem_out) =
    \exists _Bool euler_1;
        euler_tr1(mem_in, x0, u, euler_1, mem_out)
        && euler_1 ? x == x0
                   : !_arrow_init(mem_in.ni_0) ==>
                     x == mem_in._reg.euler_2;
*/
```

ACSL generation: transition relation

$$
\begin{aligned}
&euler\_tr\,(S, x_0, u, x, S') = \\
&\quad \exists S_r, \\
&\qquad euler\_reset\,(S, S_r) \\
&\qquad \wedge\, S'(\_reset) = S_r(\_reset) \\
&\qquad \wedge\, \exists euler_1, \\
&\qquad\quad \wedge\, arrow\_tr\,(S_r[ni_0], euler_1, S'[ni_0]) \\
&\qquad\quad \wedge\, euler_1 \rightarrow x = x_0 \\
&\qquad\quad \wedge\, \neg euler_1 \rightarrow x = S_r(euler_2) \\
&\qquad\quad \wedge\, S'(euler_2) = x + 0.1 \times u
\end{aligned}
$$

```
/*@ predicate euler_tr1(struct euler_mem_ghost mem_in,
                         double x0, double u, _Bool euler_1,
                         struct euler_mem_ghost mem_out) =
      euler_tr0(mem_in, x0, u, mem_out)
      && _arrow_tr(mem_in.ni_0, euler_1, mem_out.ni_0);
*/
```

ACSL GENERATION: TRANSITION RELATION

$$
\begin{aligned}
&\mathit{euler\_tr}\,(S, x_0, u, x, S') = \\
&\quad \exists S_r, \\
&\qquad \mathit{euler\_reset}\,(S, S_r) \\
&\qquad \wedge\, S'(\mathit{\_reset}) = S_r(\mathit{\_reset}) \\
&\qquad \wedge\, \exists \mathit{euler}_1, \\
&\qquad\quad \wedge\, \mathit{arrow\_tr}\,(S_r[ni_0], \mathit{euler}_1, S'[ni_0]) \\
&\qquad\quad \wedge\, \mathit{euler}_1 \rightarrow x = x_0 \\
&\qquad\quad \wedge\, \neg \mathit{euler}_1 \rightarrow x = S_r(\mathit{euler}_2) \\
&\qquad\quad \wedge\, S'(\mathit{euler}_2) = x + 0.1 \times u
\end{aligned}
$$

```
/*@ predicate euler_tr0(struct euler_mem_ghost mem_in,
                         double x0, double u,
                         struct euler_mem_ghost mem_out) =
      mem_out._reset == mem_in._reset; */
```

ACSL generation: contracts and assertions

```
/*@ requires euler_valid(self);
    requires \separated(self, mem,
                        self->ni_0);
    requires euler_pack(*mem, self);
    ensures euler_pack3(*mem, self);
    assigns self->_reset, self->ni_0->_reg,
      mem->_reset, mem->ni_0._reg;
    behavior reset:
      assumes mem->_reset == 1;
      ensures euler_init(*mem);
    behavior no_reset:
      assumes mem->_reset == 0;
      ensures *mem == \old(*mem);
    complete behaviors;
    disjoint behaviors;
*/
```

```
void euler_clear_reset
  (struct euler_mem *self)
/*@ ghost
  (struct euler_mem_ghost \ghost *mem) */
{
  if (self->_reset) {
    self->_reset = 0;
    //@ ghost mem->_reset = 0;
    _arrow_reset(self->ni_0);
    //@ ghost _arrow_reset_ghost(mem->ni_0);
  }
  return;
}
```

ACSL generation: contracts and assertions

```
/*@ ensures euler_pack(*mem, self);
    ensures mem->_reset == 1;
    assigns self->_reset, mem->_reset;
*/
void euler_set_reset(struct euler_mem *self)
/*@ ghost (struct euler_mem_ghost \ghost *mem) */ {
  self->_reset = 1;
  //@ ghost mem->_reset = 1;
  return;
}
```

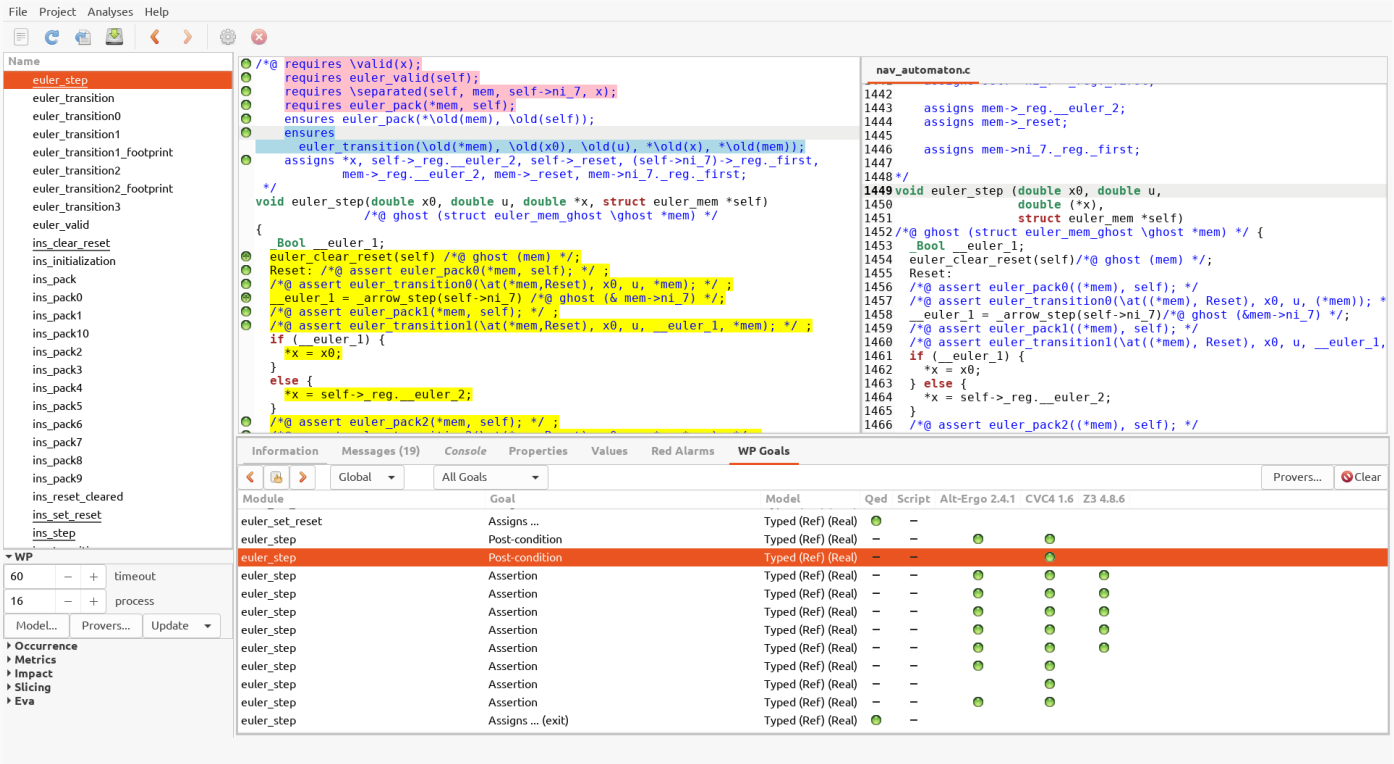
ACSL generation: contracts and assertions

```
/*@ requires \valid(x);
    requires euler_valid(self);
    requires \separated(self, mem, self->ni_0, x);
    requires euler_pack(*mem, self);
    ensures euler_pack(*mem, self);
    ensures euler_tr(\old(*mem), x0, u, *x, *mem);
    assigns *x, self->_reg, self->_reset, self->ni_0->_reg,
      mem->_reg, mem->_reset, mem->ni_0._reg;
*/
```

ACSL GENERATION: CONTRACTS AND ASSERTIONS

```
void euler_step(double x0, double u,
                double *x,
                struct euler_mem *self)
/*@ ghost
  (struct euler_mem_ghost \ghost *mem) */
{
  _Bool euler_1;
  euler_clear_reset(self)/*@ ghost (mem) */;
  Reset:
  //@ assert euler_pack0(*mem, self);
  /*@ assert euler_tr0
        (\at(*mem, Reset), x0, u, *mem); */
  euler_1 = _arrow_step(self->ni_0)
             /*@ ghost (&mem->ni_0) */;
  //@ assert euler_pack1(*mem, self);
  /*@ assert euler_tr1
        (\at(*mem, Reset),
         x0, u, euler_1, *mem); */
  if (euler_1) {
    *x = x0;
  } else {
    *x = self->_reg.euler_2;
  }
  //@ assert euler_pack2((*mem), self);
  /*@ assert euler_tr2
        (\at(*mem, Reset), x0, u, *x, *mem); */
  self->_reg.euler_2 = *x + 0.1 * u;
  //@ ghost mem->_reg.euler_2 = *x + 0.1 * u;
  //@ assert euler_pack3(*mem, self);
  /*@ assert euler_tr3
        (\at(*mem, Reset), x0, u, *x, *mem); */
  return;
}
```

FRAMA-C


File Project Analyses Help
Name
euler_step
euler_transition
euler_transition0
euler_transition1
euler_transition1_footprint
euler_transition2
euler_transition2_footprint
euler_transition3
euler_valid
ins_clear_reset
ins_initialization
ins_pack
ins_pack0
ins_pack1
ins_pack10
ins_pack2
ins_pack3
ins_pack4
ins_pack5
ins_pack6
ins_pack7
ins_pack8
ins_pack9
ins_reset_cleared
ins_set_reset
ins_step
WP
60 timeout
16 process
Model... Provers... Update
Occurrence
Metrics
Impact
Slicing
Eva
/*@ requires \valid(x);
requires euler_valid(self);
requires \separated(self, mem, self->ni_7, x);
requires euler_pack(*mem, self);
ensures euler_pack(*\old(mem), \old(self));
ensures
euler_transition(\old(*mem), \old(x0), \old(u), *\old(x), *\old(mem));
assigns *x, self->_reg.__euler_2, self->_reset, (self->ni_7)->_reg._first,
mem->_reg.__euler_2, mem->_reset, mem->ni_7._reg._first;
*/
void euler_step(double x0, double u, double *x, struct euler_mem *self)
/*@ ghost (struct euler_mem_ghost \ghost *mem) */
{
_Bool __euler_1;
euler_clear_reset(self) /*@ ghost (mem) */;
Reset: /*@ assert euler_pack0(*mem, self); */ ;
/*@ assert euler_transition0(\at(*mem,Reset), x0, u, *mem); */ ;
__euler_1 = _arrow_step(self->ni_7) /*@ ghost (& mem->ni_7) */;
/*@ assert euler_pack1(*mem, self); */ ;
/*@ assert euler_transition1(\at(*mem,Reset), x0, u, __euler_1, *mem); */ ;
if (__euler_1) {
*x = x0;
}
else {
*x = self->_reg.__euler_2;
}
/*@ assert euler_pack2(*mem, self); */ ;
nav_automaton.c
1442
1443 assigns mem->_reg.__euler_2;
1444 assigns mem->_reset;
1445
1446 assigns mem->ni_7._reg._first;
1447
1448 */
1449 void euler_step (double x0, double u,
1450 double (*x),
1451 struct euler_mem *self)
1452 /*@ ghost (struct euler_mem_ghost \ghost *mem) */ {
1453 _Bool __euler_1;
1454 euler_clear_reset(self)/*@ ghost (mem) */;
1455 Reset:
1456 /*@ assert euler_pack0((*mem), self); */
1457 /*@ assert euler_transition0(\at((*mem), Reset), x0, u, (*mem)); *
1458 __euler_1 = _arrow_step(self->ni_7)/*@ ghost (&mem->ni_7) */;
1459 /*@ assert euler_pack1((*mem), self); */
1460 /*@ assert euler_transition1(\at((*mem), Reset), x0, u, __euler_1,
1461 if (__euler_1) {
1462 *x = x0;
1463 } else {
1464 *x = self->_reg.__euler_2;
1465 }
1466 /*@ assert euler_pack2((*mem), self); */
Information Messages (19) Console Properties Values Red Alarms WP Goals
Global All Goals Provers... Clear
Module Goal Model Qed Script Alt-Ergo 2.4.1 CVC4 1.6 Z3 4.8.6
euler_set_reset Assigns ... Typed (Ref) (Real) – –
euler_step Post-condition Typed (Ref) (Real) – –
euler_step Post-condition Typed (Ref) (Real) – –
euler_step Assertion Typed (Ref) (Real) – –
euler_step Assertion Typed (Ref) (Real) – –
euler_step Assertion Typed (Ref) (Real) – –
euler_step Assertion Typed (Ref) (Real) – –
euler_step Assertion Typed (Ref) (Real) – –
euler_step Assertion Typed (Ref) (Real) – –
euler_step Assertion Typed (Ref) (Real) – –
euler_step Assertion Typed (Ref) (Real) – –
euler_step Assertion Typed (Ref) (Real) – –
euler_step Assigns ... (exit) Typed (Ref) (Real) – –

Contract Translation

Principles

High-level specification

- contracts expressed at dataflow level
- verified by model checking techniques (eg. Kind2)

Principles

High-level specification

- contracts expressed at dataflow level
- verified by model checking techniques (eg. Kind2)

Logical meaning

- synchronous observers
- $\square\left(\mathcal{H}\left(Pre(S,x)\right) \wedge tr(S,x,y,S') \rightarrow Post(S,x,y,S')\right)$

Principles

High-level specification

- contracts expressed at dataflow level
- verified by model checking techniques (eg. Kind2)

Logical meaning

- synchronous observers
- $\square\left(\mathcal{H}\left(Pre(S,x)\right) \wedge tr(S,x,y,S') \rightarrow Post(S,x,y,S')\right)$

Code-level specification

- encoding as specific transition predicates and contracts in ACSL
- focus: k-induction

Example

```
node greycounter (x:bool) returns (out:bool);
var a,b:bool;
let
  a = false -> not pre(b);
  b = false -> pre(a);
  out = a and b;
tel

node intloopcounter (x:bool) returns (out:bool);
var time: int;
let
  time = 0 -> if pre(time) = 3 then 0
              else pre time + 1;
  out = (time = 2);
tel

node top (x:bool) returns (OK:bool);
--@ guarantee OK;
var b,d:bool;
let
  b = greycounter(x);
  d = intloopcounter(x);
  OK = b = d;
  --%PROPERTY OK;
tel
```

a	*F*	*T*	*T*	*F*	*F*	···
b	*F*	*F*	*T*	*T*	*F*	···
out	*F*	*F*	*T*	*F*	*F*	···
time	0	1	2	3	0	···
out	*F*	*F*	*T*	*F*	*F*	···
OK	*T*	*T*	*T*	*T*	*T*	···

Example: Model-checking

```
node greycounter (x:bool) returns (out:bool);
var a,b:bool;
let
  a = false -> not pre(b);
  b = false -> pre(a);
  out = a and b;
tel

node intloopcounter (x:bool) returns (out:bool);
var time: int;
let
  time = 0 -> if pre(time) = 3 then 0
              else pre time + 1;
  out = (time = 2);
tel

node top (x:bool) returns (OK:bool);
--@ guarantee OK;
var b,d:bool;
let
  b = greycounter(x);
  d = intloopcounter(x);
  OK = b = d;
  --%PROPERTY OK;
tel
```

```
> kind2 --enable BMC --enable IND two_counters.lus
 kind2 v1.5.0

==========================================================================
Analyzing top
  with First top: 'top'
              subsystems
                | concrete: intloopcounter, greycounter

<Success> Property OK is valid by inductive step after 0.062s.

--------------------------------------------------------------------------
Summary of properties:
--------------------------------------------------------------------------
OK: valid (at 5)
==========================================================================
```

Example: k-induction annotation

```
node top (x:bool) returns (OK:bool);
--@ guarantee OK by 5-induction;
var b,d:bool;
let
  b = greycounter(x);
  d = intloopcounter(x);
  OK = b = d;
tel
```

Example: ACSL generation

Contract node compiled as a transition predicate

```
/*@ predicate top_contract_tr0(_Bool x, _Bool OK) =
      \true;
*/
/*@ predicate top_contract_tr1(_Bool x, _Bool OK, _Bool spec36) =
      top_contract_tr0(x, OK)
      && spec36 == OK;
*/
/*@ predicate top_contract_tr(_Bool x, _Bool OK, _Bool spec36) =
      top_contract_tr1(x, OK, spec36);
*/
```

Example: ACSL generation

Requirements as contracts

```
//@ ghost _Bool spec36;
/*@ ...
    requires top_pack(*mem, self);
    ensures top_pack(*mem, self);
    ensures top_tr(\old(*mem), x, *OK, *mem);
    ensures top_contract_tr(x, *OK, spec36) ==>
            spec36;
    ...
*/
void top_step(_Bool x, _Bool *OK, struct top_mem *self)
/*@ ghost (struct top_mem_ghost \ghost *mem) */ {
```

Example: ACSL generation

k-induction base cases predicates

```
predicate top_full_init(struct top_mem_ghost mem) =
   top_init(mem) && mem._reset;

 predicate top_base_1(struct top_mem_ghost mem_0,
                      _Bool x_1, _Bool OK_1,
                      struct top_mem_ghost mem_1) =
   \forall _Bool spec36_1;
     top_full_init(mem_0)
     && top_tr(mem_0, x_1, OK_1, mem_1)
     && top_contract_tr(x_1, OK_1, spec36_1) ==>
     spec36_1;
```

Example: ACSL generation

k-induction base cases predicates

```
predicate top_base_2(struct top_mem_ghost mem_1,
                     _Bool x_2, _Bool OK_2,
                     struct top_mem_ghost mem_2) =
  \forall _Bool spec36_2;
    \forall struct top_mem_ghost mem_0, _Bool x_1, _Bool OK_1;
      top_full_init(mem_0)
      && top_tr(mem_0, x_1, OK_1, mem_1)
      && top_contract_tr(x_1, OK_1, (_Bool) 1)
      && top_tr(mem_1, x_2, OK_2, mem_2)
      && top_contract_tr(x_2, OK_2, spec36_2) ==>
      spec36_2;
```

Example: ACSL generation

k-induction base cases predicates

```
predicate top_base_3(struct top_mem_ghost mem_2,
                     _Bool x_3, _Bool OK_3,
                     struct top_mem_ghost mem_3) =
  \forall _Bool spec36_3;
    \forall struct top_mem_ghost mem_0, mem_1,
             _Bool x_1, _Bool OK_1, _Bool x_2, _Bool OK_2;
      top_full_init(mem_0)
      && top_tr(mem_0, x_1, OK_1, mem_1)
      && top_contract_tr(x_1, OK_1, (_Bool) 1)
      && top_tr(mem_1, x_2, OK_2, mem_2)
      && top_contract_tr(x_2, OK_2, (_Bool) 1)
      && top_tr(mem_2, x_3, OK_3, mem_3)
      && top_contract_tr(x_3, OK_3, spec36_3) ==>
      spec36_3;
```

Example: ACSL generation

k-induction base cases predicates

```
predicate top_base_4(struct top_mem_ghost mem_3,
                     _Bool x_4, _Bool OK_4,
                     struct top_mem_ghost mem_4) =
  \forall _Bool spec36_4;
    \forall struct top_mem_ghost mem_0, mem_1, mem_2,
            _Bool x_1, _Bool OK_1, _Bool x_2, _Bool OK_2, _Bool x_3,
            _Bool OK_3;
      top_full_init(mem_0)
      && top_tr(mem_0, x_1, OK_1, mem_1)
      && top_contract_tr(x_1, OK_1, (_Bool) 1)
      && top_tr(mem_1, x_2, OK_2, mem_2)
      && top_contract_tr(x_2, OK_2, (_Bool) 1)
      && top_tr(mem_2, x_3, OK_3, mem_3)
      && top_contract_tr(x_3, OK_3, (_Bool) 1)
      && top_tr(mem_3, x_4, OK_4, mem_4)
      && top_contract_tr(x_4, OK_4, spec36_4) ==>
      spec36_4;
```

Example: ACSL generation

k-induction inductive case predicate

```
predicate top_inductive_5(struct top_mem_ghost mem_4,
                          _Bool x_5, _Bool OK_5,
                          struct top_mem_ghost mem_5) =
  \forall _Bool spec36_5;
    \forall struct top_mem_ghost mem_0, mem_1, mem_2, mem_3,
            _Bool x_1, _Bool OK_1, _Bool x_2, _Bool OK_2, _Bool x_3,
            _Bool OK_3, _Bool x_4, _Bool OK_4;
      top_tr(mem_0, x_1, OK_1, mem_1)
      && top_contract_tr(x_1, OK_1, (_Bool) 1)
      && top_tr(mem_1, x_2, OK_2, mem_2)
      && top_contract_tr(x_2, OK_2, (_Bool) 1)
      && top_tr(mem_2, x_3, OK_3, mem_3)
      && top_contract_tr(x_3, OK_3, (_Bool) 1)
      && top_tr(mem_3, x_4, OK_4, mem_4)
      && top_contract_tr(x_4, OK_4, (_Bool) 1)
      && top_tr(mem_4, x_5, OK_5, mem_5)
      && top_contract_tr(x_5, OK_5, spec36_5) ==>
      spec36_5;
```

Example: ACSL generation

k-induction corresponding lemmas

```
lemma top_k_induction_1:
  \forall struct top_mem_ghost mem_in, mem_out, _Bool x, _Bool OK;
    top_base_1(mem_in, x, OK, mem_out);

lemma top_k_induction_2:
  \forall struct top_mem_ghost mem_in, mem_out, _Bool x, _Bool OK;
    top_base_2(mem_in, x, OK, mem_out);

lemma top_k_induction_3:
  \forall struct top_mem_ghost mem_in, mem_out, _Bool x, _Bool OK;
    top_base_3(mem_in, x, OK, mem_out);

lemma top_k_induction_4:
  \forall struct top_mem_ghost mem_in, mem_out, _Bool x, _Bool OK;
    top_base_4(mem_in, x, OK, mem_out);

lemma top_k_induction_5:
  \forall struct top_mem_ghost mem_in, mem_out, _Bool x, _Bool OK;
    top_inductive_5(mem_in, x, OK, mem_out);
```

Example: ACSL generation

k-induction principle as an axiom

```
axiomatic top_k_Induction {
  axiom top_k_induction:
    \forall struct top_mem_ghost mem_in, mem_out, _Bool x, _Bool OK;
      top_base_1(mem_in, x, OK, mem_out)
      && top_base_2(mem_in, x, OK, mem_out)
      && top_base_3(mem_in, x, OK, mem_out)
      && top_base_4(mem_in, x, OK, mem_out)
      && top_inductive_5(mem_in, x, OK, mem_out) ==>
      \forall _Bool spec36;
        top_tr(mem_in, x, OK, mem_out)
        && top_contract_tr(x, OK, spec36) ==>
        spec36;
}
```

Stateful contract example

```
node fibo(x: bool) returns(a: int);
--@ guarantee a = (0 -> pre a) + (0 -> pre (1 -> pre a));
var b: int;
let
  a = 0 -> pre b;
  b = 1 -> pre (a + b);
  --%PROPERTY a = (0 -> pre a) + (0 -> pre (1 -> pre a));
tel
```

Stateful contract example

```
node fibo(x: bool) returns(a: int);
--@ guarantee a = (0 -> pre a) + (0 -> pre (1 -> pre a));
var b: int;
let
  a = 0 -> pre b;
  b = 1 -> pre (a + b);
  --%PROPERTY a = (0 -> pre a) + (0 -> pre (1 -> pre a));
tel
```

```
> kind2 --enable BMC --enable IND fibo.lus
 kind2 v1.5.0

==========================================================================================================================
Analyzing fibo
  with First top: 'fibo' (no subsystems)

<Success> Property (a = ((0 -> (pre a)) + (0 -> (pre (1 -> (pre a)))))) is valid by inductive step after 0.05

--------------------------------------------------------------------------------------------------------------------------
Summary of properties:
--------------------------------------------------------------------------------------------------------------------------
(a = ((0 -> (pre a)) + (0 -> (pre (1 -> (pre a)))))): valid (at 3)
==========================================================================================================================
```

Stateful contract example

```
node fibo(x: bool) returns(a: int);
--@ guarantee a = (0 -> pre a) + (0 -> pre (1 -> pre a)) by 3-induction;
var b: int;
let
  a = 0 -> pre b;
  b = 1 -> pre (a + b);
tel
```

Stateful contract example

Much more complex contract transition predicate

```
/*@ predicate fibo_contract_tr0(struct fibo_contract_mem_ghost mem_in,
                                 _Bool x, integer a,
                                 struct fibo_contract_mem_ghost mem_out) =
      mem_out._reset == mem_in._reset;
*/
/*@ predicate fibo_contract_tr1(struct fibo_contract_mem_ghost mem_in,
                                 _Bool x, integer a,
                                 _Bool __fibo_contract_7,
                                 struct fibo_contract_mem_ghost mem_out) =
      fibo_contract_tr0(mem_in, x, a, mem_out)
      && _arrow_transition(mem_in.ni_4, __fibo_contract_7, mem_out.ni_4);
*/
/*@ predicate fibo_contract_tr2(struct fibo_contract_mem_ghost mem_in,
                                 _Bool x, integer a,
                                 integer __fibo_contract_9,
                                 struct fibo_contract_mem_ghost mem_out) =
      \exists _Bool __fibo_contract_7;
        fibo_contract_tr1(mem_in, x, a, __fibo_contract_7, mem_out)
        && (__fibo_contract_7 ? __fibo_contract_9 == 0
                              : (!_arrow_initialization(mem_in.ni_4) ==>
                                 __fibo_contract_9 == mem_in._reg.__fibo_contract_8
                                 ));
*/
```

Stateful contract example

Much more complex contract transition predicate

```
/*@ predicate fibo_contract_tr3(struct fibo_contract_mem_ghost mem_in,
                                 _Bool x, integer a,
                                 integer __fibo_contract_9,
                                 _Bool __fibo_contract_1,
                                 struct fibo_contract_mem_ghost mem_out) =
      fibo_contract_tr2(mem_in, x, a, __fibo_contract_9, mem_out)
      && _arrow_transition(mem_in.ni_5, __fibo_contract_1, mem_out.ni_5);
*/
/*@ predicate fibo_contract_tr4(struct fibo_contract_mem_ghost mem_in,
                                 _Bool x, integer a,
                                 integer __fibo_contract_9,
                                 integer __fibo_contract_6,
                                 struct fibo_contract_mem_ghost mem_out) =
      \exists _Bool __fibo_contract_1;
        fibo_contract_tr3(mem_in,
                          x, a, __fibo_contract_9, __fibo_contract_1,
                          mem_out)
        && (__fibo_contract_1 ? __fibo_contract_6 == 0
                        : (!_arrow_initialization(mem_in.ni_4) ==>
                           __fibo_contract_6 == mem_in._reg.__fibo_contract_5
                           ));
*/
```

Stateful contract example

Much more complex contract transition predicate

```
/*@ predicate fibo_contract_tr5(struct fibo_contract_mem_ghost mem_in,
                                 _Bool x, integer a, _Bool spec19,
                                 struct fibo_contract_mem_ghost mem_out) =
      \exists integer __fibo_contract_9, integer __fibo_contract_6;
        fibo_contract_tr4(mem_in,
                          x, a, __fibo_contract_9, __fibo_contract_6,
                          mem_out)
        && spec19 == (a == (__fibo_contract_9 + __fibo_contract_6));
*/
/*@ predicate fibo_contract_tr6(struct fibo_contract_mem_ghost mem_in,
                                 _Bool x, integer a, _Bool spec19,
                                 struct fibo_contract_mem_ghost mem_out) =
      fibo_contract_tr5(mem_in, x, a, spec19, mem_out)
      && mem_out._reg.__fibo_contract_8 == a;
*/
/*@ predicate fibo_contract_tr7(struct fibo_contract_mem_ghost mem_in,
                                 _Bool x, integer a,
                                 _Bool __fibo_contract_2, _Bool spec19,
                                 struct fibo_contract_mem_ghost mem_out) =
      fibo_contract_tr6(mem_in, x, a, spec19, mem_out)
      && _arrow_transition(mem_in.ni_6, __fibo_contract_2, mem_out.ni_6);
*/
```

Stateful contract example

Much more complex contract transition predicate

```
/*@ predicate fibo_contract_tr8(struct fibo_contract_mem_ghost mem_in,
                                 _Bool x, integer a,
                                 integer __fibo_contract_4,
                                 _Bool spec19,
                                 struct fibo_contract_mem_ghost mem_out) =
      \exists _Bool __fibo_contract_2;
        fibo_contract_tr7(mem_in, x, a, __fibo_contract_2, spec19,
                          mem_out)
        && (__fibo_contract_2 ? __fibo_contract_4 == 1
                        : (!_arrow_initialization(mem_in.ni_4) ==>
                           __fibo_contract_4 == mem_in._reg.__fibo_contract_3
                           ));
*/
/*@ predicate fibo_contract_tr9(struct fibo_contract_mem_ghost mem_in,
                                 _Bool x, integer a, _Bool spec19,
                                 struct fibo_contract_mem_ghost mem_out) =
      \exists integer __fibo_contract_4;
        fibo_contract_tr8(mem_in, x, a, __fibo_contract_4, spec19,
                          mem_out)
        && mem_out._reg.__fibo_contract_5 == __fibo_contract_4;
*/
```

Stateful contract example

Much more complex contract transition predicate

```
/*@ predicate fibo_contract_tr10(struct fibo_contract_mem_ghost mem_in,
                                  _Bool x, integer a, _Bool spec19,
                                  struct fibo_contract_mem_ghost mem_out) =
      fibo_contract_tr9(mem_in, x, a, spec19, mem_out)
      && mem_out._reg.__fibo_contract_3 == a;
*/
/*@ predicate fibo_contract_tr(struct fibo_contract_mem_ghost mem_in,
                               _Bool x, integer a, _Bool spec19,
                               struct fibo_contract_mem_ghost mem_out) =
      \exists struct fibo_contract_mem_ghost mem_reset;
        fibo_contract_reset_cleared(mem_in, mem_reset)
        && fibo_contract_tr10(mem_reset, x, a, spec19, mem_out);
*/
```

Stateful contract example

Same k-induction principles, with state

```
/*@ ghost
      _Bool spec19;
      struct fibo_contract_mem_ghost mem_in_c, mem_out_c;
*/
/*@ ...
    requires fibo_pack(*mem, self);
    ensures fibo_pack(*mem, self);
    ensures fibo_transition(\old(*mem), x, *a, *mem);
    ensures fibo_contract_tr(mem_in_c, x, *a, spec19, mem_out_c) ==>
            spec19;
    ...
*/
void fibo_step(_Bool x, int *a, struct fibo_mem *self)
/*@ ghost (struct fibo_mem_ghost \ghost *mem) */ {
```

Conclusion

Summary

- A certifying compiler from Lustre to C / ACSL
- Certificate of correctness wrt a state / transition semantics
- Automatic translation of high-level contracts

Future

- Prototype
- Other alternative for compiling the *reset*
- Optimizations
- Invariant generation
- Encoding a synchronous dataflow semantics?
- SPARK / Ada

References I

- 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.
- 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.
- Xavier Leroy (July 2009). "Formal Verification of a Realistic Compiler". In: *Communications of the ACM* 52.7, pp. 107–115.
- Andrew W. Appel (2011). "Verified Software Toolchain". In: *Programming Languages and Systems*. Ed. by Gilles Barthe. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, pp. 1–17.

References II

- 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.
- Xavier Thirioux (Sept. 19, 2016). "Verifying Embedded Systems". Habilitation. Institut National Polytechnique de Toulouse. 187 pp.
- 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.

References III

- Brandon Bohrer, Yong Kiam Tan, Stefan Mitsch, Magnus O. Myreen, and André Platzer (June 11, 2018). "VeriPhy: Verified Controller Executables from Verified Cyber-Physical System Models". In: *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI 2018. New York, NY, USA: Association for Computing Machinery, pp. 617–630.