From Lustre to C:
a Translation Validation Approach to Verified Compilation

AID Meeting

Lélio Brun[1]

March 9, 2022

[1]ISAE-SUPAERO – DISC – IpSC

Context

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

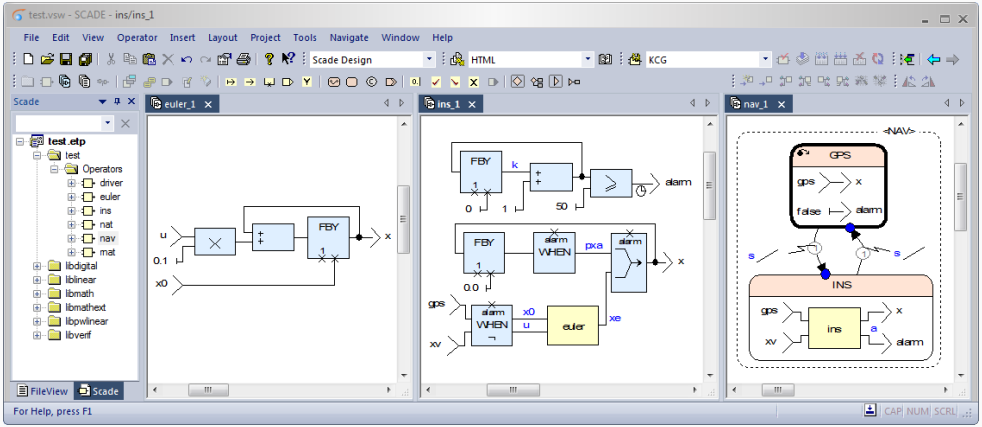
- computer systems within physical systems that interact with the real world, often with real-time constraints
- software usually written in low-level languages: C, Ada, Assembly

Model-Based Design

Executable high-level abstract specifications

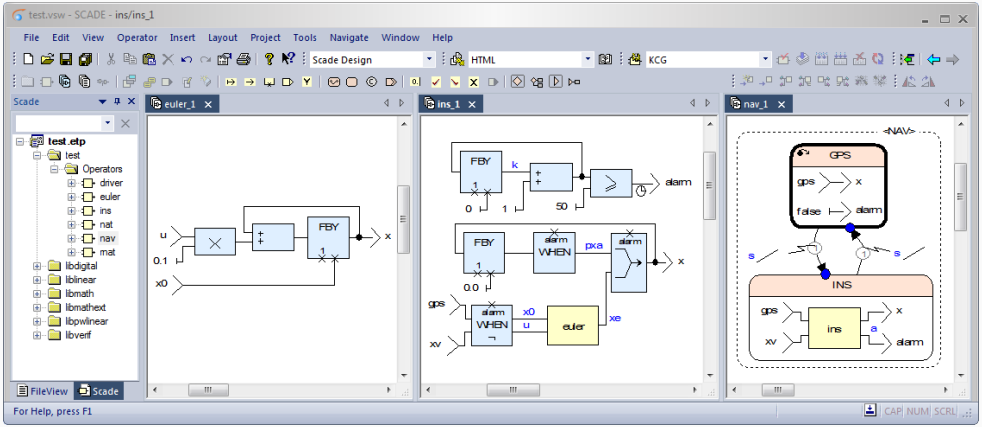
Model-Based Design in SCADE Suite

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



Model-Based Design in SCADE Suite

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

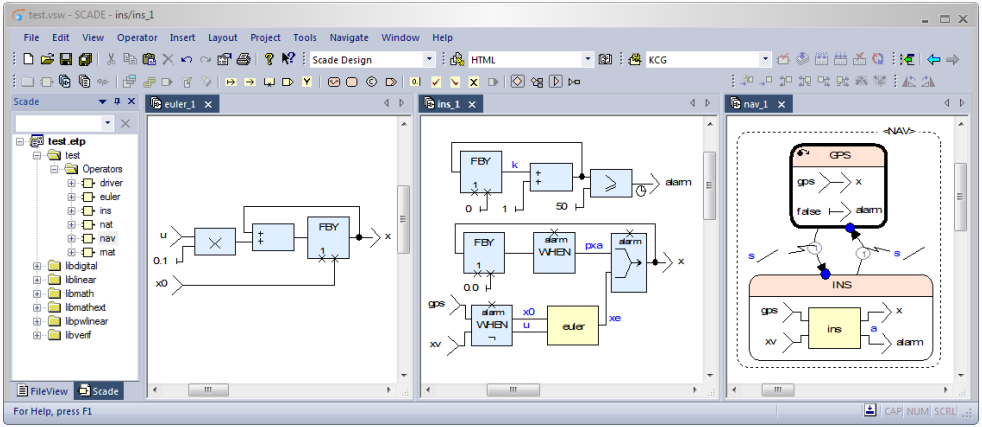


block / node = system

line = signal

Model-Based Design in SCADE Suite

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

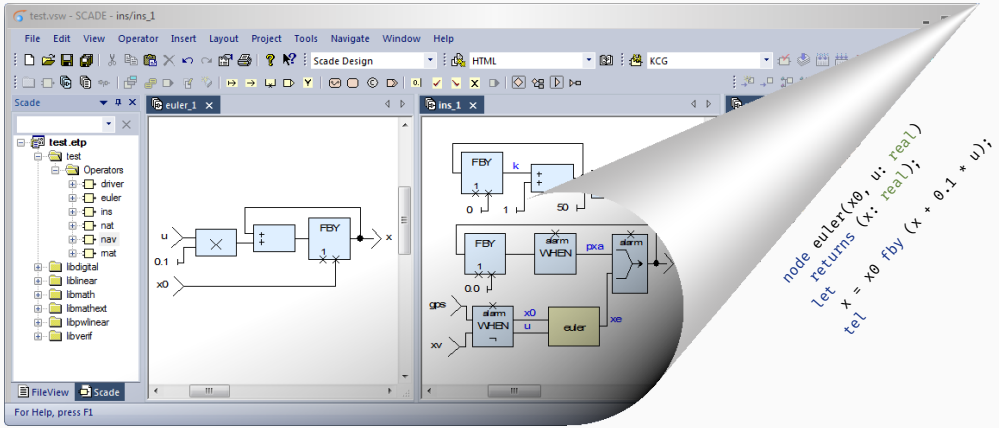


block / node = system = stream function

line = signal = stream of values

Model-Based Design in SCADE Suite

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

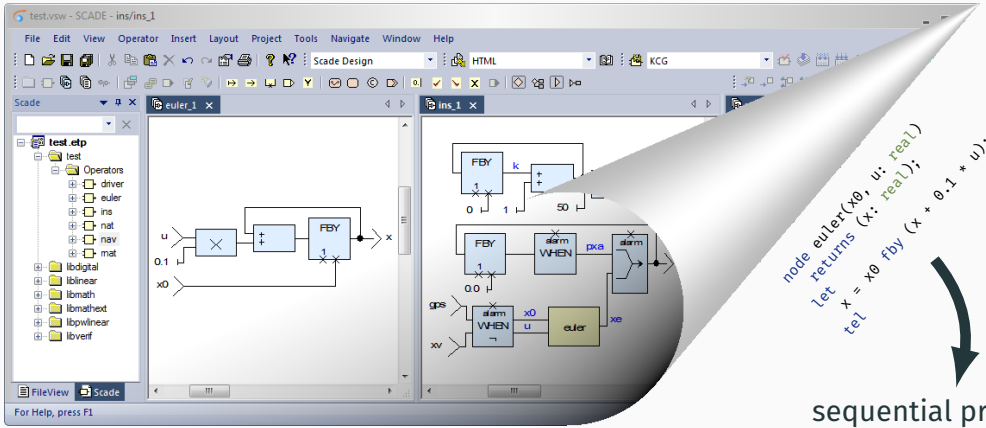


block / node = system = stream function

line = signal = stream of values

Model-Based Design in SCADE Suite

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



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?~~

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

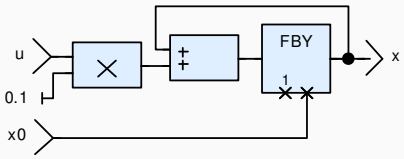
Vélus a verified Lustre compiler in Coq, based on CompCert

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~~

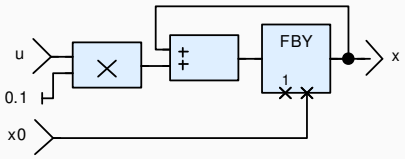
Example




```
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	$\cdots$

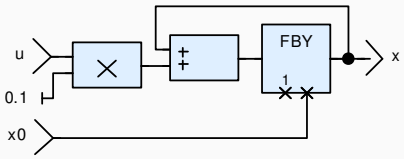
Example




```
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	$\cdots$

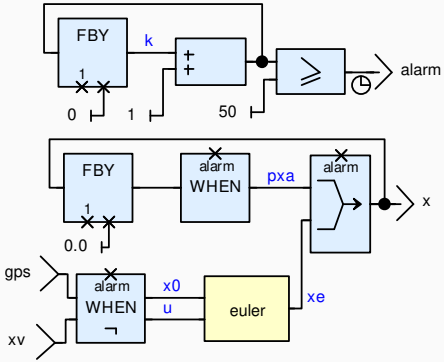
Example




```
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	$\cdots$

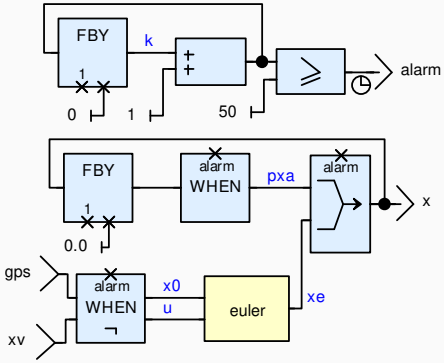
Example




```
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	···

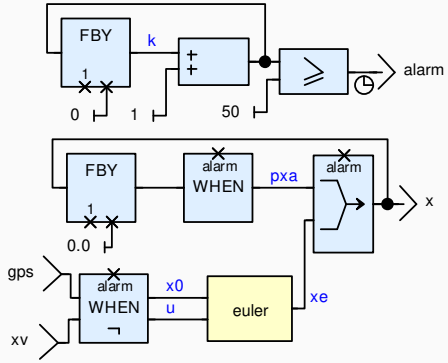
Example




```
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	···

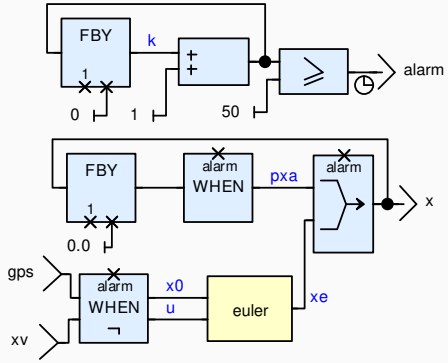
Example




```
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	···

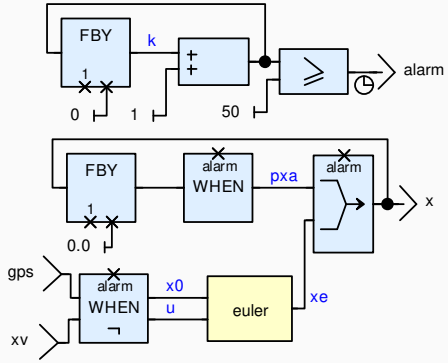
Example




```
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	···

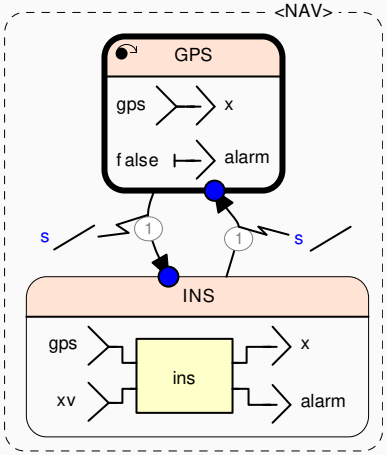
Example




```
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
```

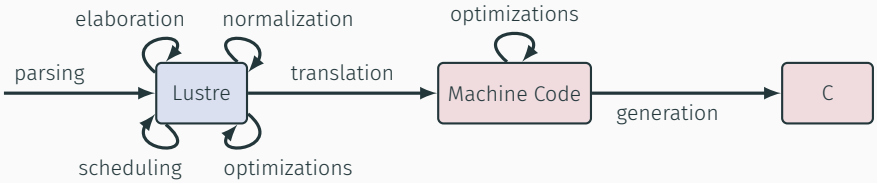
gps	0.00	1.55	3.62	5.46	···	86.52	88.40	90.91	···
xv	15.00	20.00	17.00	12.00	···	18.00	23.00	20.00	···
k	0	1	2	3	···	49	50	51	···
alarm	F	F	F	F	···	F	T	T	···
xe	0.00	1.50	3.50	5.20	···	77.35			···
pxa					···		77.35	77.35	···
x	0.00	1.50	3.50	5.20	···	77.35	77.35	77.35	···

Example



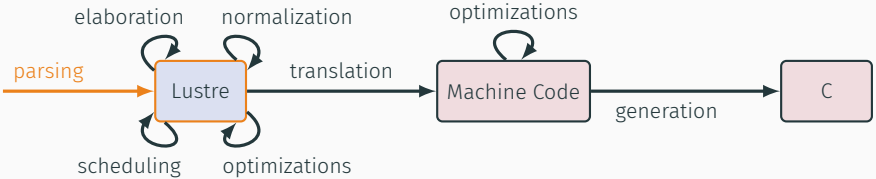

```
node nav(gps, xv: real, s: bool)
  returns (x: real, alarm: bool)
let
  automaton a
    state GPS:
      let
        x = gps;
        alarm = false;
      tel until s restart INS
    state INS:
      let
        (x, alarm) = ins(gps, xv);
      tel until s resume GPS
tel
```

LustreC: a Lustre Compiler



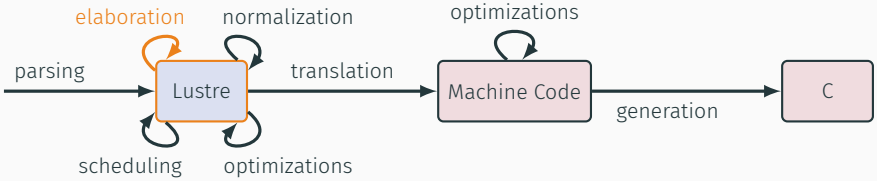
Implemented in OCaml ($\approx 35\,000$ loc)

LustreC: a Lustre Compiler



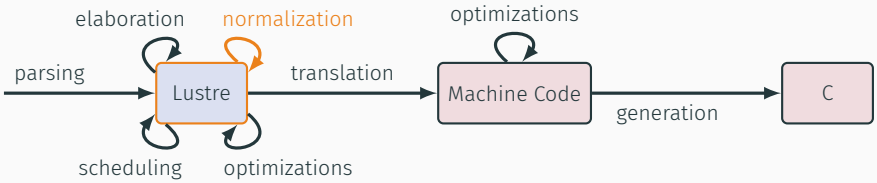
- parsing (`menhir`)

LustreC: a Lustre Compiler



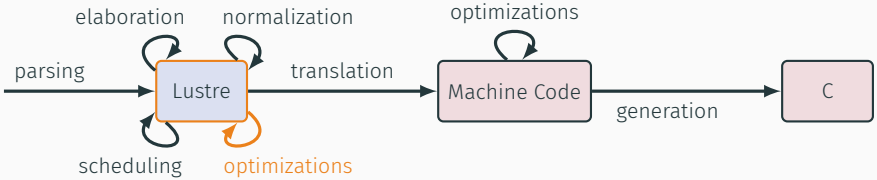
- parsing (`menhir`)
- elaboration to get clock and type information

LustreC: a Lustre Compiler



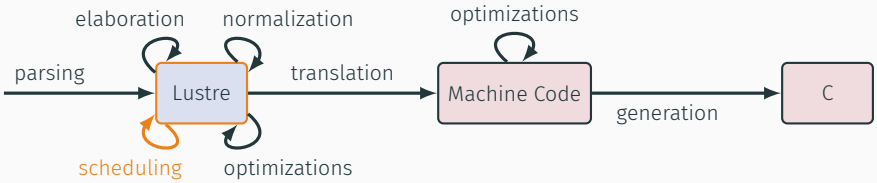
- parsing (`menhir`)
- elaboration to get clock and type information
- normalization of Lustre code

LustreC: a Lustre Compiler



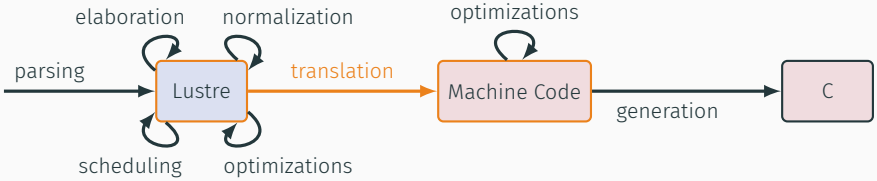
- parsing (`menhir`)
- elaboration to get clock and type information
- normalization of Lustre code
- optimization of Lustre code

LustreC: a Lustre Compiler



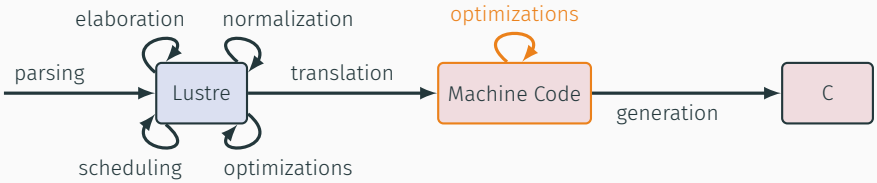
- parsing (`menhir`)
- elaboration to get clock and type information
- normalization of Lustre code
- optimization of Lustre code
- scheduling of Lustre code

LustreC: a Lustre Compiler



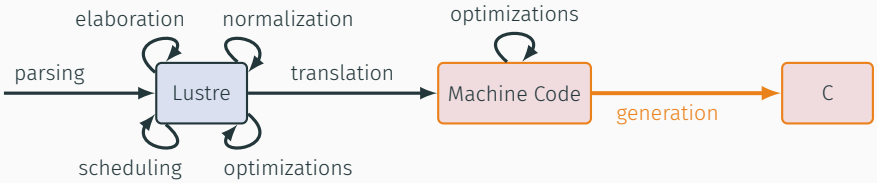
- parsing (`menhir`)
- elaboration to get clock and type information
- normalization of Lustre code
- optimization of Lustre code
- scheduling of Lustre code
- translation to Machine code

LustreC: a Lustre Compiler

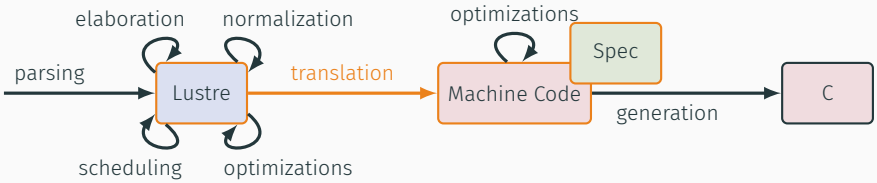


- parsing (`menhir`)
- elaboration to get clock and type information
- normalization of Lustre code
- optimization of Lustre code
- scheduling of Lustre code
- translation to Machine code
- optimisation of Machine code

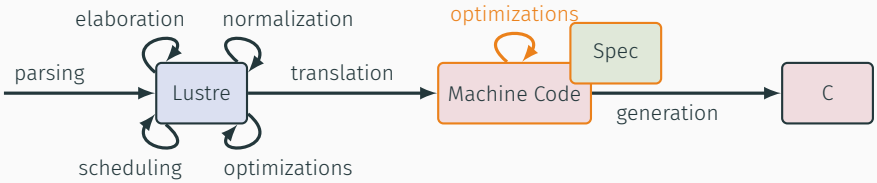
LustreC: a Lustre Compiler



- 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

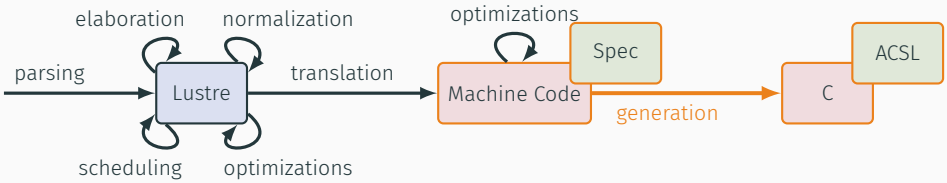


- parsing (`menhir`)
- elaboration to get clock and type information
- normalization of Lustre code
- optimization of Lustre code
- scheduling of Lustre code
- translation to Machine code
- optimisation of Machine code
- generation of C code



- 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

```
#include <assert.h>
#include "acc_automaton.h"

// Struct definitions */
struct euler_mem {
  double x;
  struct euler_reg {double _euler_2;} _reg;
  struct _acrmp_mem ins_1;
};

struct ins_mem {
  _Bool _reset;
  struct ins_reg {double __ins_2;
                  int __ins_1;} _reg;
  struct _acrmp_mem ins_2;
  struct euler_mem ins_3;
};

struct a_INS_handler_until_mem {
  _Bool _reset;
  struct a_INS_handler_until_reg {int __a_INS_handler_until_1;} _reg;
  struct ins_mem ins_18;
  struct _acrmp_mem ins_4;
};

struct nav_mem {
  _Bool _reset;
  struct nav_reg { _Bool __nav_16;
                   double __ins_17;} _reg;
  struct _acrmp_mem ins_5;
  struct a_INS_handler_until_mem ins_19;
};

void euler_clear_reset (struct euler_mem *self) {
  if (self->_reset) {
    self->_reset = 0;
    _acrmp_reset(&self->ins_7);
  }
  return;
}

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

void euler_step (double u0, double u,
                 double *res,
                 struct euler_mem *self) {
  _Bool __euler_1;
  euler_set_reset(self);
  {
    _euler_2 = _acrmp_step(self->ins_7);
    if (self->_reset) {
      x = u0;
    } else {
      x = self->_reg._euler_2;
    }
    self->_reg._euler_2 = (*res = (0.1 * u));
  }
  return;
}

void ins_clear_reset (struct ins_mem *self) {
  if (self->_reset) {
    self->_reset = 0;
    _acrmp_reset(&self->ins_8);
    euler_set_reset(self->ins_9);
  }
  return;
}

void ins_set_reset (struct ins_mem *self) {
  self->_reset = 1;
  return;
}

void ins_step (double gps, double xv,
               double *res, _Bool *relarm_6),
               struct ins_mem *self) {
  int k;
  _Bool alarm;
  double __ins_2;
  _Bool __ins_1;
  double pxa;
  double xe;
  ins_clear_reset(self);
  {
    __ins_1 = _acrmp_step(self->ins_8);
    if (__ins_1) {
      k = 0;
    } else {
      k = self->_reg.__ins_2;
    }
    alarm = (k >= 50);
    if (!alarm) {
      euler_step(gps, xv, &xe, self->ins_9);
    }
    if (__ins_1) {
      __ins_2 = 0.0;
    } else {
      __ins_2 = self->_reg.__ins_2;
    }
    if (alarm) {
      pxa = __ins_2;
      *relarm_6 = 1;
    } else {
      pxa = 0.;
      *relarm_6 = 0;
    }
    self->_reg.__ins_2 = (k + 1);
    self->_reg.__ins_1 = 1;
    return;
  }
}

void a_GPS_handler_until_step (_Bool a_restart_ent, double gps, _Bool a,
                               _Bool (*a_restart_int),
                               a_type (*a_state_int), _Bool (*relarm_out),
                               int (*k_out), double (*xe_out)) {
  *a_out = gps;
  *k_out = 0;
  *xe_out = 0.;
  if (a) {
    *a_restart_int = 1;
    *a_state_int = GPS;
    *a_restart_int = 0;
  } else {
  }
  return;
}

void a_GPS_unless_step (_Bool a_restart_in,
                        _Bool (*a_restart_ent), a_type (*a_state_ent)) {
  *a_state_ent = GPS;
  *a_restart_ent = a_restart_in;
  return;
}

void a_INS_handler_until_clear_reset (struct a_INS_handler_until_mem *self) {
  if (self->_reset) {
    self->_reset = 0;
    ins_set_reset(self->ins_18);
    _acrmp_reset(&self->ins_19);
  }
  return;
}

void a_INS_handler_until_set_reset (struct a_INS_handler_until_mem *self) {
  self->_reset = 1;
  return;
}

void a_INS_handler_until_step (_Bool a_restart_ent, double gps, double xv,
                               _Bool a,
                               _Bool (*a_restart_int),
                               a_type (*a_state_int), _Bool (*relarm_out),
                               int (*k_out), double (*xe_out),
                               struct a_INS_handler_until_mem *self) {

  double xe;
  int k;
  _Bool relarm;
  a_INS_handler_until_2();
  a_INS_handler_until_clear_reset(self);
  {
    ins_step(gps, xv, &xe, &relarm, self->ins_18);
    *k_out = k;
    if (a_INS_handler_until_1 = _acrmp_step(self->ins_19)) {
      if (_a_INS_handler_until_2) {
        k = 0;
      } else {
        k = self->_reg._a_INS_handler_until_1;
      }
    }
    *k_out = k;
    relarm_out = alarm;
    if (alarm) {
      *a_restart_int = 1;
      *a_state_int = GPS;
      *a_restart_int = 0;
    } else {
      *xe_out = xe;
    }
    self->_reg._a_INS_handler_until_2 = (k < 1);
    return;
  }
}

void a_INS_unless_step (_Bool a_restart_in,
                        _Bool (*a_restart_ent), a_type (*a_state_ent)) {
  *a_state_ent = INS;
  *a_restart_ent = a_restart_in;
  return;
}

void nav_clear_reset (struct nav_mem *self) {
  if (self->_reset) {
    self->_reset = 0;
    _acrmp_reset(&self->ins_12);
    a_INS_handler_until_set_reset(&self->ins_13);
  }
  return;
}

void nav_set_reset (struct nav_mem *self) {
  self->_reset = 1;
  return;
}

void nav_step (double gps, double xv, _Bool s,
               double (*x), int (*ok), _Bool (*relarm),
               struct nav_mem *self) {
  _Bool a_next_restart_in;
  _Bool a_restart_ent;
  _Bool a_restart_int;
  a_type a_next_state_in;
  a_type a_state_ent;
  a_type a_state_int;
  _Bool __nav_15;
  double __nav_9;
  int __nav_10;
  _Bool __nav_11;
  double __nav_12;
  _Bool __nav_13;
  _Bool __nav_14;
  a_type __nav_7;
  a_type __nav_8;
  nav_clear_reset(self);
  {
    __nav_10 = _acrmp_step(self->ins_12);
    if (self->_reset) {
      __nav_9 = 0.0;
      __nav_11 = 1;
      __nav_12 = 0.;
      __nav_13 = 1;
      __nav_14 = 0;
      __nav_7 = INS;
      __nav_8 = INS;
    } else {
      __nav_9 = self->_reg.__nav_12;
      __nav_11 = self->_reg.__nav_13;
      __nav_12 = self->_reg.__nav_14;
      __nav_13 = self->_reg.__nav_15;
      __nav_14 = self->_reg.__nav_16;
      __nav_7 = self->_reg.__nav_17;
      __nav_8 = self->_reg.__nav_18;
    }
    switch(a_state_int) {
      case GPS:
        a_GPS_unless_step(s, a_restart_in, &__nav_11, &__nav_13);
        a_state_ent = __nav_7;
        a_restart_ent = __nav_11;
        break;
      default: // INS
        a_INS_unless_step(s, a_restart_in, &__nav_11, &__nav_13);
        a_state_ent = __nav_8;
        a_restart_ent = __nav_11;
    }
    switch(a_state_int) {
      case GPS:
        a_GPS_handler_until_step(a_restart_ent, gps, x, &__nav_10, &__nav_12,
                                 &__nav_13, &__nav_14,
                                 &__nav_15);
        *x = __nav_9;
        *ok = __nav_10;
        *relarm = __nav_12;
        a_next_restart_in = __nav_13;
        a_next_restart_int = __nav_14;
        break;
      default: // INS
        if (a_restart_ent) {
          a_INS_handler_until_set_reset(self->ins_13);
        }
        a_INS_handler_until_step(a_restart_ent, gps, xv, x, &__nav_9,
                                 &__nav_10,
                                 &__nav_11,
                                 &__nav_12,
                                 &__nav_13,
                                 &__nav_14,
                                 &__nav_15,
                                 &__nav_16,
                                 &__nav_17,
                                 &__nav_18);
        *x = __nav_9;
        *ok = __nav_10;
        *relarm = __nav_12;
        a_next_state_in = __nav_8;
        a_next_restart_in = __nav_13;
        a_next_restart_int = __nav_14;
    }
    self->_reg.__nav_12 = a_next_state_in;
    self->_reg.__nav_13 = a_next_restart_in;
    self->_reg.__nav_14 = a_next_restart_int;
    return;
  }
}
```

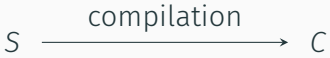
+ separated code for the main loop

Lustre

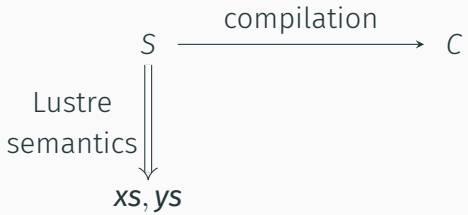
```
node euler(x0, u: real)
  returns (x: real);
let
  x = x0 fby (x + 0.1 * u);
tel

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

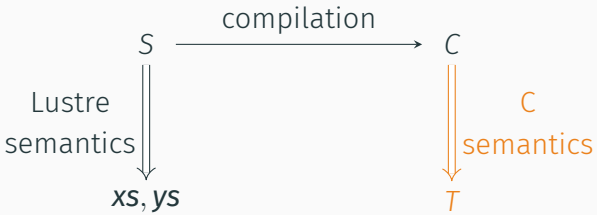
Correctness?



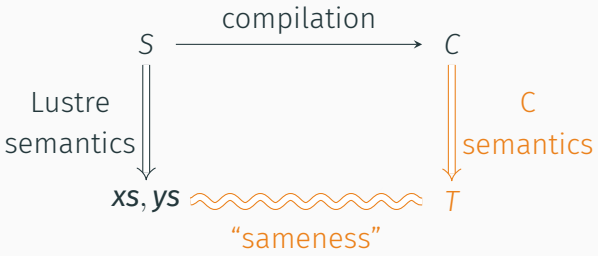
Correctness?



Correctness?



Correctness?



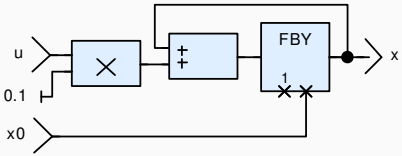
Correctness?

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 y: real; c: bool;
let
  x = if c then x0 else y;
  y = pre (x + 0.1 * u);
  c = true -> false;
tel
```

```
a fby b ≡ a -> pre (b)
        ≡ if (true -> false) then a else pre (b)
```

Example: Scheduling

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

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

Example: Machine code generation

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

```
machine euler {
  mem y: real;
  instance a: arrow<>;

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

Example: Machine code generation

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

```
machine euler {
  mem y: real;
  instance a: arrow<>;

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

Example: Machine code generation

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

```
machine euler {
  mem y: real;
  instance a: arrow<>;

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

Example: C code generation

```
machine euler {
  mem y: real;
  instance a: arrow<>;

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

```
struct _arrow_mem {
  _Bool _first;
};

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

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

Example: C code generation

```
machine euler {
  mem y: real;
  instance a: arrow<>;

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

```
struct euler_mem {
  _Bool _reset;
  double y;
  struct _arrow_mem *a;
};
```

Example: C code generation

```
machine euler {
  mem y: real;
  instance a: arrow<>;

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

```
struct euler_mem {
  _Bool _reset;
  double y;
  struct _arrow_mem *a;
};
```

Example: C code generation

```
machine euler {
  mem y: real;
  instance a: arrow<>;

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

```
struct euler_mem {
  _Bool _reset;
  double y;
  struct _arrow_mem *a;
};
```

Example: C code generation

```
machine euler {
  mem y: real;
  instance a: arrow<>;

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

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

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

Example: C code generation

```
machine euler {
  mem y: real;
  instance a: arrow<>;

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

```
void euler_step(double x0, double u,
                double *x,
                struct euler_mem *self) {
  _Bool c;
  euler_clear_reset(self);
  c = _arrow_step(self->a);
  if (c) {
    *x = x0;
  } else {
    *x = self->y;
  }
  self->y = *x + 0.1 * u;
  return;
}
```

Encoding of the Semantics

State / Transition Semantics

$S[i]$ sub-instance

$S(x)$ state variabe

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

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

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

State / Transition Semantics

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

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

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 {
  _Bool _first;
};

struct euler_mem {
  _Bool _reset;
  double y;
  struct _arrow_mem *a;
};
```

```
/*@ ghost struct _arrow_mem_ghost {
            _Bool _first;
    };
*/

/*@ ghost struct euler_mem_ghost {
            _Bool _reset;
            double y;
            struct _arrow_mem_ghost a;
    };
*/
```

ACSL generation: memory correspondence

```
/*@ predicate euler_pack_base(struct euler_mem_ghost mem, struct euler_mem *self) =
      mem._reset == self->_reset && mem._reset == 0
      && _arrow_pack(mem.a, self->a);

    predicate euler_pack0(struct euler_mem_ghost mem_reset,
                          struct euler_mem_ghost mem, struct euler_mem *self) =
      euler_pack_base(mem, self)
      && !_arrow_initialization(mem.a) ==> mem.y == self->y;

    predicate euler_pack1(struct euler_mem_ghost mem_reset,
                          struct euler_mem_ghost mem, struct euler_mem *self) =
      euler_pack_base(mem, self)
      && !_arrow_initialization(mem_reset.a) ==> mem.y == self->y;

    predicate euler_pack(struct euler_mem_ghost mem, struct euler_mem *self) =
      mem._reset ? mem._reset == self->_reset : euler_pack0(mem, mem, self);
*/
```

ACSL generation: transition relation

$$
\begin{aligned}
&\mathit{euler\_init}\,(S) = \\
&\quad \mathit{arrow\_init}\,(S[a])
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{euler\_reset}\,(S, S') = \\
&\quad S(\_reset) \rightarrow S'(\_reset) = \mathit{false} \\
&\qquad\qquad \wedge\, \mathit{euler\_init}(S') \\
&\quad \wedge\, \neg S(\_reset) \rightarrow S' = S
\end{aligned}
$$

```
/*@ predicate euler_init(struct euler_mem_ghost mem) =
      _arrow_init(mem.a);
*/

/*@ 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

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

```
/*@ 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

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

```
/*@ 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.y == x + 0.1 * u;
*/
```

ACSL generation: transition relation

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

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

ACSL generation: transition relation

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

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

ACSL generation: transition relation

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

```
/*@ 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->a);
    requires euler_pack(*mem, self);
    ensures euler_pack0(*mem, *mem, self);
    ensures euler_reset(\old(*mem), *mem);
    assigns self->_reset, self->a->_first, mem->_reset, mem->a._first;
*/
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->a);
    //@ ghost _arrow_reset_ghost(mem->a);
  }
  return;
}
```

ACSL generation: contracts and assertions

```
/*@ requires euler_valid(self);
    requires \separated(self, mem, self->a);
    requires euler_pack(*mem, self);
    ensures euler_pack0(*mem, *mem, self);
    ensures euler_reset(\old(*mem), *mem);
    assigns self->_reset, self->a->_first, mem->_reset, mem->a._first;
*/
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->a);
    //@ ghost _arrow_reset_ghost(mem->a);
  }
  return;
}
```

ACSL generation: contracts and assertions

```
/*@ requires euler_valid(self);
    requires \separated(self, mem, self->a);
    requires euler_pack(*mem, self);
    ensures euler_pack0(*mem, *mem, self);
    ensures euler_reset(\old(*mem), *mem);
    assigns self->_reset, self->a->_first, mem->_reset, mem->a._first;
*/
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->a);
    //@ ghost _arrow_reset_ghost(mem->a);
  }
  return;
}
```

ACSL generation: contracts and assertions

```
/*@ requires euler_valid(self);
    requires \separated(self, mem, self->a);
    requires euler_pack(*mem, self);
    ensures euler_pack0(*mem, *mem, self);
    ensures euler_reset(\old(*mem), *mem);
    assigns self->_reset, self->a->_first, mem->_reset, mem->a._first;
*/
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->a);
    //@ ghost _arrow_reset_ghost(mem->a);
  }
  return;
}
```

ACSL generation: contracts and assertions

```
/*@ requires euler_valid(self);
    requires \separated(self, mem, self->a);
    requires euler_pack(*mem, self);
    ensures euler_pack0(*mem, *mem, self);
    ensures euler_reset(\old(*mem), *mem);
    assigns self->_reset, self->a->_first, mem->_reset, mem->a._first;
*/
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->a);
    //@ ghost _arrow_reset_ghost(mem->a);
  }
  return;
}
```

ACSL generation: contracts and assertions

```
/*@ requires euler_valid(self);
    requires \separated(self, mem, self->a);
    requires euler_pack(*mem, self);
    ensures euler_pack0(*mem, *mem, self);
    ensures euler_reset(\old(*mem), *mem);
    assigns self->_reset, self->a->_first, mem->_reset, mem->a._first;
*/
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->a);
    //@ ghost _arrow_reset_ghost(mem->a);
  }
  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->a, x);
    requires euler_pack(*mem, self);
    ensures euler_pack(*mem, self);
    ensures euler_tr(\old(*mem), x0, u, *x, *mem);
    assigns *x, self->y, self->_reset, self->a->_first,
            mem->y, mem->_reset, mem->a._first;
*/
void euler_step(double x0,
                double u, double *x,
                struct euler_mem *self)
     /*@ ghost (struct euler_mem_ghost \ghost *mem) */ {
```

ACSL generation: contracts and assertions

```
/*@ requires \valid(x);
    requires euler_valid(self);
    requires \separated(self, mem, self->a, x);
    requires euler_pack(*mem, self);
    ensures euler_pack(*mem, self);
    ensures euler_tr(\old(*mem), x0, u, *x, *mem);
    assigns *x, self->y, self->_reset, self->a->_first,
            mem->y, mem->_reset, mem->a._first;
*/
void euler_step(double x0,
                double u, double *x,
                struct euler_mem *self)
 /*@ ghost (struct euler_mem_ghost \ghost *mem) */ {
```

ACSL generation: contracts and assertions

```
/*@ requires \valid(x);
    requires euler_valid(self);
    requires \separated(self, mem, self->a, x);
    requires euler_pack(*mem, self);
    ensures euler_pack(*mem, self);
    ensures euler_tr(\old(*mem), x0, u, *x, *mem);
    assigns *x, self->y, self->_reset, self->a->_first,
            mem->y, mem->_reset, mem->a._first;
*/
void euler_step(double x0,
                double u, double *x,
                struct euler_mem *self)
 /*@ ghost (struct euler_mem_ghost \ghost *mem) */ {
```

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 c;
  euler_clear_reset(self)/*@ ghost (mem) */;
  /*@ ghost struct euler_mem_ghost mem_reset = *mem; */
  /*@ assert euler_pack0(mem_reset, *mem, self); */
  /*@ assert euler_tr0(mem_reset, x0, u, *mem); */
  c = _arrow_step(self->a)/*@ ghost (&mem->a) */;
  /*@ assert euler_pack1(mem_reset, *mem, self); */
  /*@ assert euler_tr1(mem_reset, x0, u, c, *mem); */
  if (c) { *x = x0; } else { *x = self->y; }
  /*@ assert euler_tr2(mem_reset, x0, u, *x, *mem); */
  self->y = *x + 0.1 * u;
  //@ ghost mem->y = *x + 0.1 * u;
  /*@ assert euler_pack0(mem_reset, *mem, self); */
  /*@ assert euler_tr3(mem_reset, x0, u, *x, *mem); */
  return;
}
```

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 c;
  euler_clear_reset(self)/*@ ghost (mem) */;
  /*@ ghost struct euler_mem_ghost mem_reset = *mem; */
  /*@ assert euler_pack0(mem_reset, *mem, self); */
  /*@ assert euler_tr0(mem_reset, x0, u, *mem); */
  c = _arrow_step(self->a)/*@ ghost (&mem->a) */;
  /*@ assert euler_pack1(mem_reset, *mem, self); */
  /*@ assert euler_tr1(mem_reset, x0, u, c, *mem); */
  if (c) { *x = x0; } else { *x = self->y; }
  /*@ assert euler_tr2(mem_reset, x0, u, *x, *mem); */
  self->y = *x + 0.1 * u;
  //@ ghost mem->y = *x + 0.1 * u;
  /*@ assert euler_pack0(mem_reset, *mem, self); */
  /*@ assert euler_tr3(mem_reset, x0, u, *x, *mem); */
  return;
}
```

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 c;
  euler_clear_reset(self)/*@ ghost (mem) */;
  /*@ ghost struct euler_mem_ghost mem_reset = *mem; */
  /*@ assert euler_pack0(mem_reset, *mem, self); */
  /*@ assert euler_tr0(mem_reset, x0, u, *mem); */
  c = _arrow_step(self->a)/*@ ghost (&mem->a) */;
  /*@ assert euler_pack1(mem_reset, *mem, self); */
  /*@ assert euler_tr1(mem_reset, x0, u, c, *mem); */
  if (c) { *x = x0; } else { *x = self->y; }
  /*@ assert euler_tr2(mem_reset, x0, u, *x, *mem); */
  self->y = *x + 0.1 * u;
  //@ ghost mem->y = *x + 0.1 * u;
  /*@ assert euler_pack0(mem_reset, *mem, self); */
  /*@ assert euler_tr3(mem_reset, x0, u, *x, *mem); */
  return;
}
```

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 c;
  euler_clear_reset(self)/*@ ghost (mem) */;
  /*@ ghost struct euler_mem_ghost mem_reset = *mem; */
  /*@ assert euler_pack0(mem_reset, *mem, self); */
  /*@ assert euler_tr0(mem_reset, x0, u, *mem); */
  c = _arrow_step(self->a)/*@ ghost (&mem->a) */;
  /*@ assert euler_pack1(mem_reset, *mem, self); */
  /*@ assert euler_tr1(mem_reset, x0, u, c, *mem); */
  if (c) { *x = x0; } else { *x = self->y; }
  /*@ assert euler_tr2(mem_reset, x0, u, *x, *mem); */
  self->y = *x + 0.1 * u;
  //@ ghost mem->y = *x + 0.1 * u;
  /*@ assert euler_pack0(mem_reset, *mem, self); */
  /*@ assert euler_tr3(mem_reset, x0, u, *x, *mem); */
  return;
}
```

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 c;
  euler_clear_reset(self)/*@ ghost (mem) */;
  /*@ ghost struct euler_mem_ghost mem_reset = *mem; */
  /*@ assert euler_pack0(mem_reset, *mem, self); */
  /*@ assert euler_tr0(mem_reset, x0, u, *mem); */
  c = _arrow_step(self->a)/*@ ghost (&mem->a) */;
  /*@ assert euler_pack1(mem_reset, *mem, self); */
  /*@ assert euler_tr1(mem_reset, x0, u, c, *mem); */
  if (c) { *x = x0; } else { *x = self->y; }
  /*@ assert euler_tr2(mem_reset, x0, u, *x, *mem); */
  self->y = *x + 0.1 * u;
  //@ ghost mem->y = *x + 0.1 * u;
  /*@ assert euler_pack0(mem_reset, *mem, self); */
  /*@ assert euler_tr3(mem_reset, x0, u, *x, *mem); */
  return;
}
```

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 c;
  euler_clear_reset(self)/*@ ghost (mem) */;
  /*@ ghost struct euler_mem_ghost mem_reset = *mem; */
  /*@ assert euler_pack0(mem_reset, *mem, self); */
  /*@ assert euler_tr0(mem_reset, x0, u, *mem); */
  c = _arrow_step(self->a)/*@ ghost (&mem->a) */;
  /*@ assert euler_pack1(mem_reset, *mem, self); */
  /*@ assert euler_tr1(mem_reset, x0, u, c, *mem); */
  if (c) { *x = x0; } else { *x = self->y; }
  /*@ assert euler_tr2(mem_reset, x0, u, *x, *mem); */
  self->y = *x + 0.1 * u;
  //@ ghost mem->y = *x + 0.1 * u;
  /*@ assert euler_pack0(mem_reset, *mem, self); */
  /*@ assert euler_tr3(mem_reset, x0, u, *x, *mem); */
  return;
}
```

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 c;
  euler_clear_reset(self)/*@ ghost (mem) */;
  /*@ ghost struct euler_mem_ghost mem_reset = *mem; */
  /*@ assert euler_pack0(mem_reset, *mem, self); */
  /*@ assert euler_tr0(mem_reset, x0, u, *mem); */
  c = _arrow_step(self->a)/*@ ghost (&mem->a) */;
  /*@ assert euler_pack1(mem_reset, *mem, self); */
  /*@ assert euler_tr1(mem_reset, x0, u, c, *mem); */
  if (c) { *x = x0; } else { *x = self->y; }
  /*@ assert euler_tr2(mem_reset, x0, u, *x, *mem); */
  self->y = *x + 0.1 * u;
  //@ ghost mem->y = *x + 0.1 * u;
  /*@ assert euler_pack0(mem_reset, *mem, self); */
  /*@ assert euler_tr3(mem_reset, x0, u, *x, *mem); */
  return;
}
```

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 c;
  euler_clear_reset(self)/*@ ghost (mem) */;
  /*@ ghost struct euler_mem_ghost mem_reset = *mem; */
  /*@ assert euler_pack0(mem_reset, *mem, self); */
  /*@ assert euler_tr0(mem_reset, x0, u, *mem); */
  c = _arrow_step(self->a)/*@ ghost (&mem->a) */;
  /*@ assert euler_pack1(mem_reset, *mem, self); */
  /*@ assert euler_tr1(mem_reset, x0, u, c, *mem); */
  if (c) { *x = x0; } else { *x = self->y; }
  /*@ assert euler_tr2(mem_reset, x0, u, *x, *mem); */
  self->y = *x + 0.1 * u;
  //@ ghost mem->y = *x + 0.1 * u;
  /*@ assert euler_pack0(mem_reset, *mem, self); */
  /*@ assert euler_tr3(mem_reset, x0, u, *x, *mem); */
  return;
}
```

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 c;
  euler_clear_reset(self)/*@ ghost (mem) */;
  /*@ ghost struct euler_mem_ghost mem_reset = *mem; */
  /*@ assert euler_pack0(mem_reset, *mem, self); */
  /*@ assert euler_tr0(mem_reset, x0, u, *mem); */
  c = _arrow_step(self->a)/*@ ghost (&mem->a) */;
  /*@ assert euler_pack1(mem_reset, *mem, self); */
  /*@ assert euler_tr1(mem_reset, x0, u, c, *mem); */
  if (c) { *x = x0; } else { *x = self->y; }
  /*@ assert euler_tr2(mem_reset, x0, u, *x, *mem); */
  self->y = *x + 0.1 * u;
  //@ ghost mem->y = *x + 0.1 * u;
  /*@ assert euler_pack0(mem_reset, *mem, self); */
  /*@ assert euler_tr3(mem_reset, x0, u, *x, *mem); */
  return;
}
```

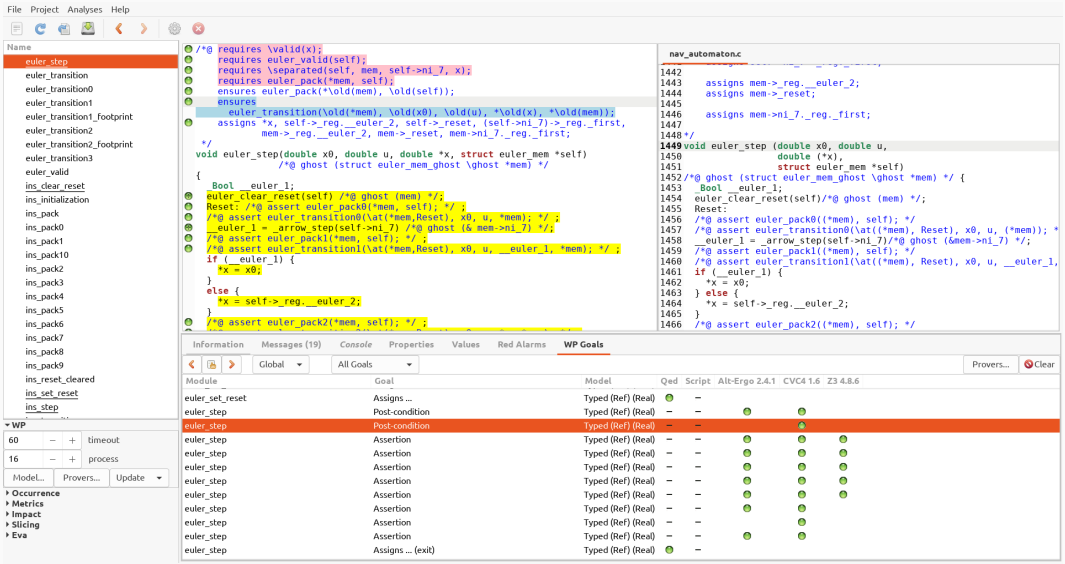
ACSL generation: contracts and assertions

```
void euler_step(double x0,
                double u, double *x,
                struct euler_mem *self)
    /*@ ghost (struct euler_mem_ghost \ghost *mem) */ {
  _Bool c;
  euler_clear_reset(self)/*@ ghost (mem) */;
  /*@ ghost struct euler_mem_ghost mem_reset = *mem; */
  /*@ assert euler_pack0(mem_reset, *mem, self); */
  /*@ assert euler_tr0(mem_reset, x0, u, *mem); */
  c = _arrow_step(self->a)/*@ ghost (&mem->a) */;
  /*@ assert euler_pack1(mem_reset, *mem, self); */
  /*@ assert euler_tr1(mem_reset, x0, u, c, *mem); */
  if (c) { *x = x0; } else { *x = self->y; }
  /*@ assert euler_tr2(mem_reset, x0, u, *x, *mem); */
  self->y = *x + 0.1 * u;
  //@ ghost mem->y = *x + 0.1 * u;
  /*@ assert euler_pack0(mem_reset, *mem, self); */
  /*@ assert euler_tr3(mem_reset, x0, u, *x, *mem); */
  return;
}
```

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 c;
  euler_clear_reset(self)/*@ ghost (mem) */;
  /*@ ghost struct euler_mem_ghost mem_reset = *mem; */
  /*@ assert euler_pack0(mem_reset, *mem, self); */
  /*@ assert euler_tr0(mem_reset, x0, u, *mem); */
  c = _arrow_step(self->a)/*@ ghost (&mem->a) */;
  /*@ assert euler_pack1(mem_reset, *mem, self); */
  /*@ assert euler_tr1(mem_reset, x0, u, c, *mem); */
  if (c) { *x = x0; } else { *x = self->y; }
  /*@ assert euler_tr2(mem_reset, x0, u, *x, *mem); */
  self->y = *x + 0.1 * u;
  //@ ghost mem->y = *x + 0.1 * u;
  /*@ assert euler_pack0(mem_reset, *mem, self); */
  /*@ assert euler_tr3(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->n1_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->n1_7)->_reg._first,
mem->_reg.__euler_2, mem->_reset, mem->n1_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 assigns mem->_reg.__euler_2;
1443 assigns mem->_reset;
1444
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 Assigns ... (exit) Typed (Ref) (Real)

Preliminary results

Test bench: $\sim$ 500 Lustre files

- $\sim$ 70% success (per file)
- $\sim$ 99% success (per proof obligation)

Test bench: $\sim$ 500 Lustre files

- $\sim$ 70% success (per file)
- $\sim$ 99% success (per proof obligation)

Custom proof tactics as an OCaml plugin

Conclusion

Summary

- A certifying compiler from Lustre to C / ACSL
- Certificate of correctness wrt a state / transition semantics

Currently

- Promising results
- Automatic translation of high-level contracts

Future

- 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.