Equation-Directed Axiomatization of Lustre Semantics to Enable Optimized Code Validation

EMSOFT

September 18 2023

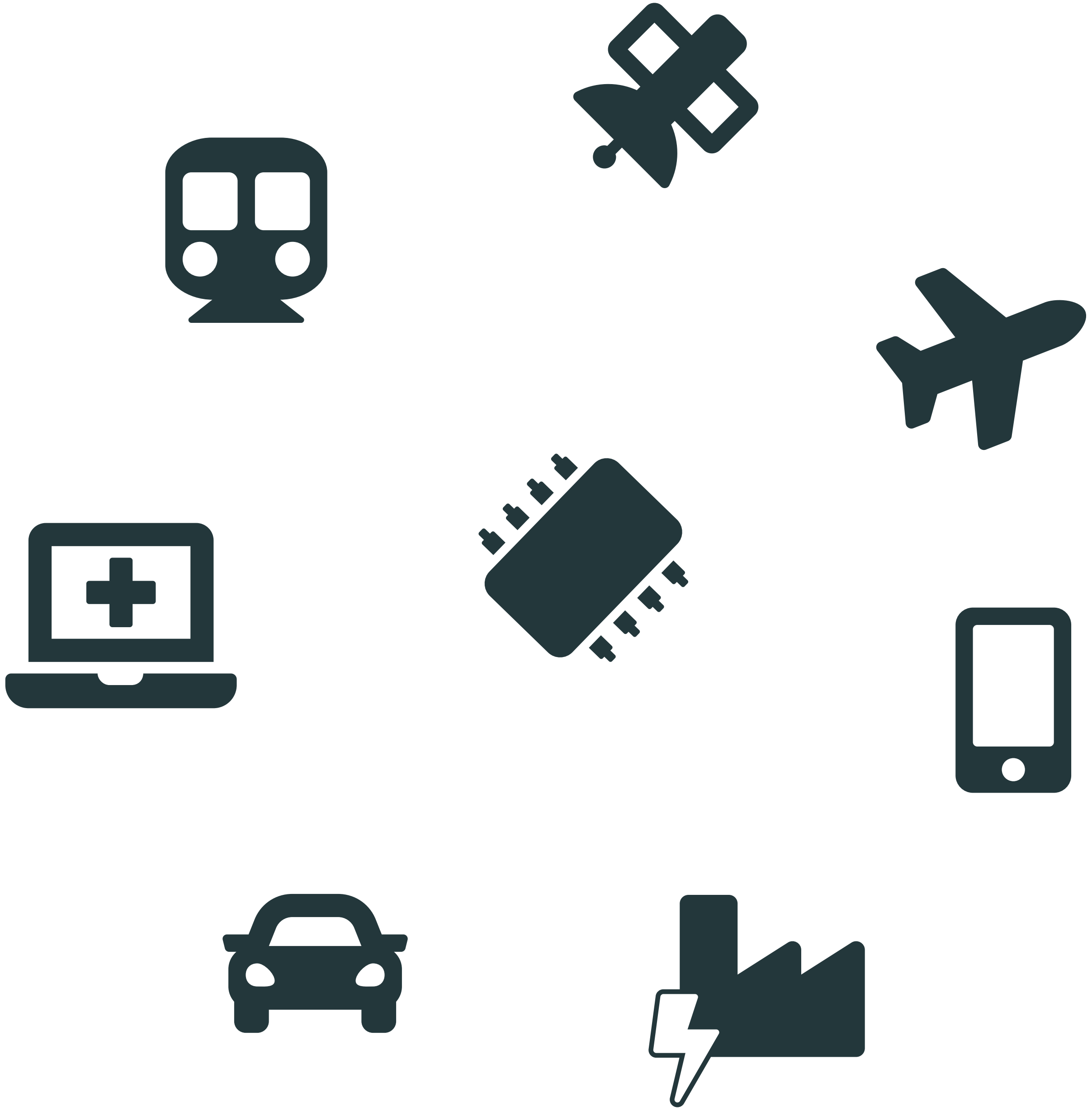
Lélio Brun • Christophe Garion • Pierre-Loïc Garoche • Xavier Thirioux

NII National Institute of Informatics

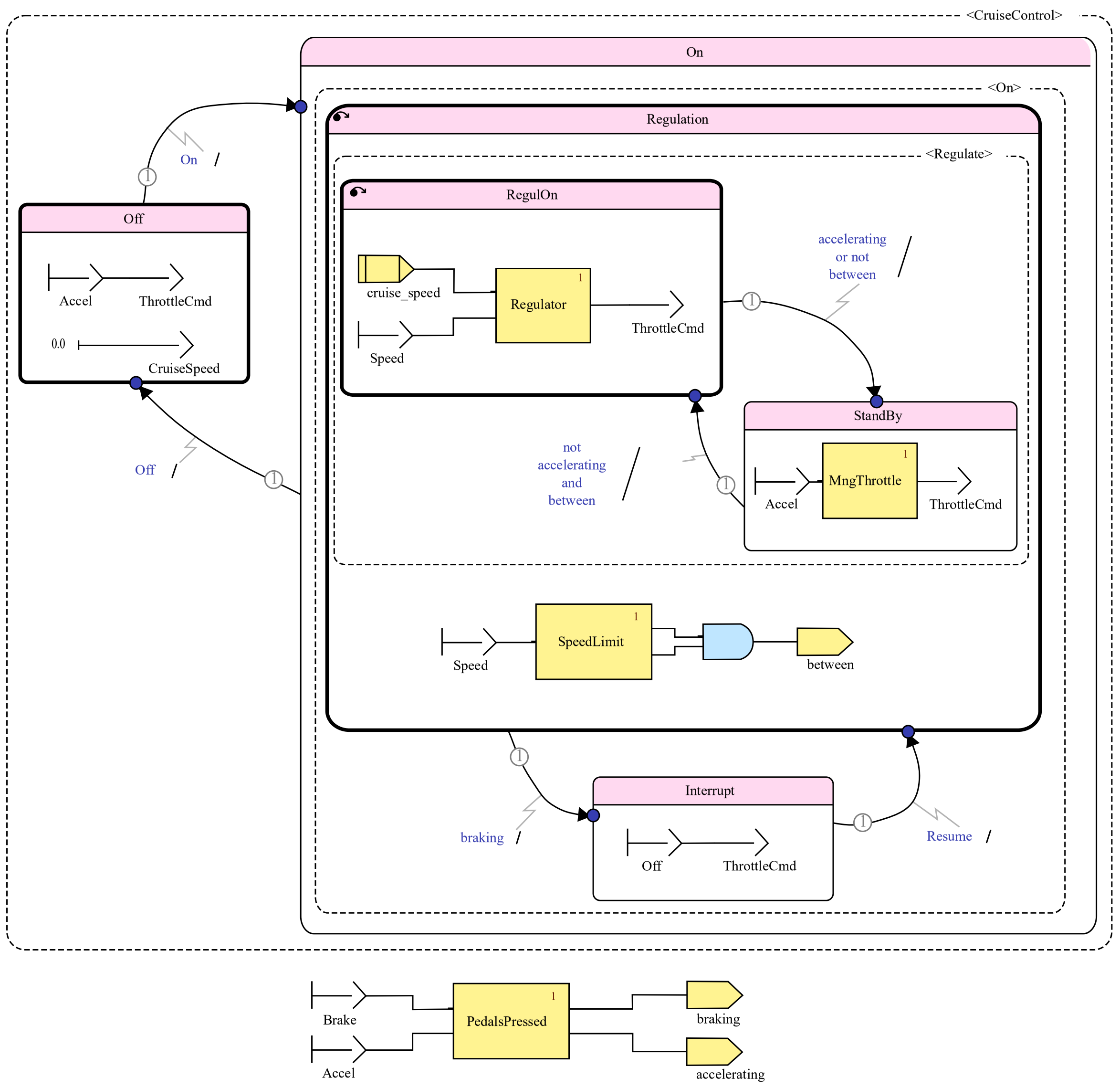
isae Institut Supérieur de l'Aéronautique et de l'Espace SUPAERO

ENAC 1ère école aéronautique européenne

Embedded Systems



Model-Based Design: SCADE



Lustre and Certified Compilation

INDUSTRIAL QUALIFICATION

Ansys

SCADE Suite
KCG Code Generator

DO-178C

Lustre and Certified Compilation

Industrial Qualification

Ansys

SCADE Suite
KCG Code Generator

DO-178C

Verified Compilation

Vélus

Lustre and Certified Compilation

Industrial Qualification

Ansys

SCADE Suite
KCG Code Generator

DO-178C

Verified Compilation

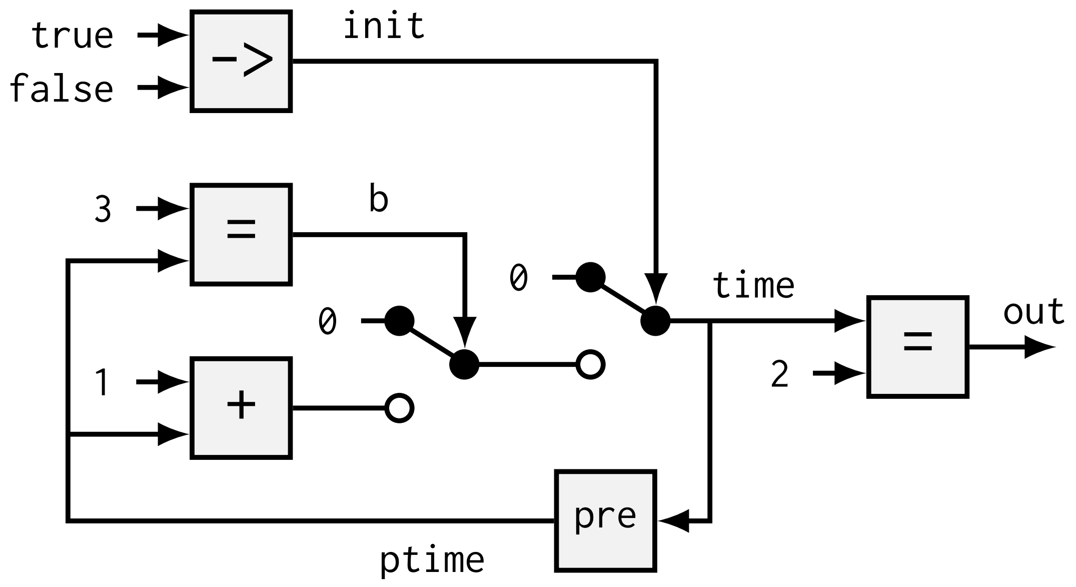
Vélus

Translation Validation

LustreC

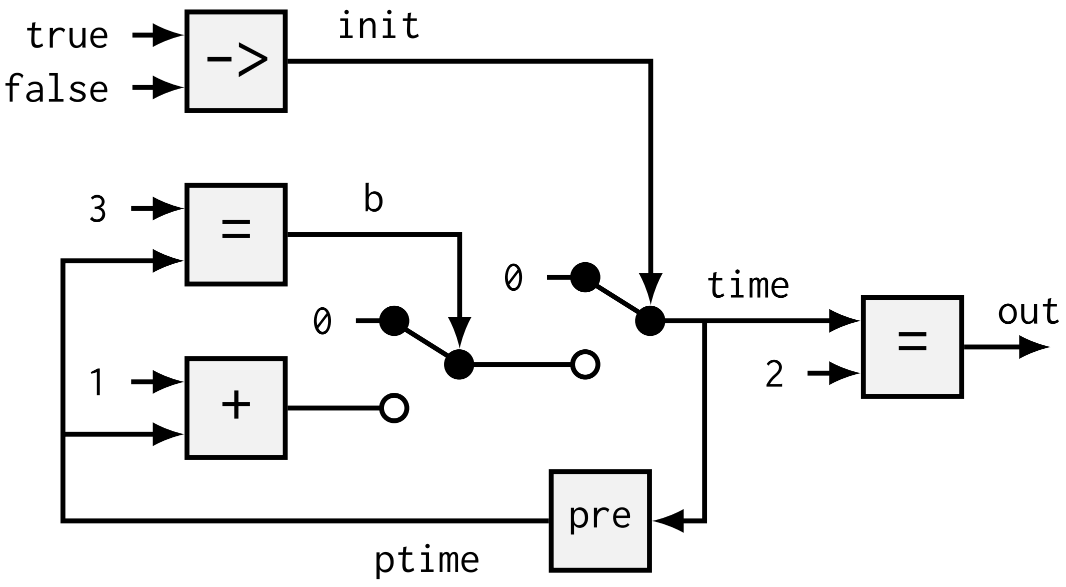
frama C
Software Analyzers

Example



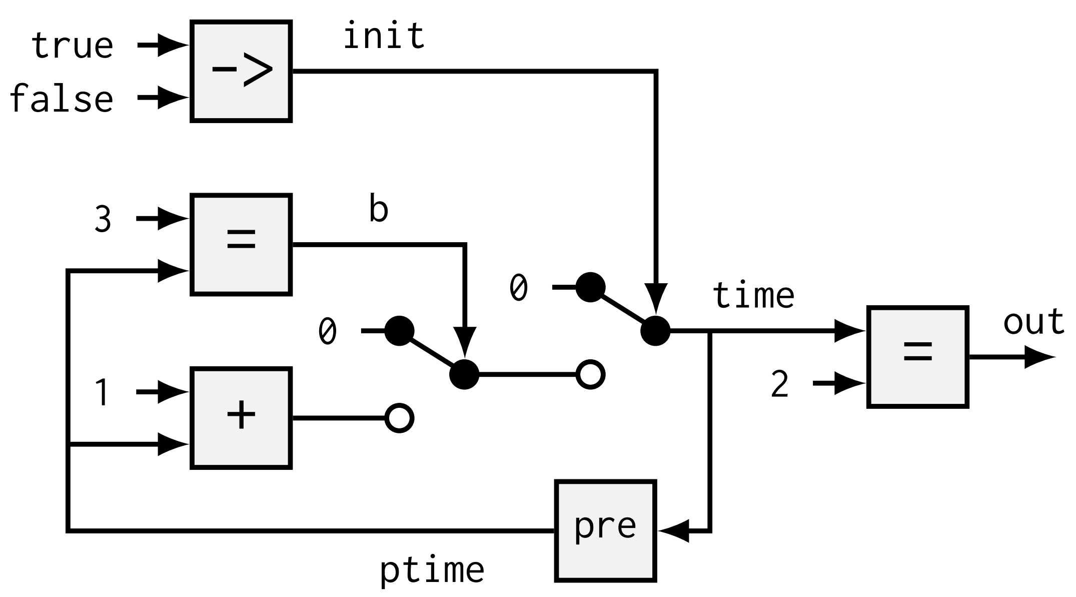
```
node count() returns (out: bool)
var time, ptime: int;
    init, b: bool;
let
  init = true -> false;
  b = (ptime = 3);
  time = if init then 0
         else if b then 0
         else ptime + 1;
  ptime = pre time;
  out = (time = 2);
tel
```

EXAMPLE



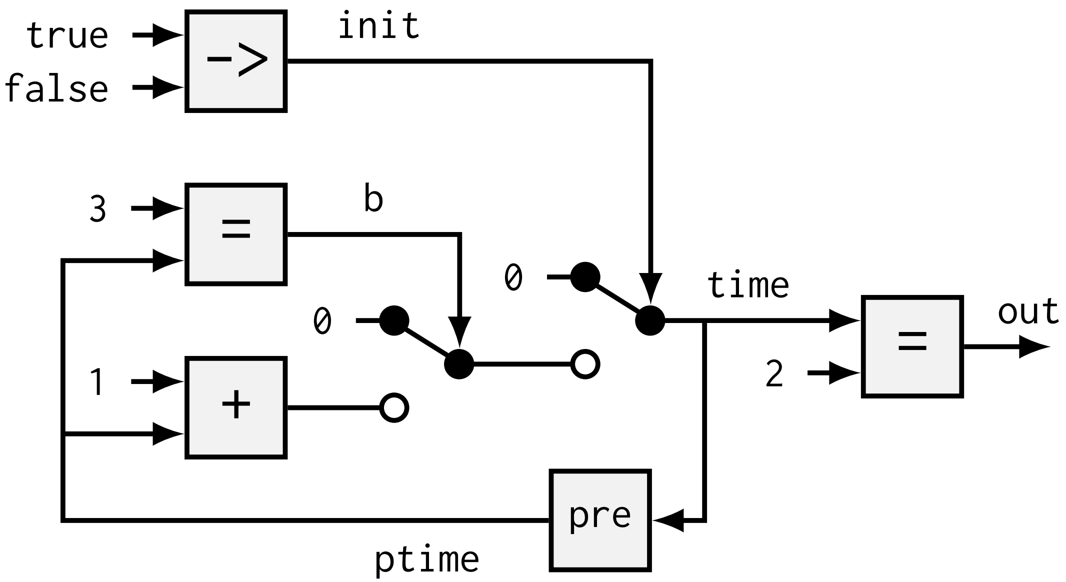
```
node count() returns (out: bool)
var time, ptime: int;
    init, b: bool;
let
  out = (time = 2);
  b = (ptime = 3);
  ptime = pre time;
  time = if init then 0
         else if b then 0
         else ptime + 1;
  init = true -> false;
tel
```

EXAMPLE



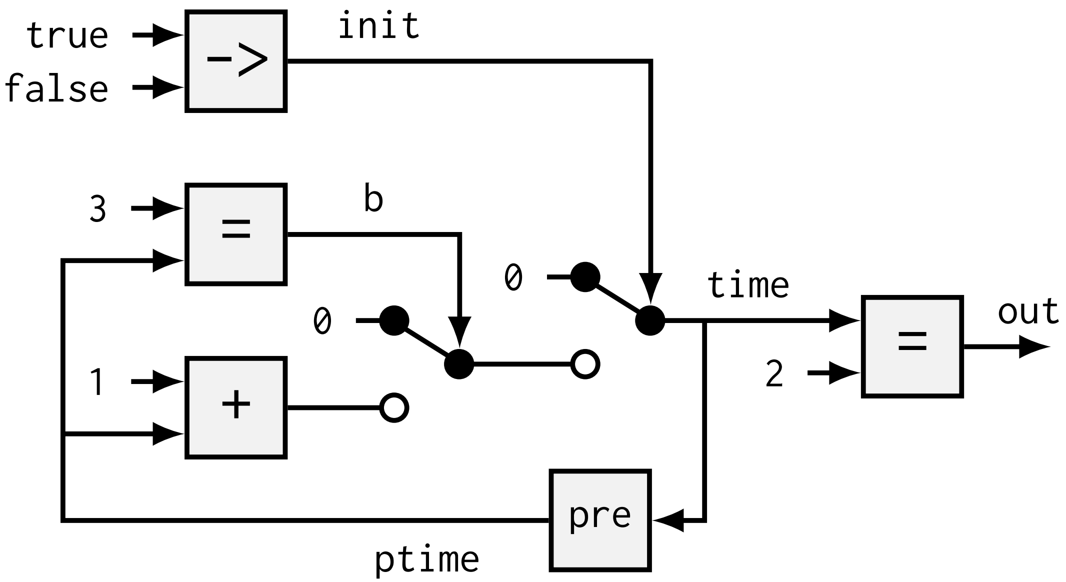
```
node count() returns (out: bool)
var time, ptime: int;
    init, b: bool;
let
  init = true -> false;
  b = (ptime = 3);
  time = if init then 0
         else if b then 0
         else ptime + 1;
  ptime = pre time;
  out = (time = 2);
tel
```

Example



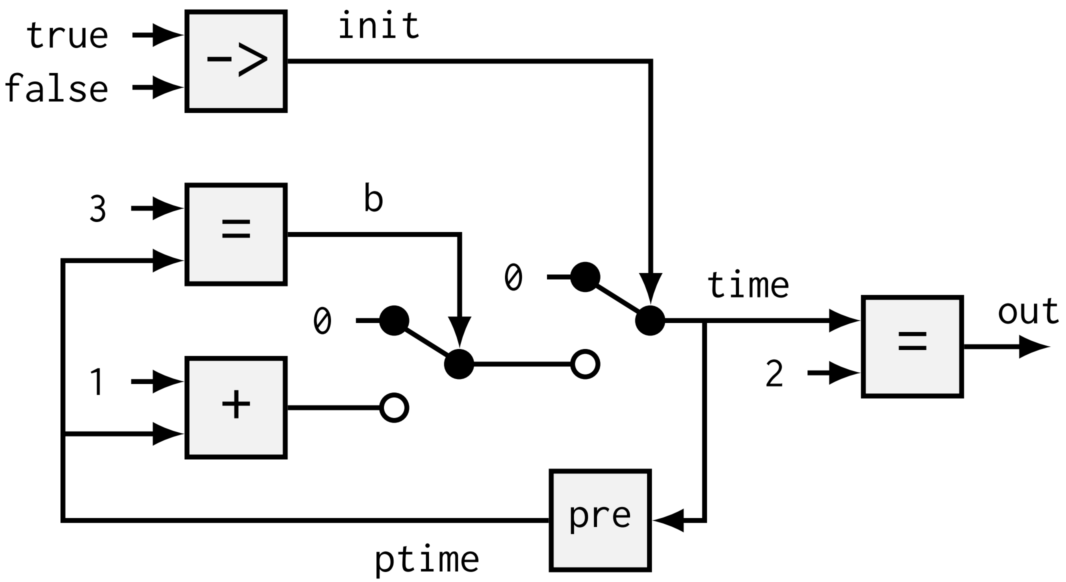
```
node count() returns (out: bool)
var time, ptime: int;
    init, b: bool;
let
  init = true -> false;
  b = (ptime = 3);
  time = if init then 0
         else if b then 0
         else ptime + 1;
  ptime = pre time;
  out = (time = 2);
tel
```

Example



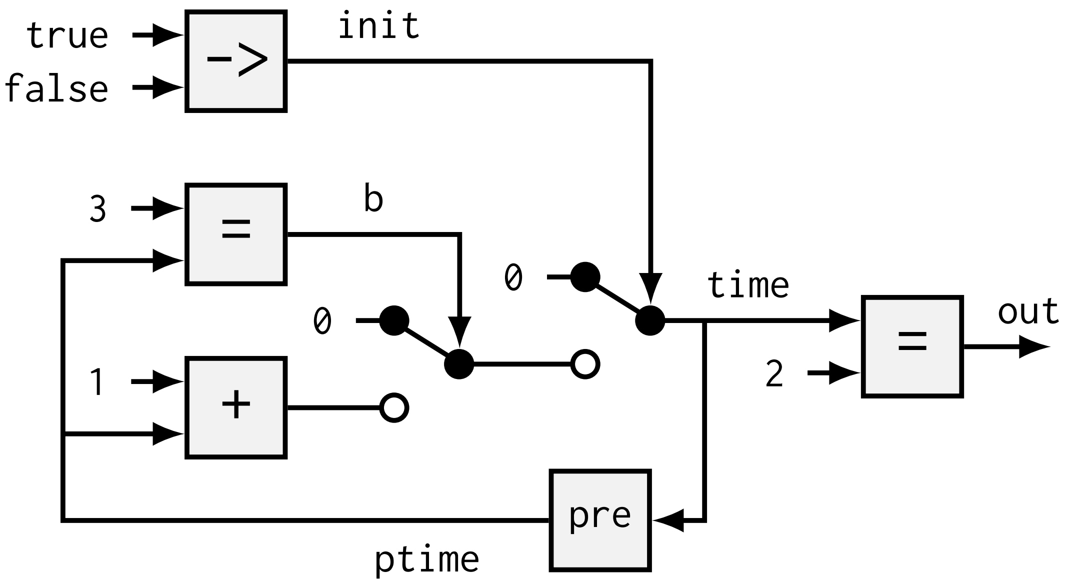
```
node count() returns (out: bool)
var time, ptime: int;
    init, b: bool;
let
  init = true -> false;
  b = (ptime = 3);
  time = if init then 0
         else if b then 0
         else ptime + 1;
  ptime = pre time;
  out = (time = 2);
tel
```

Example



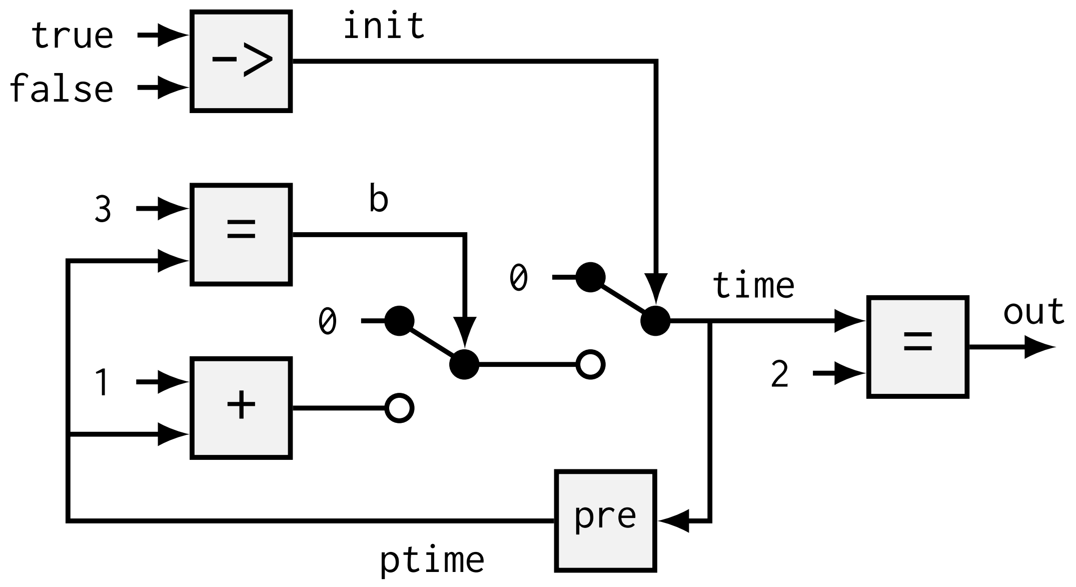
```
node count() returns (out: bool)
var time, ptime: int;
    init, b: bool;
let
  init = true -> false;
  b = (ptime = 3);
  time = if init then 0
         else if b then 0
         else ptime + 1;
  ptime = pre time;
  out = (time = 2);
tel
```

Example



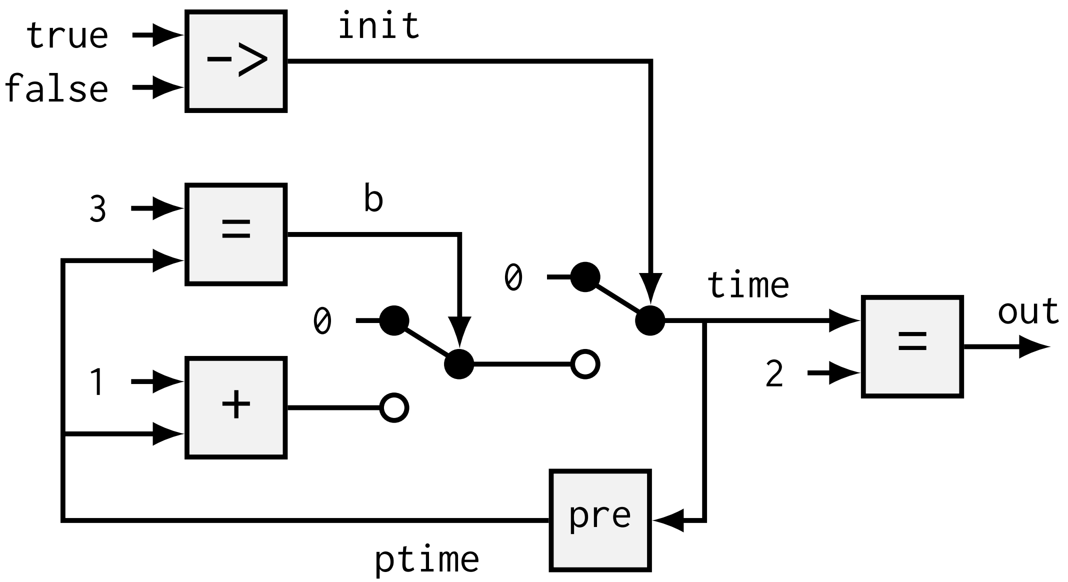
```
node count() returns (out: bool)
var time, ptime: int;
    init, b: bool;
let
  init = true -> false;
  b = (ptime = 3);
  time = if init then 0
         else if b then 0
         else ptime + 1;
  ptime = pre time;
  out = (time = 2);
tel
```

Example



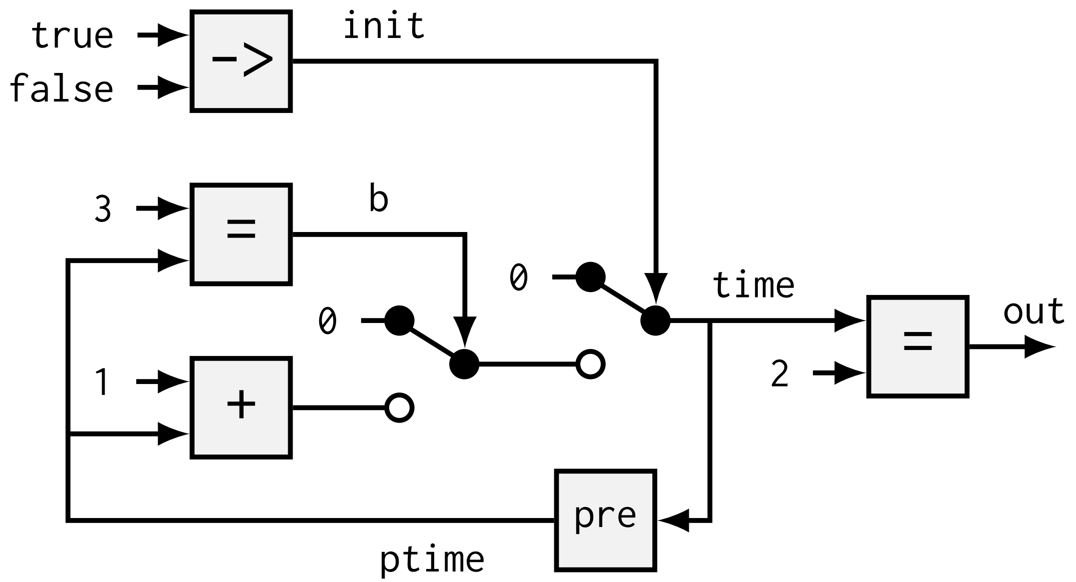
```
node count() returns (out: bool)
var time, ptime: int;
    init, b: bool;
let
  init = true -> false;
  b = (ptime = 3);
  time = if init then 0
         else if b then 0
         else ptime + 1;
  ptime = pre time;
  out = (time = 2);
tel
```

Example

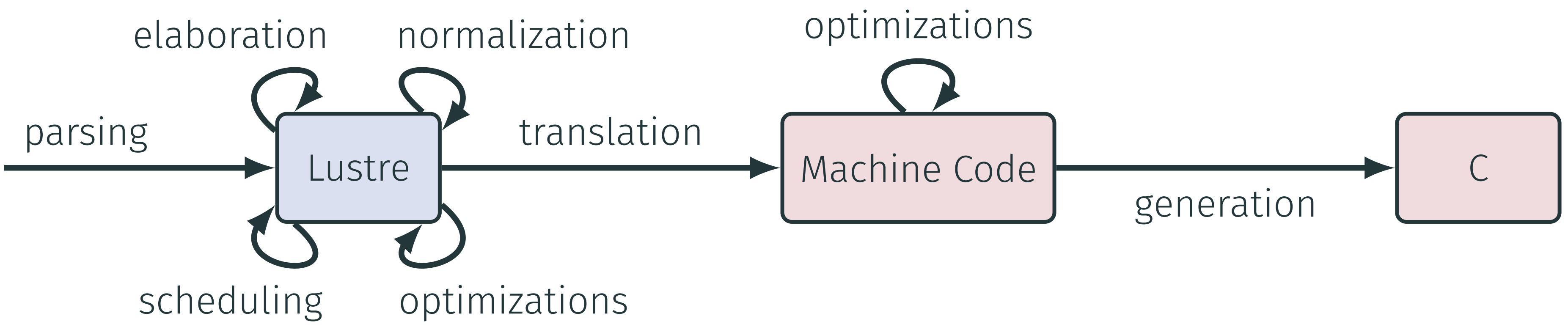


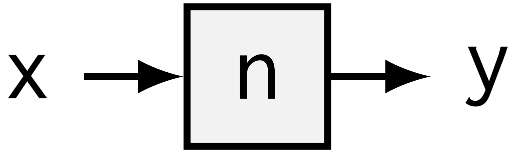
```
node count() returns (out: bool)
var time, ptime: int;
    init, b: bool;
let
  init = true -> false;
  b = (ptime = 3);
  time = if init then 0
         else if b then 0
         else ptime + 1;
  ptime = pre time;
  out = (time = 2);
tel
```

Example

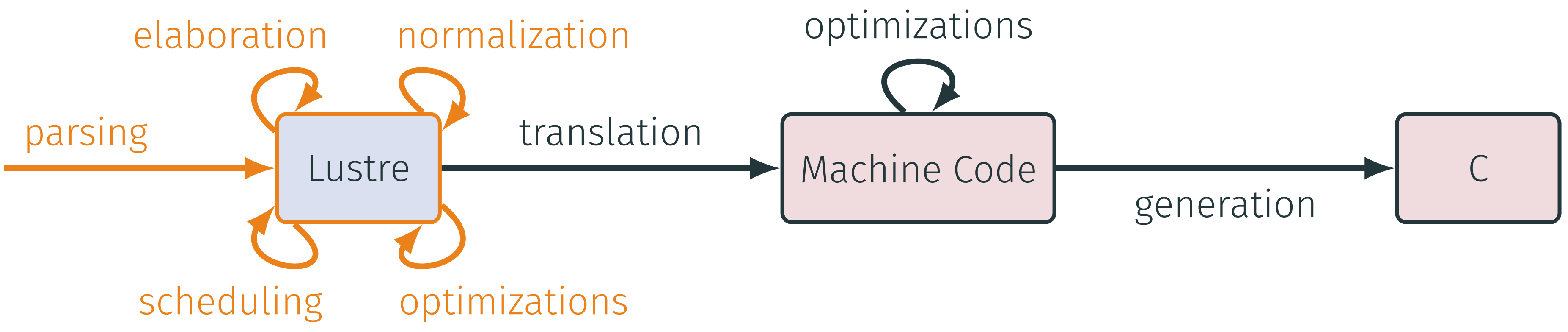


```
node count() returns (out: bool)
var time, ptime: int;
    init, b: bool;
let
  init = true -> false;
  b = (ptime = 3);
  time = if init then 0
         else if b then 0
         else ptime + 1;
  ptime = pre time;
  out = (time = 2);
tel
```

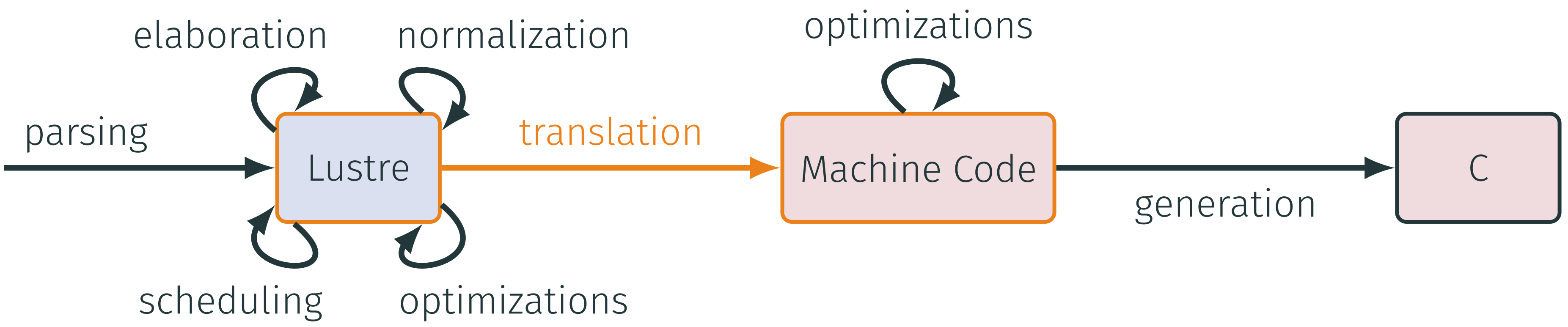




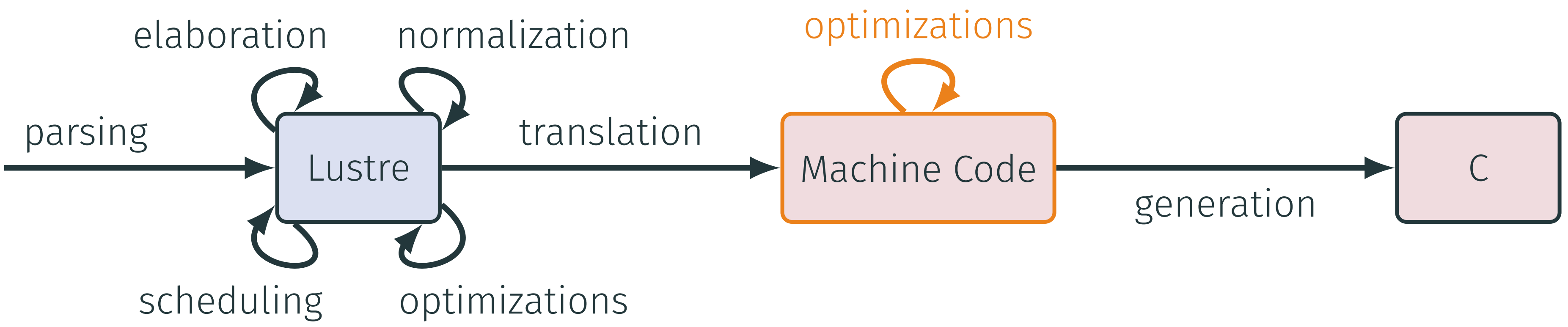
```
// state initialization
n_reset(&self);

// execution loop
while (1) {
  read_input(&x);

  // execution step
  n_step(x, &y, &self);

  write_output(y);
}
```



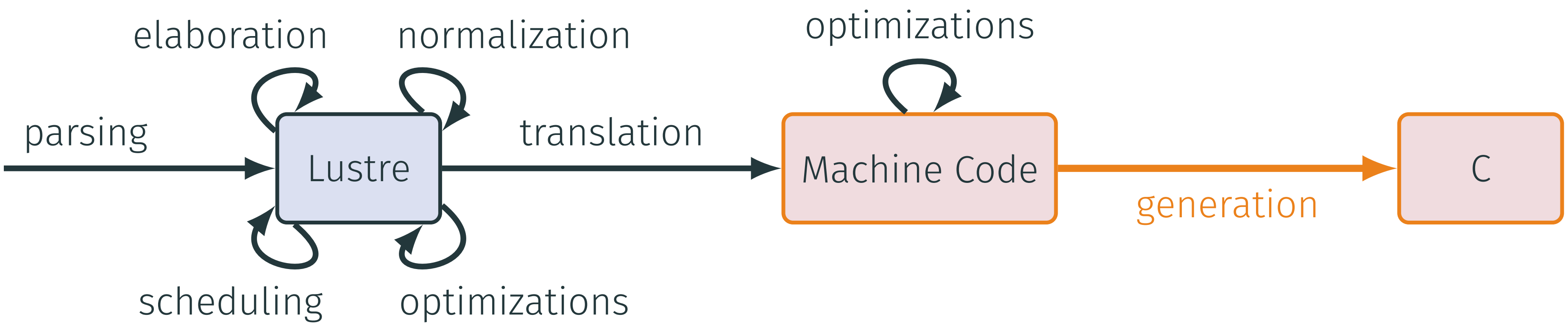
```
node count() returns (out: bool)
var time, ptime: int; init, b: bool;
let
  init = true -> false;
  b = (ptime = 3);
  time = if init then 0 else if b then 0 else ptime + 1;
  ptime = pre time;
  out = (time = 2);
tel
```



```
machine count {
  state ptime: int;
  instance a: _arrow;

  step () => (out: bool) {
    var time: int; init, b: bool;

    init := a.step(true, false);
    b := state(ptime) = 3;
    if (init) { time := 0 } else {
      if (b) { time := 0 } else { time := state(ptime) + 1 }
    }
    state(ptime) := time;
    out := (time = 2);
  }
}
```



```
machine count {
  state ptime: int;
  instance a: _arrow;

  step () => (out: bool) {
    var time: int; init, b: bool;

    init := a.step(true, false);
    b := state(ptime) = 3;
    if (init) { time := 0 } else {
      if (b) { time := 0 } else { time := state(ptime) + 1 }
    }
    state(ptime) := time;
    out := (time = 2);
  }
}
```



elaboration

normalization

optimizations

parsing

Lustre

translation

Machine Code

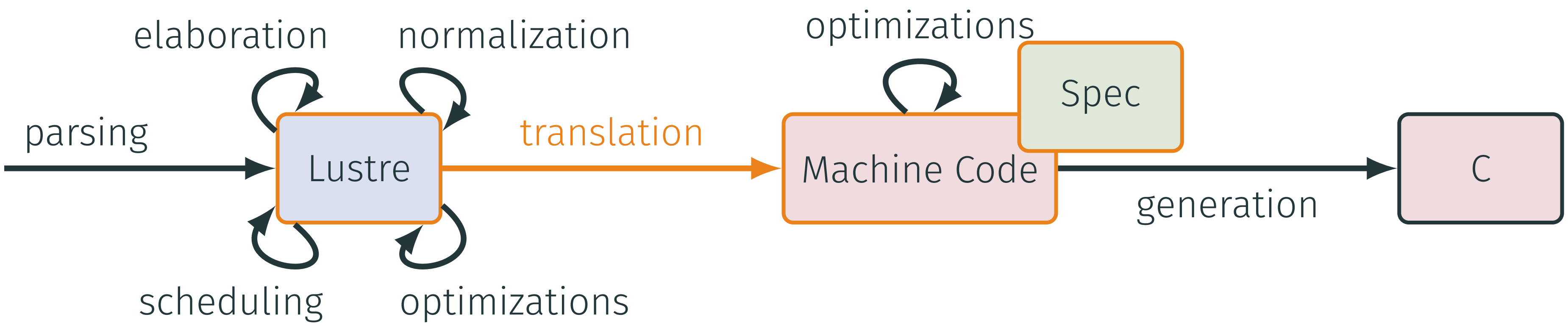
C

generation

scheduling

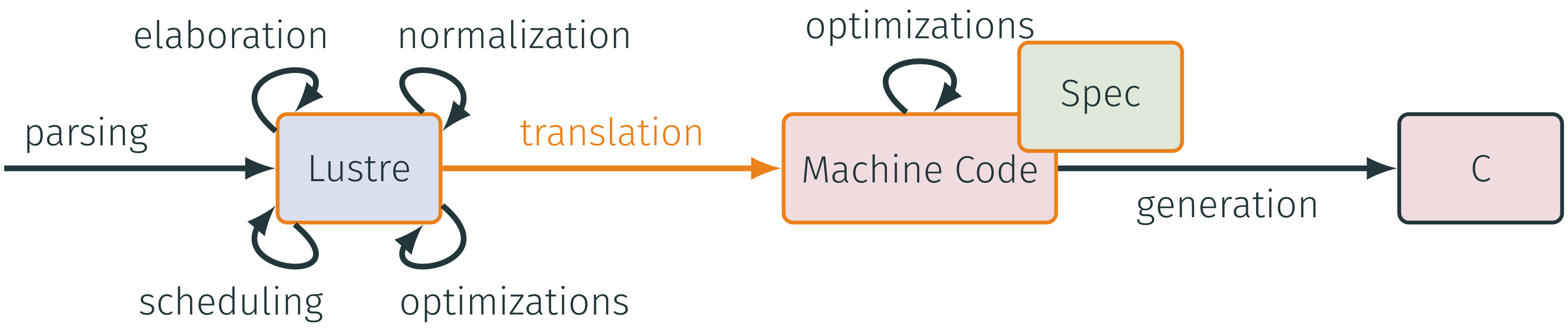
optimizations

```
struct count_mem { _Bool _reset; int ptime; struct _arrow_mem *a; };

#define count_set_reset(self) { self->_reset = 1; }
void count_clear_reset(struct count_mem *self ) {
  if (self->_reset) {
    self->_reset = 0;
    _arrow_reset(self->a);
  }
}

void count_step(_Bool *out, struct count_mem *self) {
  int time; _Bool init, b;
  count_clear_reset(self);
  init = _arrow_step(self->a);
  b = self->ptime == 3;
  if (init) { time = 0; } else { if (b) { time = 0; } else { time = self->ptime + 1; } }
  self->ptime = time;
  *out = time == 2;
}
```



```
node count() returns (out: bool)
var time, ptime: int;
    init, b: bool;
let
  init = true -> false;
  b = (ptime = 3);
  time = if init then 0
         else if b then 0
         else ptime + 1;
  ptime = pre time;
  out = (time = 2);
tel
```

$$count\_tr(S, out, S') \triangleq$$
$$count\_tr_5(S, out, S')$$



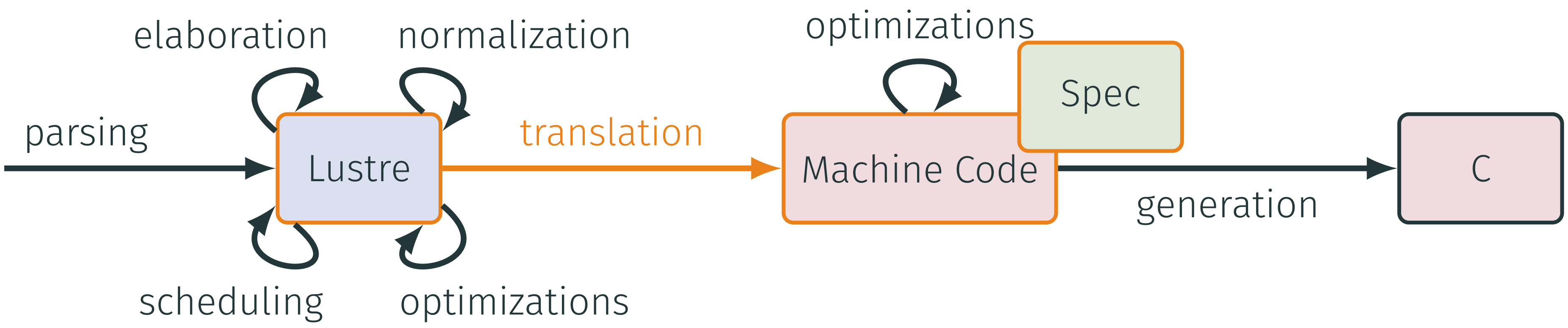
```
node count() returns (out: bool)
var time, ptime: int;
    init, b: bool;
let
  init = true -> false;
  b = (ptime = 3);
  time = if init then 0
         else if b then 0
         else ptime + 1;
  ptime = pre time;
  out = (time = 2);
tel
```

$$count\_tr(S, out, S') \triangleq$$

$$\exists time,$$

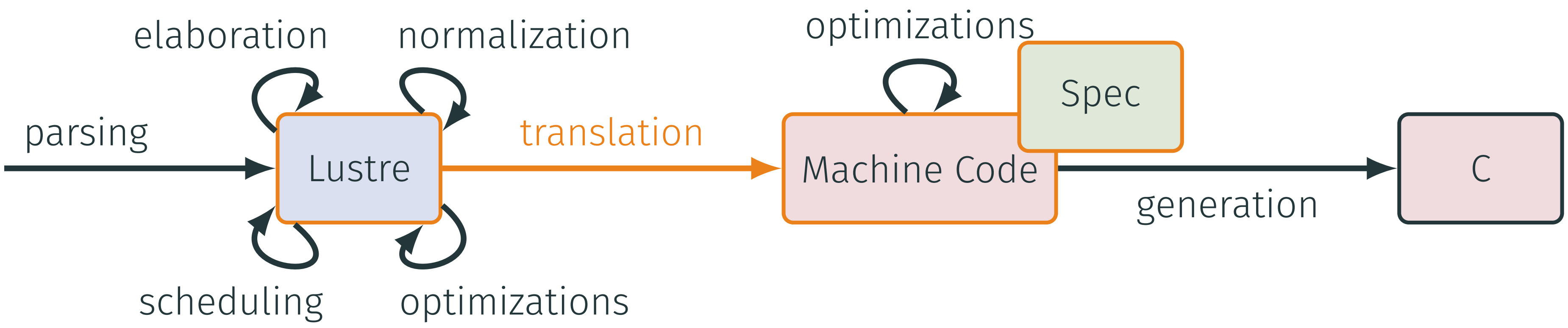
$$count\_tr_4(S, time, S')$$

$$\wedge\ out = (time = 2)$$



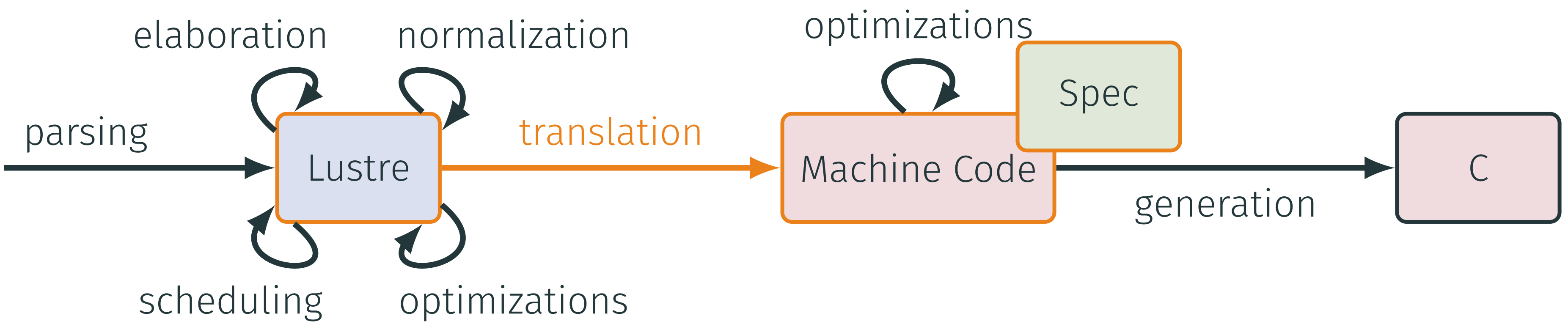
```
node count() returns (out: bool)
var time, ptime: int;
    init, b: bool;
let
  init = true -> false;
  b = (ptime = 3);
  time = if init then 0
         else if b then 0
         else ptime + 1;
  ptime = pre time;
  out = (time = 2);
tel
```

$$
\begin{aligned}
&count\_tr(S, out, S') \triangleq \\
&\quad \exists time, \\
&\qquad count\_tr_3(S, time, S') \\
&\qquad \land S'(ptime) = time \\
&\qquad \land out = (time = 2)
\end{aligned}
$$



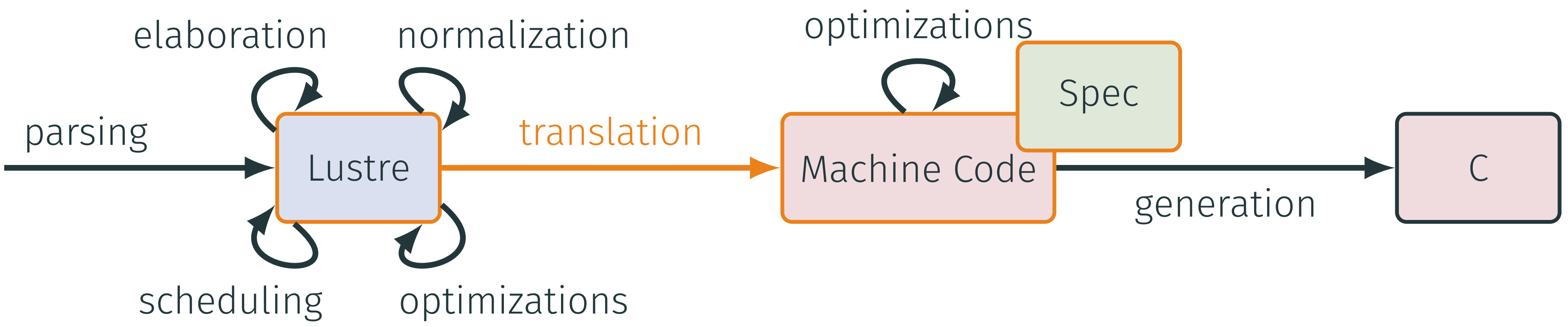
```
node count() returns (out: bool)
var time, ptime: int;
    init, b: bool;
let
  init = true -> false;
  b = (ptime = 3);
  time = if init then 0
         else if b then 0
         else ptime + 1;
  ptime = pre time;
  out = (time = 2);
tel
```

$$
\begin{aligned}
& count\_tr(S, out, S') \triangleq \\
& \quad \exists time, \\
& \qquad \exists b,\ init, \\
& \qquad\quad count\_tr_2(S, time, S') \\
& \qquad\quad \wedge\ init \implies time = 0 \\
& \qquad\quad \wedge\ (\neg init \wedge b) \implies time = 0 \\
& \qquad\quad \wedge\ (\neg init \wedge \neg b) \implies time = S(pti \\
& \qquad \wedge\ S'(ptime) = time \\
& \qquad \wedge\ out = (time = 2)
\end{aligned}
$$



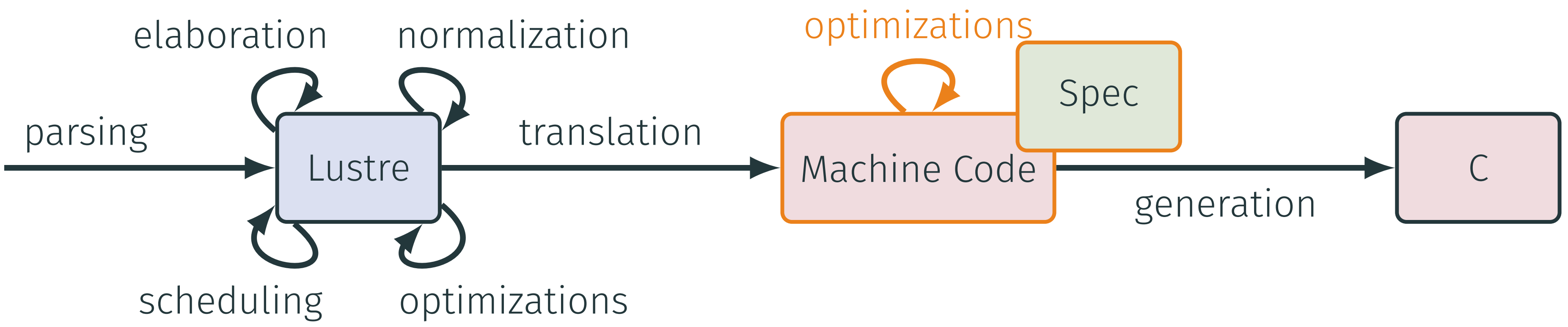
```
node count() returns (out: bool)
var time, ptime: int;
    init, b: bool;
let
  init = true -> false;
  b = (ptime = 3);
  time = if init then 0
         else if b then 0
         else ptime + 1;
  ptime = pre time;
  out = (time = 2);
tel
```

$$
\begin{aligned}
&\mathit{count\_tr}(S, \mathit{out}, S') \triangleq\\
&\quad \exists \mathit{time},\\
&\qquad \exists b, \mathit{init},\\
&\qquad\quad \mathit{count\_tr}_1(S, \mathit{time}, S')\\
&\qquad\quad \wedge\, b = (S(\mathit{ptime}) = 3)\\
&\qquad\quad \wedge\, \mathit{init} \implies \mathit{time} = 0\\
&\qquad\quad \wedge\, (\neg\mathit{init} \wedge b) \implies \mathit{time} = 0\\
&\qquad\quad \wedge\, (\neg\mathit{init} \wedge \neg b) \implies \mathit{time} = S(\mathit{pti}\\
&\quad\; \wedge\, S'(\mathit{ptime}) = \mathit{time}\\
&\quad\; \wedge\, \mathit{out} = (\mathit{time} = 2)
\end{aligned}
$$



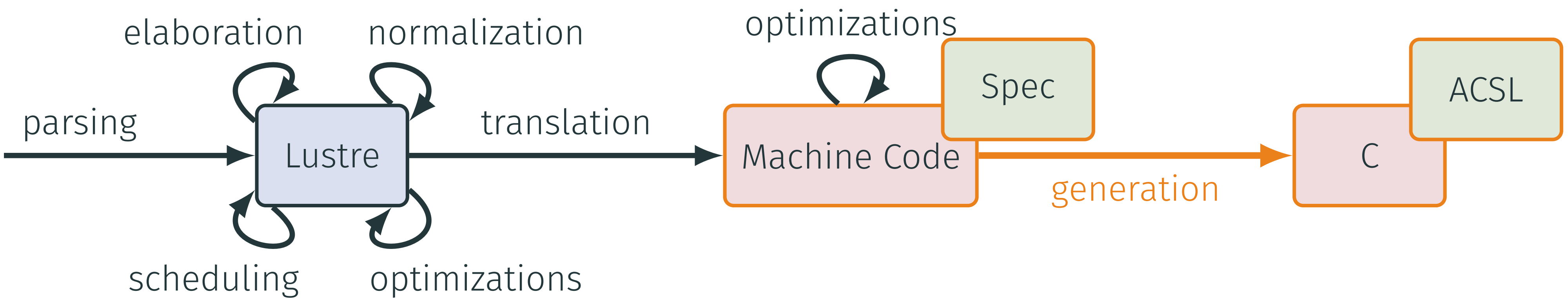
```
node count() returns (out: bool)
var time, ptime: int;
    init, b: bool;
let
  init = true -> false;
  b = (ptime = 3);
  time = if init then 0
         else if b then 0
         else ptime + 1;
  ptime = pre time;
  out = (time = 2);
tel
```

$$
\begin{aligned}
& \mathit{count\_tr}(S, \mathit{out}, S') \triangleq \\
& \quad \exists \mathit{time}, \\
& \qquad \exists b, \mathit{init}, \\
& \qquad\quad \mathit{arrow\_tr}(S[a], \mathit{init}, S'[a]) \\
& \qquad\quad \wedge b = (S(\mathit{ptime}) = 3) \\
& \qquad\quad \wedge \mathit{init} \implies \mathit{time} = 0 \\
& \qquad\quad \wedge (\neg \mathit{init} \wedge b) \implies \mathit{time} = 0 \\
& \qquad\quad \wedge (\neg \mathit{init} \wedge \neg b) \implies \mathit{time} = S(\mathit{pti} \\
& \qquad \wedge S'(\mathit{ptime}) = \mathit{time} \\
& \qquad \wedge \mathit{out} = (\mathit{time} = 2)
\end{aligned}
$$

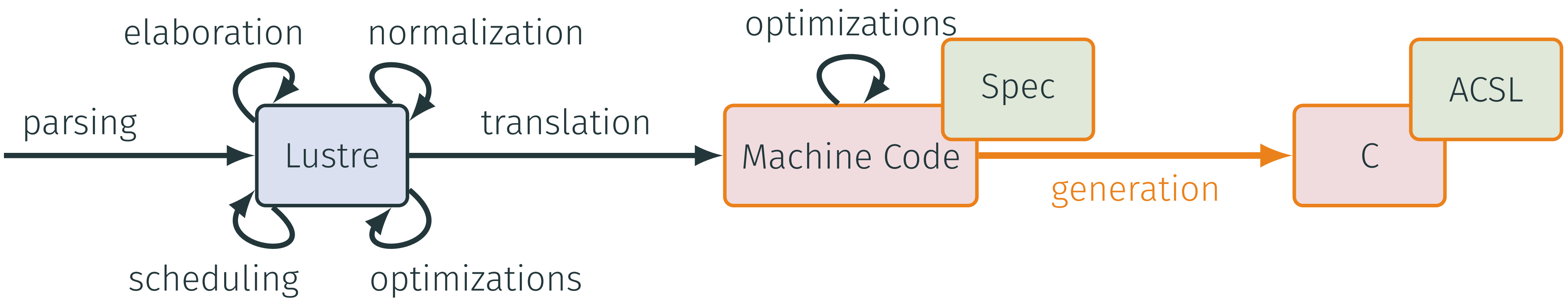


```
node count() returns (out: bool)
var time, ptime: int;
    init, b: bool;
let
  init = true -> false;
  b = (ptime = 3);
  time = if init then 0
         else if b then 0
         else ptime + 1;
  ptime = pre time;
  out = (time = 2);
tel
```

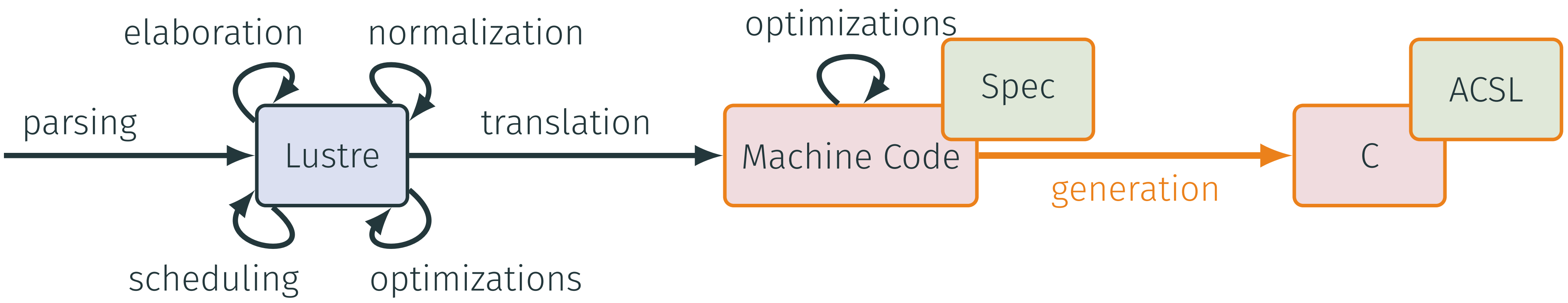
$$
\begin{aligned}
&\mathit{count\_tr}(S, \mathit{out}, S') \triangleq \\
&\quad \exists \mathit{time}, \\
&\qquad \exists b, \mathit{init}, \\
&\qquad\quad \mathit{arrow\_tr}(S[a], \mathit{init}, S'[a]) \\
&\qquad\quad \wedge b = (S(\mathit{ptime}) = 3) \\
&\qquad\quad \wedge \mathit{init} \implies \mathit{time} = 0 \\
&\qquad\quad \wedge (\neg \mathit{init} \wedge b) \implies \mathit{time} = 0 \\
&\qquad\quad \wedge (\neg \mathit{init} \wedge \neg b) \implies \mathit{time} = S(\mathit{pti} \\
&\qquad \wedge S'(\mathit{ptime}) = \mathit{time} \\
&\qquad \wedge \mathit{out} = (\mathit{time} = 2)
\end{aligned}
$$



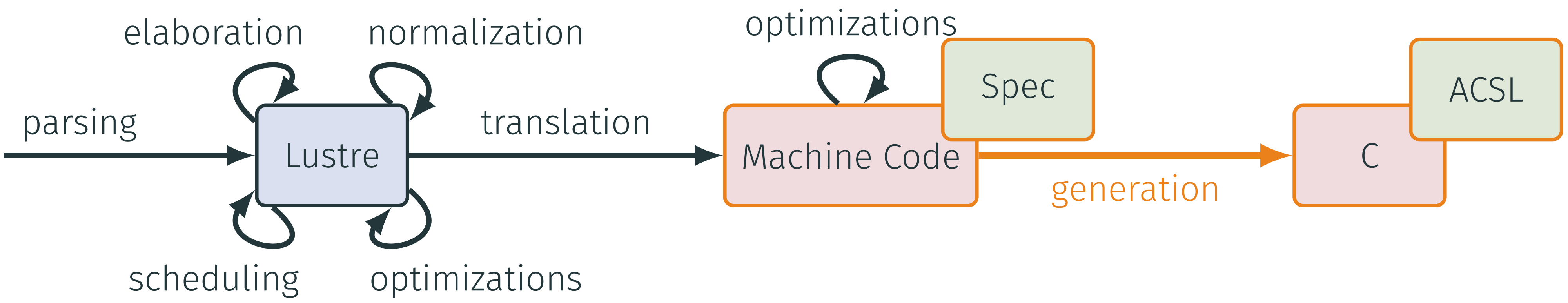
```
/*@ requires count_pack(*mem, self);
    ensures count_pack(*mem, self);
    ensures count_tr(\old(*mem), *out, *mem); */
void count_step(_Bool *out, struct count_mem *self)
     /*@ ghost (struct count_mem_ghost \ghost *mem) */ {
  int time; _Bool init, b;
  count_clear_reset(self);
  init = _arrow_step(self->a) /*@ ghost (&mem->a) */;
  //@ assert count_tr1(\at(*mem, Pre), b, *mem);
  b = self->ptime == 3;
  //@ assert count_tr2(\at(*mem, Pre), b, init, *mem);
  if (init) { time = 0; } else { if (b) { time = 0; } else { time = self->ptime + 1; } }
  //@ assert count_tr3(\at(*mem, Pre), time, *mem);
  self->ptime = time;
  //@ ghost mem->ptime = time;
  //@ assert count_tr4(\at(*mem, Pre), time, *mem);
  *out = time == 2;
  //@ assert count_tr5(\at(*mem, Pre), *out, *mem);
}
```



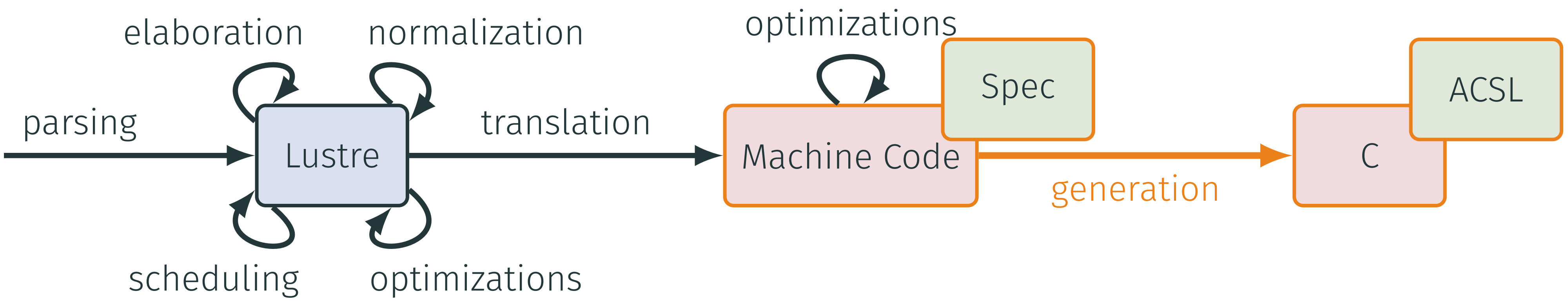
```
/*@ requires count_pack(*mem, self);
    ensures count_pack(*mem, self);
    ensures count_tr(\old(*mem), *out, *mem); */
void count_step(_Bool *out, struct count_mem *self)
     /*@ ghost (struct count_mem_ghost \ghost *mem) */ {
  int time; _Bool init, b;
  count_clear_reset(self);
  init = _arrow_step(self->a) /*@ ghost (&mem->a) */;
  //@ assert count_tr1(\at(*mem, Pre), b, *mem);
  b = self->ptime == 3;
  //@ assert count_tr2(\at(*mem, Pre), b, init, *mem);
  if (init) { time = 0; } else { if (b) { time = 0; } else { time = self->ptime + 1; } }
  //@ assert count_tr3(\at(*mem, Pre), time, *mem);
  self->ptime = time;
  //@ ghost mem->ptime = time;
  //@ assert count_tr4(\at(*mem, Pre), time, *mem);
  *out = time == 2;
  //@ assert count_tr5(\at(*mem, Pre), *out, *mem);
}
```



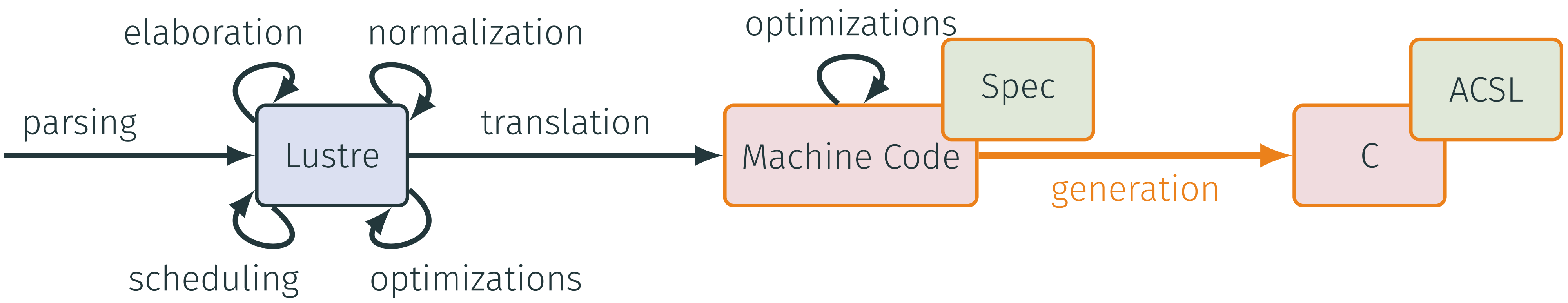
```
/*@ requires count_pack(*mem, self);
    ensures count_pack(*mem, self);
    ensures count_tr(\old(*mem), *out, *mem); */
void count_step(_Bool *out, struct count_mem *self)
     /*@ ghost (struct count_mem_ghost \ghost *mem) */ {
  int time; _Bool init, b;
  count_clear_reset(self);
  init = _arrow_step(self->a) /*@ ghost (&mem->a) */;
  //@ assert count_tr1(\at(*mem, Pre), b, *mem);
  b = self->ptime == 3;
  //@ assert count_tr2(\at(*mem, Pre), b, init, *mem);
  if (init) { time = 0; } else { if (b) { time = 0; } else { time = self->ptime + 1; } }
  //@ assert count_tr3(\at(*mem, Pre), time, *mem);
  self->ptime = time;
  //@ ghost mem->ptime = time;
  //@ assert count_tr4(\at(*mem, Pre), time, *mem);
  *out = time == 2;
  //@ assert count_tr5(\at(*mem, Pre), *out, *mem);
}
```



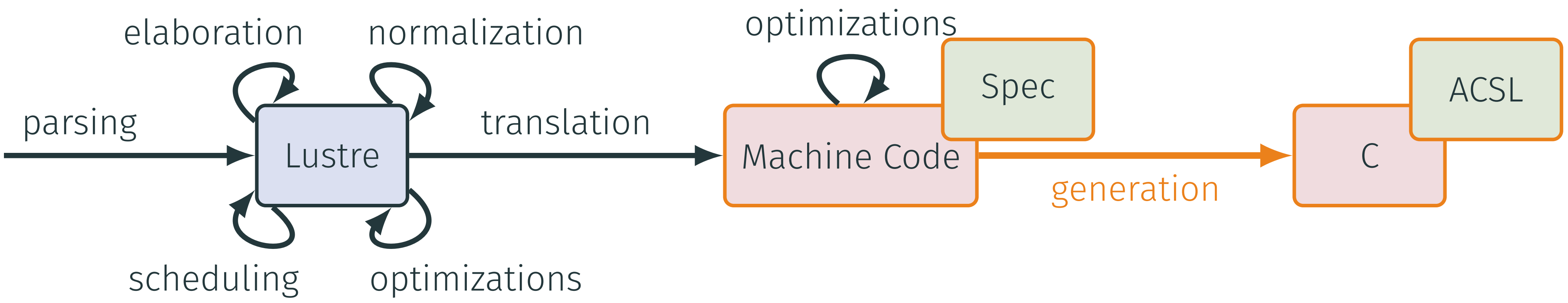
```
/*@ requires count_pack(*mem, self);
    ensures count_pack(*mem, self);
    ensures count_tr(\old(*mem), *out, *mem); */
void count_step(_Bool *out, struct count_mem *self)
     /*@ ghost (struct count_mem_ghost \ghost *mem) */ {
  int time; _Bool init, b;
  count_clear_reset(self);
  init = _arrow_step(self->a) /*@ ghost (&mem->a) */;
  //@ assert count_tr1(\at(*mem, Pre), b, *mem);
  b = self->ptime == 3;
  //@ assert count_tr2(\at(*mem, Pre), b, init, *mem);
  if (init) { time = 0; } else { if (b) { time = 0; } else { time = self->ptime + 1; } }
  //@ assert count_tr3(\at(*mem, Pre), time, *mem);
  self->ptime = time;
  //@ ghost mem->ptime = time;
  //@ assert count_tr4(\at(*mem, Pre), time, *mem);
  *out = time == 2;
  //@ assert count_tr5(\at(*mem, Pre), *out, *mem);
}
```



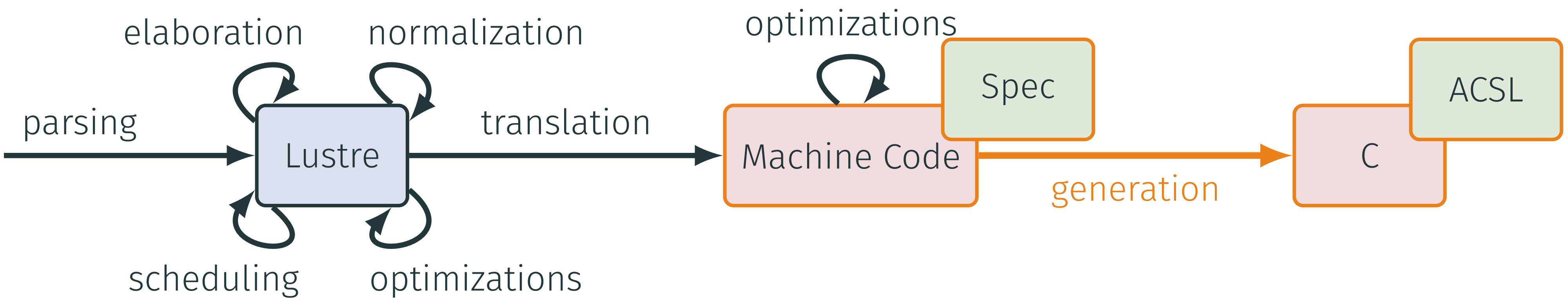
```
/*@ requires count_pack(*mem, self);
    ensures count_pack(*mem, self);
    ensures count_tr(\old(*mem), *out, *mem); */
void count_step(_Bool *out, struct count_mem *self)
     /*@ ghost (struct count_mem_ghost \ghost *mem) */ {
  int time; _Bool init, b;
  count_clear_reset(self);
  init = _arrow_step(self->a) /*@ ghost (&mem->a) */;
  //@ assert count_tr1(\at(*mem, Pre), b, *mem);
  b = self->ptime == 3;
  //@ assert count_tr2(\at(*mem, Pre), b, init, *mem);
  if (init) { time = 0; } else { if (b) { time = 0; } else { time = self->ptime + 1; } }
  //@ assert count_tr3(\at(*mem, Pre), time, *mem);
  self->ptime = time;
  //@ ghost mem->ptime = time;
  //@ assert count_tr4(\at(*mem, Pre), time, *mem);
  *out = time == 2;
  //@ assert count_tr5(\at(*mem, Pre), *out, *mem);
}
```



```
/*@ requires count_pack(*mem, self);
    ensures count_pack(*mem, self);
    ensures count_tr(\old(*mem), *out, *mem); */
void count_step(_Bool *out, struct count_mem *self)
     /*@ ghost (struct count_mem_ghost \ghost *mem) */ {
  int time; _Bool init, b;
  count_clear_reset(self);
  init = _arrow_step(self->a) /*@ ghost (&mem->a) */;
  //@ assert count_tr1(\at(*mem, Pre), b, *mem);
  b = self->ptime == 3;
  //@ assert count_tr2(\at(*mem, Pre), b, init, *mem);
  if (init) { time = 0; } else { if (b) { time = 0; } else { time = self->ptime + 1; } }
  //@ assert count_tr3(\at(*mem, Pre), time, *mem);
  self->ptime = time;
  //@ ghost mem->ptime = time;
  //@ assert count_tr4(\at(*mem, Pre), time, *mem);
  *out = time == 2;
  //@ assert count_tr5(\at(*mem, Pre), *out, *mem);
}
```

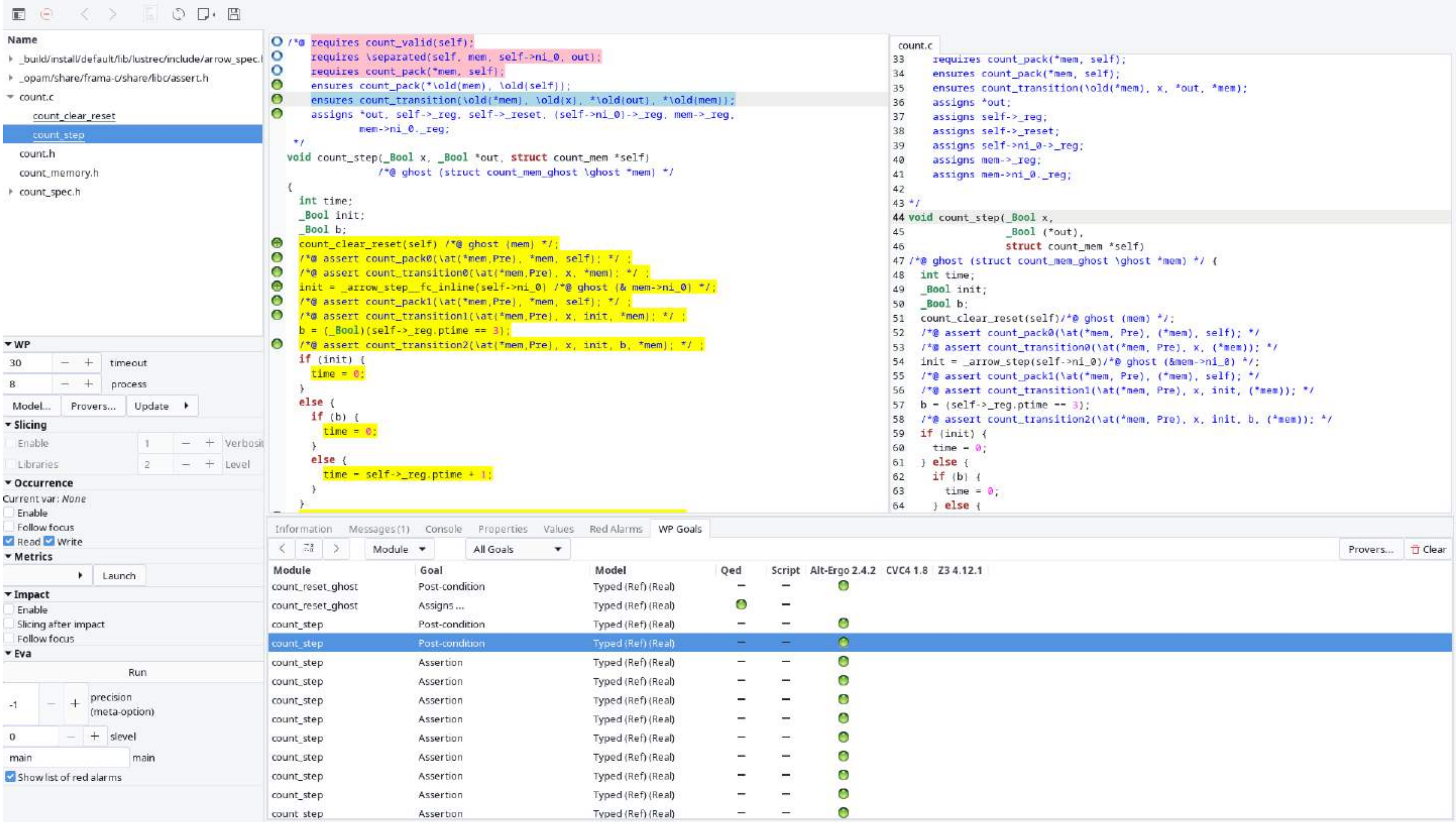


```
/*@ requires count_pack(*mem, self);
    ensures count_pack(*mem, self);
    ensures count_tr(\old(*mem), *out, *mem); */
void count_step(_Bool *out, struct count_mem *self)
     /*@ ghost (struct count_mem_ghost \ghost *mem) */ {
  int time; _Bool init, b;
  count_clear_reset(self);
  init = _arrow_step(self->a) /*@ ghost (&mem->a) */;
  //@ assert count_tr1(\at(*mem, Pre), b, *mem);
  b = self->ptime == 3;
  //@ assert count_tr2(\at(*mem, Pre), b, init, *mem);
  if (init) { time = 0; } else { if (b) { time = 0; } else { time = self->ptime + 1; } }
  //@ assert count_tr3(\at(*mem, Pre), time, *mem);
  self->ptime = time;
  //@ ghost mem->ptime = time;
  //@ assert count_tr4(\at(*mem, Pre), time, *mem);
  *out = time == 2;
  //@ assert count_tr5(\at(*mem, Pre), *out, *mem);
}
```



```
/*@ requires count_pack(*mem, self);
    ensures count_pack(*mem, self);
    ensures count_tr(\old(*mem), *out, *mem); */
void count_step(_Bool *out, struct count_mem *self)
     /*@ ghost (struct count_mem_ghost \ghost *mem) */ {
  int time; _Bool init, b;
  count_clear_reset(self);
  init = _arrow_step(self->a) /*@ ghost (&mem->a) */;
  //@ assert count_tr1(\at(*mem, Pre), b, *mem);
  b = self->ptime == 3;
  //@ assert count_tr2(\at(*mem, Pre), b, init, *mem);
  if (init) { time = 0; } else { if (b) { time = 0; } else { time = self->ptime + 1; } }
  //@ assert count_tr3(\at(*mem, Pre), time, *mem);
  self->ptime = time;
  //@ ghost mem->ptime = time;
  //@ assert count_tr4(\at(*mem, Pre), time, *mem);
  *out = time == 2;
  //@ assert count_tr5(\at(*mem, Pre), *out, *mem);
}
```

Verification with Frama-C



```c
/*@ requires count_valid(self);
    requires \separated(self, mem, self->ni_0, out);
    requires count_pack(*mem, self);
    ensures count_pack(*\old(mem), \old(self));
    ensures count_transition(\old(*mem), \old(x), *\old(out), *\old(mem));
    assigns *out, self->_reg, self->_reset, (self->ni_0)->_reg, mem->_reg,
            mem->ni_0._reg;
 */
void count_step(_Bool x, _Bool *out, struct count_mem *self)
              /*@ ghost (struct count_mem_ghost \ghost *mem) */
{
  int time;
  _Bool init;
  _Bool b;
  count_clear_reset(self) /*@ ghost (mem) */;
  /*@ assert count_pack0(\at(*mem,Pre), *mem, self); */ ;
  /*@ assert count_transition0(\at(*mem,Pre), x, *mem); */ ;
  init = _arrow_step__fc_inline(self->ni_0) /*@ ghost (& mem->ni_0) */;
  /*@ assert count_pack1(\at(*mem,Pre), *mem, self); */ ;
  /*@ assert count_transition1(\at(*mem,Pre), x, init, *mem); */ ;
  b = (_Bool)(self->_reg.ptime == 3);
  /*@ assert count_transition2(\at(*mem,Pre), x, init, b, *mem); */ ;
  if (init) {
    time = 0;
  }
  else {
    if (b) {
      time = 0;
    }
    else {
      time = self->_reg.ptime + 1;
    }
  }
```

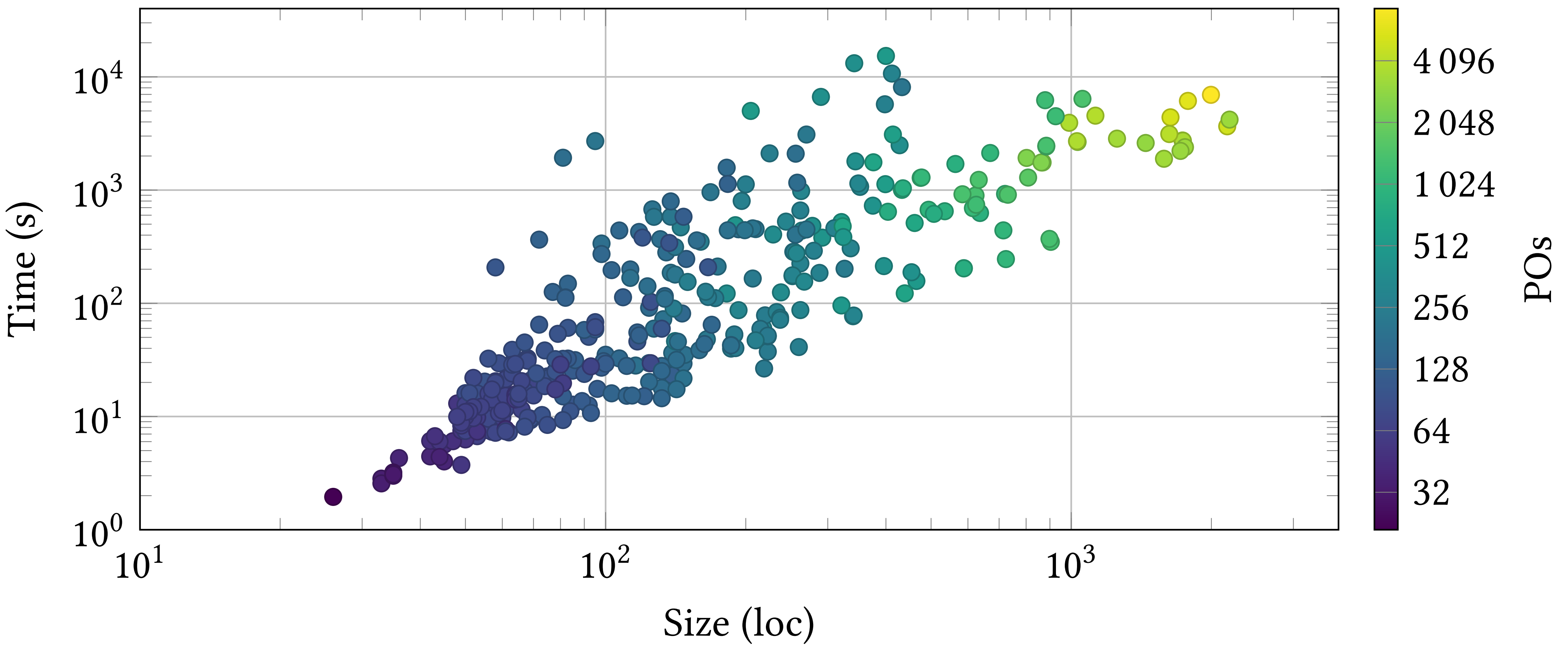
```c
   requires count_pack(*mem, self);
   ensures count_pack(*mem, self);
   ensures count_transition(\old(*mem), x, *out, *mem);
   assigns *out;
   assigns self->_reg;
   assigns self->_reset;
   assigns self->ni_0->_reg;
   assigns mem->_reg;
   assigns mem->ni_0._reg;

*/
void count_step(_Bool x,
                _Bool (*out),
                struct count_mem *self)
/*@ ghost (struct count_mem_ghost \ghost *mem) */ {
  int time;
  _Bool init;
  _Bool b;
  count_clear_reset(self)/*@ ghost (mem) */;
  /*@ assert count_pack0(\at(*mem, Pre), (*mem), self); */
  /*@ assert count_transition0(\at(*mem, Pre), x, (*mem)); */
  init = _arrow_step(self->ni_0)/*@ ghost (&mem->ni_0) */;
  /*@ assert count_pack1(\at(*mem, Pre), (*mem), self); */
  /*@ assert count_transition1(\at(*mem, Pre), x, init, (*mem)); */
  b = (self->_reg.ptime == 3);
  /*@ assert count_transition2(\at(*mem, Pre), x, init, b, (*mem)); */
  if (init) {
    time = 0;
  } else {
    if (b) {
      time = 0;
    } else {
```

Module	Goal	Model	Qed	Script	Alt-Ergo 2.4.2	CVC4 1.8	Z3 4.12.1
count_reset_ghost	Post-condition	Typed (Ref) (Real)	–	–	●		
count_reset_ghost	Assigns ...	Typed (Ref) (Real)	●	–			
count_step	Post-condition	Typed (Ref) (Real)	–	–	●		
count_step	Post-condition	Typed (Ref) (Real)	–	–	●		
count_step	Assertion	Typed (Ref) (Real)	–	–	●		
count_step	Assertion	Typed (Ref) (Real)	–	–	●		
count_step	Assertion	Typed (Ref) (Real)	–	–	●		
count_step	Assertion	Typed (Ref) (Real)	–	–	●		
count_step	Assertion	Typed (Ref) (Real)	–	–	●		
count_step	Assertion	Typed (Ref) (Real)	–	–	●		
count_step	Assertion	Typed (Ref) (Real)	–	–	●		
count_step	Assertion	Typed (Ref) (Real)	–	–	●		
count_step	Assertion	Typed (Ref) (Real)	–	–	●		

Results

~400 files ✓ 92.7%

~231 500 POs ✓ 99.9%



OPTIMIZATIONS

```
type en1 = enum { On, Off };
type en2 = enum { Up, Down };

node clocks (x: int) returns (y: int)
var c: en1 clock; d: en2 clock; b1, b2, b3, z: int; c1, c2: bool
let
  c1 = (x >= 0);
  d = if c1 then Up else Down;
  c2 = (x = 0) when Up(d);
  c = if c2 then Off else On;
  b2 = 2 when Off(c);
  b1 = 1 when On(c);
  z = merge c (On -> b1) (Off -> b2);
  b3 = 3 when Down(d);
  y = merge d (Up -> z) (Down -> b3);
tel
```

Optimizations

```
type en1 = enum { On, Off };
type en2 = enum { Up, Down };

node clocks (x: int) returns (y: int)
var c: en1 clock; d: en2 clock; b1, b2, b3, z: int; c1, c2: bool
let
  c1 = (x >= 0);
  d = if c1 then Up else Down;
  c2 = (x = 0) when Up(d);
  c = if c2 then Off else On;
  b2 = 2 when Off(c);
  b1 = 1 when On(c);
  z = merge c (On -> b1) (Off -> b2);
  b3 = 3 when Down(d);
  y = merge d (Up -> z) (Down -> b3);
tel
```

Optimizations

```
type en1 = enum { On, Off };
type en2 = enum { Up, Down };

node clocks (x: int) returns (y: int)
var c: en1 clock; d: en2 clock; b1, b2, b3, z: int; c1, c2: bool
let
  c1 = (x >= 0);
  d = if c1 then Up else Down;
  c2 = (x = 0) when Up(d);
  c = if c2 then Off else On;
  b2 = 2 when Off(c);
  b1 = 1 when On(c);
  z = merge c (On -> b1) (Off -> b2);
  b3 = 3 when Down(d);
  y = merge d (Up -> z) (Down -> b3);
tel
```

Optimizations

```
type en1 = enum { On, Off };
type en2 = enum { Up, Down };

node clocks (x: int) returns (y: int)
var c: en1 clock; d: en2 clock; b1, b2, b3, z: int; c1, c2: bool
let
  c1 = (x >= 0);
  d = if c1 then Up else Down;
  c2 = (x = 0) when Up(d);
  c = if c2 then Off else On;
  b2 = 2 when Off(c);
  b1 = 1 when On(c);
  z = merge c (On -> b1) (Off -> b2);
  b3 = 3 when Down(d);
  y = merge d (Up -> z) (Down -> b3);
tel
```

Optimizations

```
step (x: int) => (y: int) {
  var c: en1; d: en2; b1, b2, b3, z: int; c1, c2: bool;
  c1 := x >= 0;
  //@ clocks_tr1(x, c1)
  if (c1) { d := Up } else { d := Down }
  //@ clocks_tr2(x, d)
  case (d) { Up: c2 := x = 0 }
  //@ clocks_tr3(x, d, c2)
  case (d) { Up: if (c2) { c := Off } else { c := On } }
  //@ clocks_tr4(x, d, c)
  case (d) { Up: case (c) { Off: b2 := 2 } }
  //@ clocks_tr5(x, d, c, b2)
  case (d) { Up: case (c) { On: b1 := 1 } }
  //@ clocks_tr6(x, d, c, b1, b2)
  case (d) { Up: case (c) { On: z := b1 ; Off: z := b2 } }
  //@ clocks_tr7(x, d, z)
  case (d) { Down: b3 := 3 }
  //@ clocks_tr8(x, d, b3, z)
  case (d) { Up: y := z ; Down: y := b3 }
  //@ clocks_tr9(x, y)
}
```

CONDITIONALS FUSION

```
step (x: int) => (y: int) {
  var c: en1; d: en2; b1, b2, b3, z: int; c1, c2: bool;
  c1 := x >= 0;
  //@ clocks_tr1(x, c1)
  if (c1) { d := Up } else { d := Down }
  //@ clocks_tr2(x, d)
  case (d) { Up: c2 := x = 0 }
  //@ clocks_tr3(x, d, c2)
  case (d) { Up: if (c2) { c := Off } else { c := On } }
  //@ clocks_tr4(x, d, c)
  case (d) { Up: case (c) { Off: b2 := 2 } }
  //@ clocks_tr5(x, d, c, b2)
  case (d) { Up: case (c) { On: b1 := 1 } }
  //@ clocks_tr6(x, d, c, b1, b2)
  case (d) { Up: case (c) { On: z := b1 ; Off: z := b2 } }
  //@ clocks_tr7(x, d, z)
  case (d) { Down: b3 := 3 }
  //@ clocks_tr8(x, d, b3, z)
  case (d) { Up: y := z ; Down: y := b3 }
  //@ clocks_tr9(x, y)
}
```

CONDITIONALS FUSION

```
step (x: int) => (y: int) {
  var c: en1; d: en2; b1, b2, b3, z: int; c1, c2: bool;
  c1 := x >= 0;
  //@ clocks_tr1(x, c1)
  if (c1) { d := Up } else { d := Down }
  //@ clocks_tr2(x, d)
  case (d) {
    Up:
      c2 := x = 0;
      if (c2) { c := Off } else { c := On }
      case (c) {
        On:
          b1 := 1;
          z := b1;
        Off:
          b2 := 2;
          z := b2;
      }
      y := z;
    Down:
      b3 := 3;
      y := b3;
  }
  //@ clocks_tr3(x, d, c2)
  //@ clocks_tr4(x, d, c)
  //@ clocks_tr5(x, d, c, b2)
  //@ clocks_tr6(x, d, c, b1, b2)
  //@ clocks_tr7(x, d, z)
  //@ clocks_tr8(x, d, b3, z)
  //@ clocks_tr9(x, y)
}
```

Variable inlining

```
step (x: int) => (y: int) {
  var c: en1; d: en2; b1, b2, b3, z: int; c1, c2: bool;
  c1 := x >= 0;
  //@ clocks_tr1(x, c1)
  if (c1) { d := Up } else { d := Down }
  //@ clocks_tr2(x, d)
  case (d) {
    Up:
      c2 := x = 0;
      if (c2) { c := Off } else { c := On }
      case (c) {
        On:
          b1 := 1;
          z := b1;
        Off:
          b2 := 2;
          z := b2;
      }
      y := z;
    Down:
      b3 := 3;
      y := b3;
  }
  //@ clocks_tr3(x, d, c2)
  //@ clocks_tr4(x, d, c)
  //@ clocks_tr5(x, d, c, b2)
  //@ clocks_tr6(x, d, c, b1, b2)
  //@ clocks_tr7(x, d, z)
  //@ clocks_tr8(x, d, b3, z)
  //@ clocks_tr9(x, y)
}
```

VARIABLE INLINING

```
step (x: int) => (y: int) {
  var c: en1; d: en2; z: int; //@ ghost b1, b2, b3: int; c1, c2: bool
  //@ c1 := x >= 0
  //@ clocks_tr1(x, c1)
  if (x >= 0) { d := Up } else { d := Down }
  //@ clocks_tr2(x, d)
  case (d) {
    Up:
      //@ c2 := x = 0
      if (x = 0) { c := Off } else { c := On }
      case (c) {
        On:
          //@ b1 := 1
          z := 1;
        Off:
          //@ b2 := 2
          z := 2;
      }
      y := z;
    Down:
      //@ b3 := 3
      y := 3;
  }
  //@ clocks_tr3(x, d, c2)
  //@ clocks_tr4(x, d, c)
  //@ clocks_tr5(x, d, c, b2)
  //@ clocks_tr6(x, d, c, b1, b2)
  //@ clocks_tr7(x, d, z)
  //@ clocks_tr8(x, d, b3, z)
  //@ clocks_tr9(x, y)
}
```

Variable recycling

```
step (x: int) => (y: int) {
  var c: en1; d: en2; z: int; //@ ghost b1, b2, b3: int; c1, c2: bool
  //@ c1 := x >= 0
  //@ clocks_tr1(x, c1)
  if (x >= 0) { d := Up } else { d := Down }
  //@ clocks_tr2(x, d)
  case (d) {
    Up:
      //@ c2 := x = 0
      if (x = 0) { c := Off } else { c := On }
      case (c) {
        On:
          //@ b1 := 1
          z := 1;
        Off:
          //@ b2 := 2
          z := 2;
      }
      y := z;
    Down:
      //@ b3 := 3
      y := 3;
  }
  //@ clocks_tr3(x, d, c2)
  //@ clocks_tr4(x, d, c)
  //@ clocks_tr5(x, d, c, b2)
  //@ clocks_tr6(x, d, c, b1, b2)
  //@ clocks_tr7(x, d, z)
  //@ clocks_tr8(x, d, b3, z)
  //@ clocks_tr9(x, y)
}
```

Variable recycling

```
step (x: int) => (y: int) {
  var c: en1; d: en2; //@ ghost b1, b2, b3, z: int; c1, c2: bool
  //@ c1 := x >= 0
  //@ clocks_tr1(x, c1)
  if (x >= 0) { d := Up } else { d := Down }
  //@ clocks_tr2(x, d)
  case (d) {
    Up:
      //@ c2 := x = 0
      if (x = 0) { c := Off } else { c := On }
      case (c) {
        On:
          //@ b1 := 1
          y := 1;
          //@ z := y
        Off:
          //@ b2 := 2
          y := 2;
          //@ z := y
      }
    Down:
      //@ b3 := 3
      y := 3;
  }
  //@ clocks_tr3(x, d, c2)
  //@ clocks_tr4(x, d, c)
  //@ clocks_tr5(x, d, c, b2)
  //@ clocks_tr6(x, d, c, b1, b2)
  //@ clocks_tr7(x, d, z)
  //@ clocks_tr8(x, d, b3, z)
  //@ clocks_tr9(x, y)
}
```

Enum elimination

```
step (x: int) => (y: int) {
  var c: en1; d: en2; //@ ghost b1, b2, b3, z: int; c1, c2: bool
  //@ c1 := x >= 0
  //@ clocks_tr1(x, c1)
  if (x >= 0) { d := Up } else { d := Down }
  //@ clocks_tr2(x, d)
  case (d) {
    Up:
      //@ c2 := x = 0
      if (x = 0) { c := Off } else { c := On }
      case (c) {
        On:
          //@ b1 := 1
          y := 1;
          //@ z := y
        Off:
          //@ b2 := 2
          y := 2;
          //@ z := y
      }
    Down:
      //@ b3 := 3
      y := 3;
  }
  //@ clocks_tr3(x, d, c2)
  //@ clocks_tr4(x, d, c)
  //@ clocks_tr5(x, d, c, b2)
  //@ clocks_tr6(x, d, c, b1, b2)
  //@ clocks_tr7(x, d, z)
  //@ clocks_tr8(x, d, b3, z)
  //@ clocks_tr9(x, y)
}
```

Enum elimination

```
step (x: int) => (y: int) {
  //@ ghost c: en1; d: en2; b1, b2, b3, z: int; c1, c2: bool
  //@ c1 := x >= 0
  //@ clocks_tr1(x, c1)
  if (x >= 0) {
    //@ d := Up
    //@ c2 := x = 0
    if (x = 0) {
      //@ c := Off
      //@ b2 := 2
      y := 2;
      //@ z := y
    } else {
      //@ c := On
      //@ b1 := 1
      y := 1;
      //@ z := y
    }
  } else {
    //@ d := Down
    //@ b3 := 3
    y := 3;
  }
  //@ clocks_tr2(x, d)
  //@ clocks_tr3(x, d, c2)
  //@ clocks_tr4(x, d, c)
  //@ clocks_tr5(x, d, c, b2)
  //@ clocks_tr6(x, d, c, b1, b2)
  //@ clocks_tr7(x, d, z)
  //@ clocks_tr8(x, d, b3, z)
  //@ clocks_tr9(x, y)
}
```

Conclusion

- Extension of a Lustre compiler to support Translation Validation
- High automatic verification success rate
- Aggressive validated optimizations

Perspectives

- Functional contracts from Lustre to C
- Floats, arrays, records, ...
- More optimizations