

EQUATION-DIRECTED AXIOMATIZATION OF LUSTRE SEMANTICS TO ENABLE OPTIMIZED CODE VALIDATION

EMSOFT

SEPTEMBER 18 2023

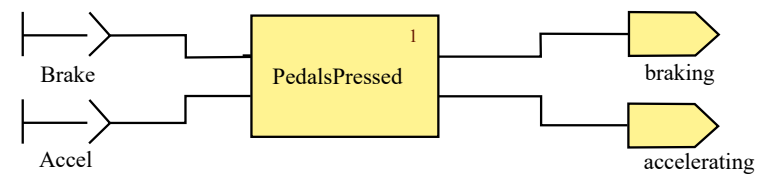
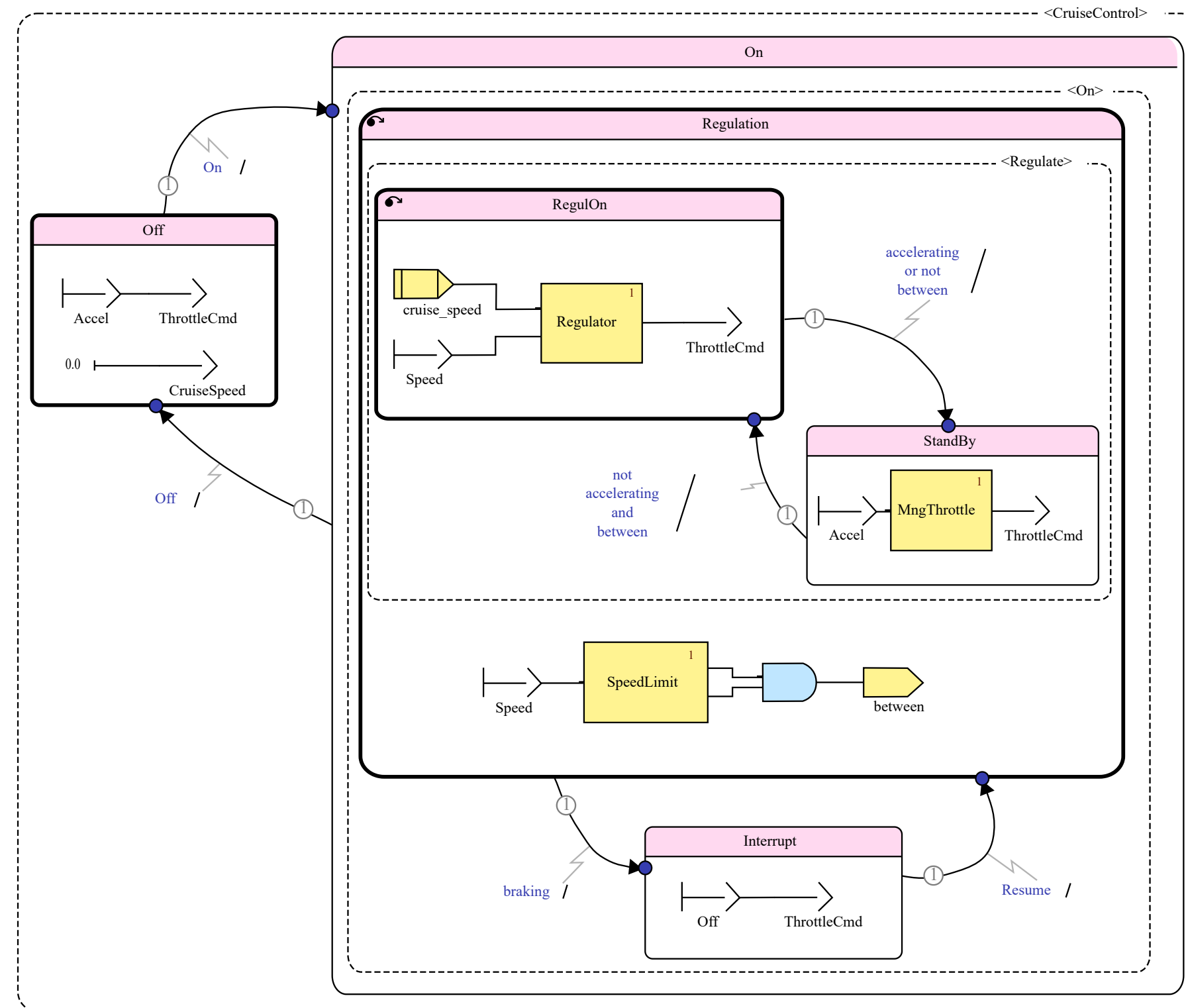
Lélio Brun • Christophe Garion • Pierre-Loïc Garoche • Xavier Thirioux



EMBEDDED SYSTEMS



MODEL-BASED DESIGN: SCADE



LUSTRE AND CERTIFIED COMPILATION

INDUSTRIAL QUALIFICATION

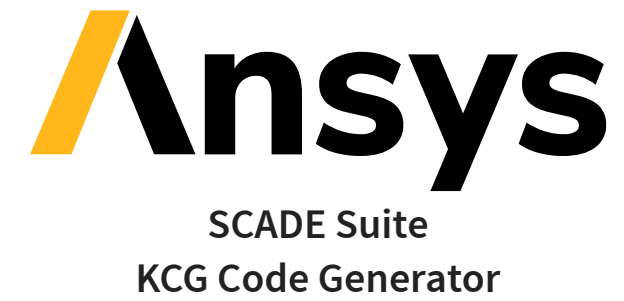


SCADE Suite
KCG Code Generator

DO-178C

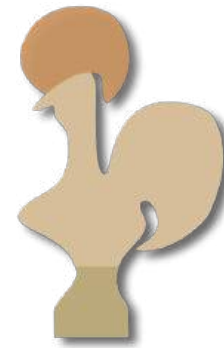
LUSTRE AND CERTIFIED COMPILATION

INDUSTRIAL QUALIFICATION



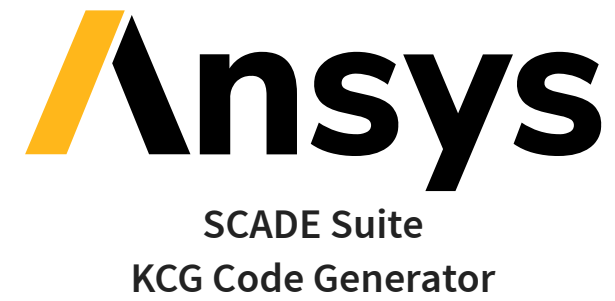
DO-178C

VERIFIED COMPILATION



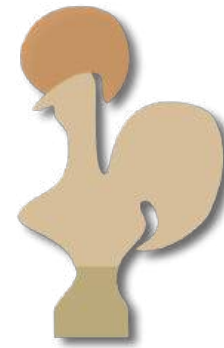
LUSTRE AND CERTIFIED COMPILATION

INDUSTRIAL QUALIFICATION



DO-178C

VERIFIED COMPILATION

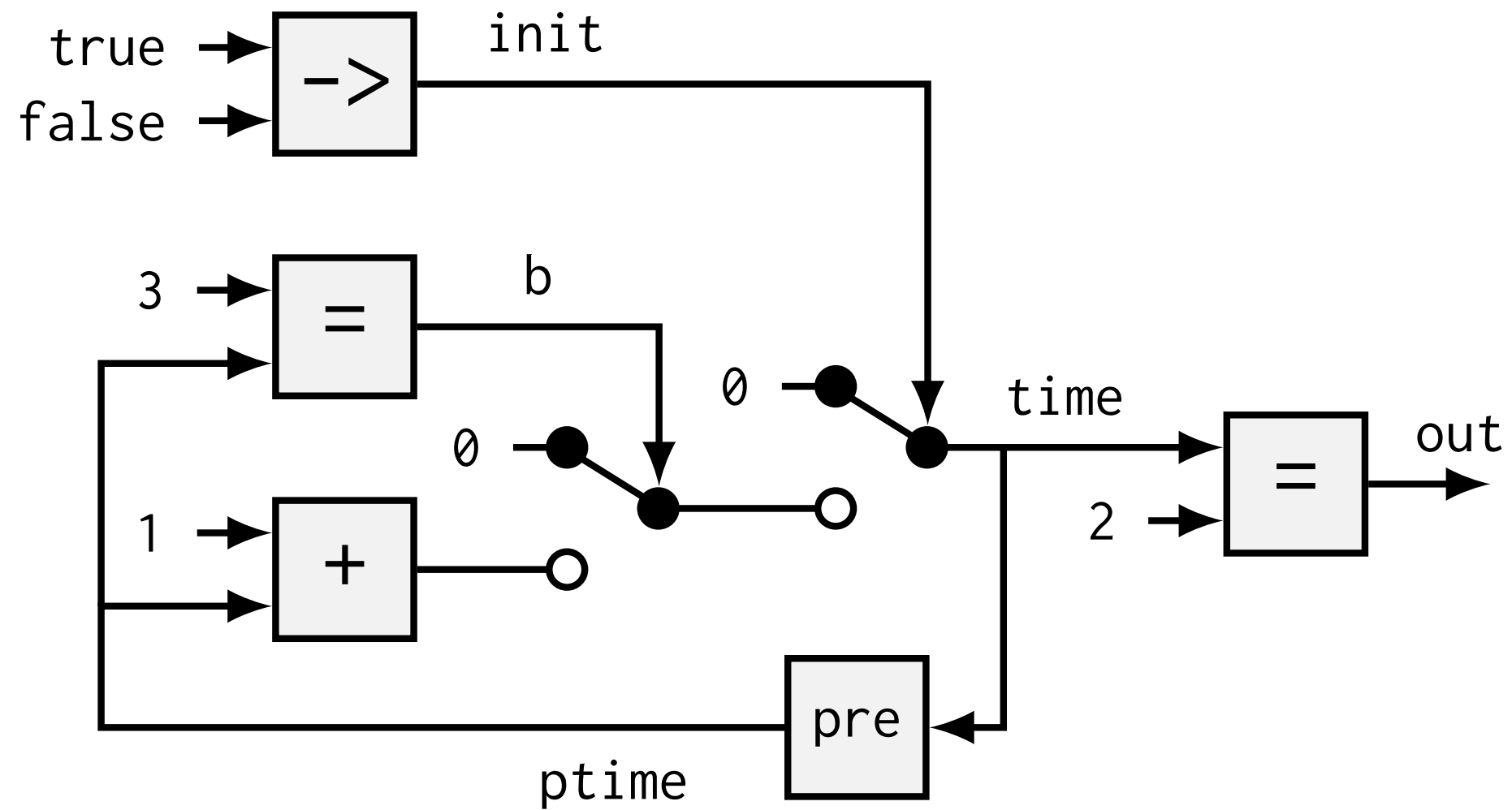


TRANSLATION VALIDATION



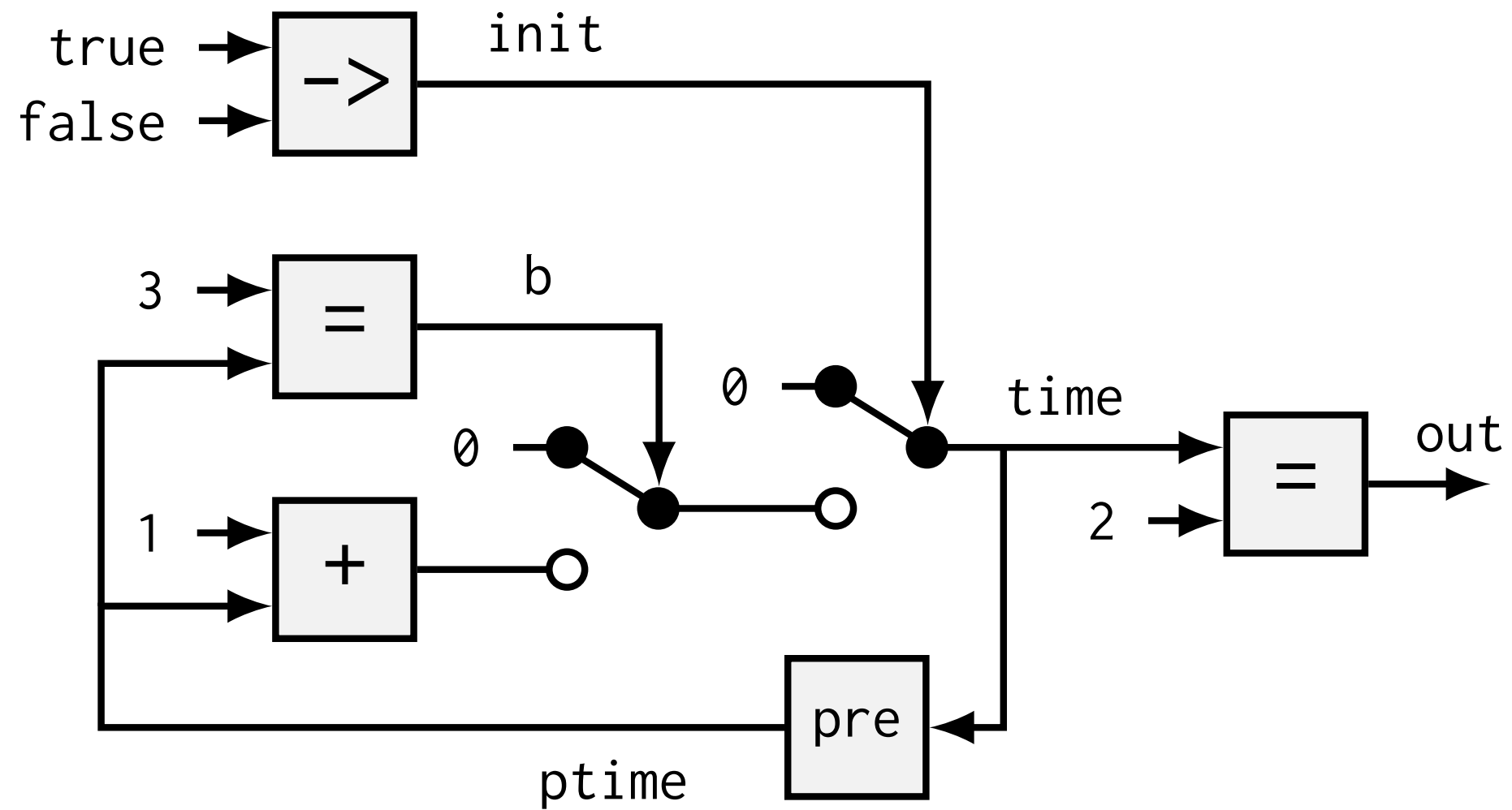
Software Analyzers

EXAMPLE



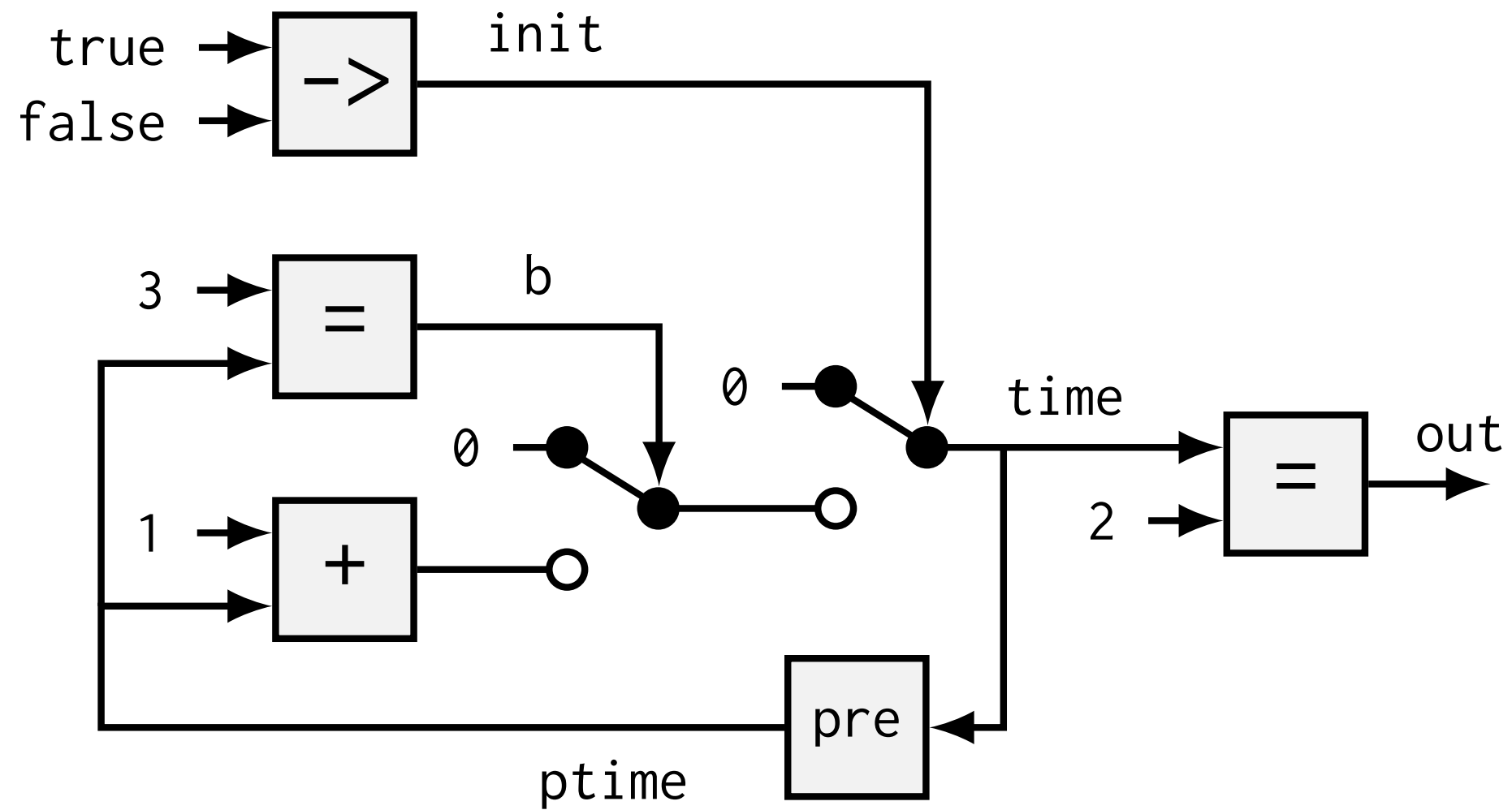
```
node count() returns (out: bool)
var time, ptime: int;
    init, b: bool;
let
  out = (time = 2);
  b = (ptime = 3);
  ptime = pre time;
  time = if init then 0
         else if b then 0
         else ptime + 1;
  init = true -> false;
tel
```


EXAMPLE



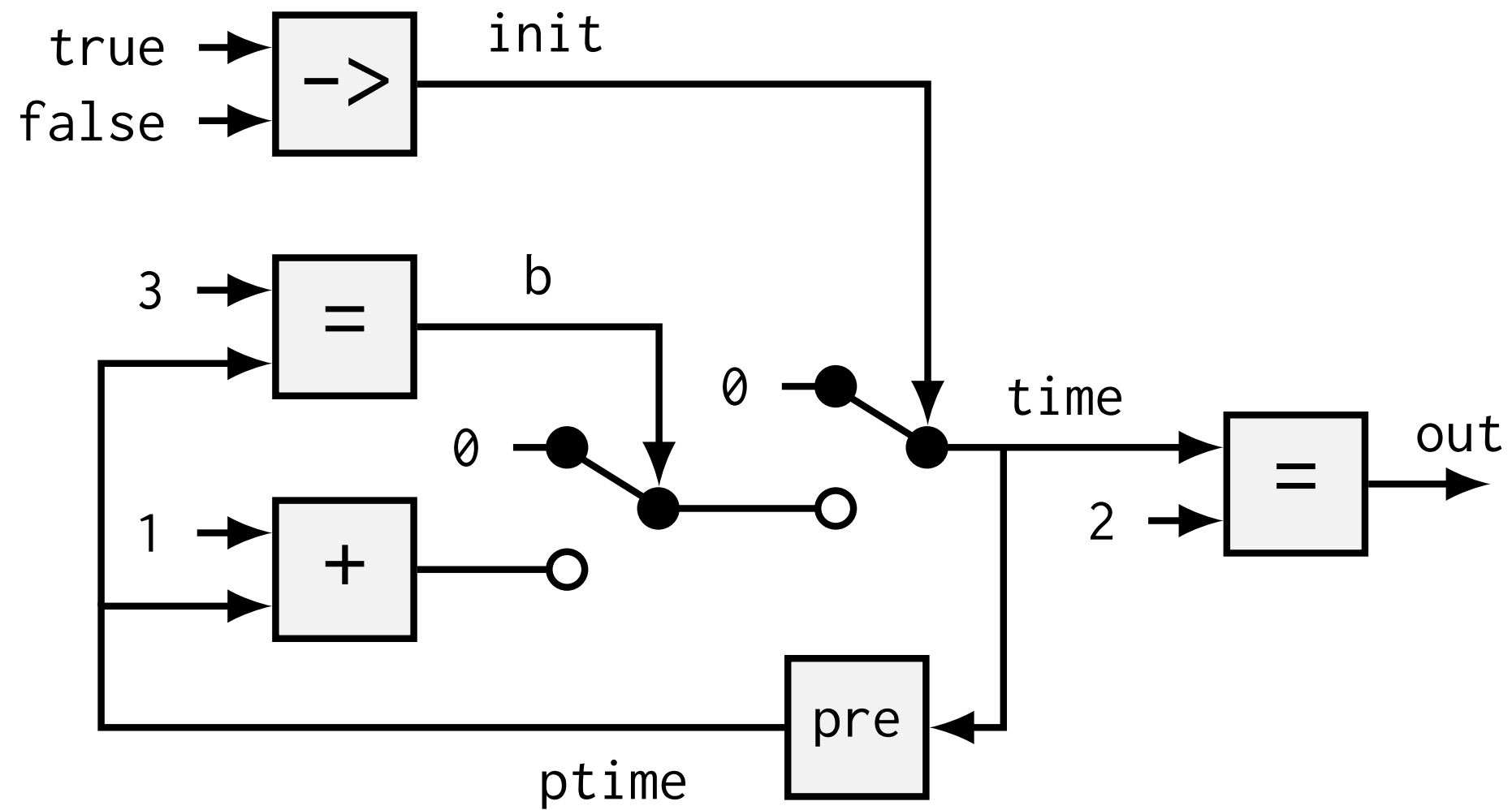
```
node count() returns (out: bool)
var time, ptime: int;
    init, b: bool;
let
  init = true -> false;
  b = (ptime = 3);
  time = if init then 0
         else if b then 0
         else ptime + 1;
  ptime = pre time;
  out = (time = 2);
tel
```

EXAMPLE



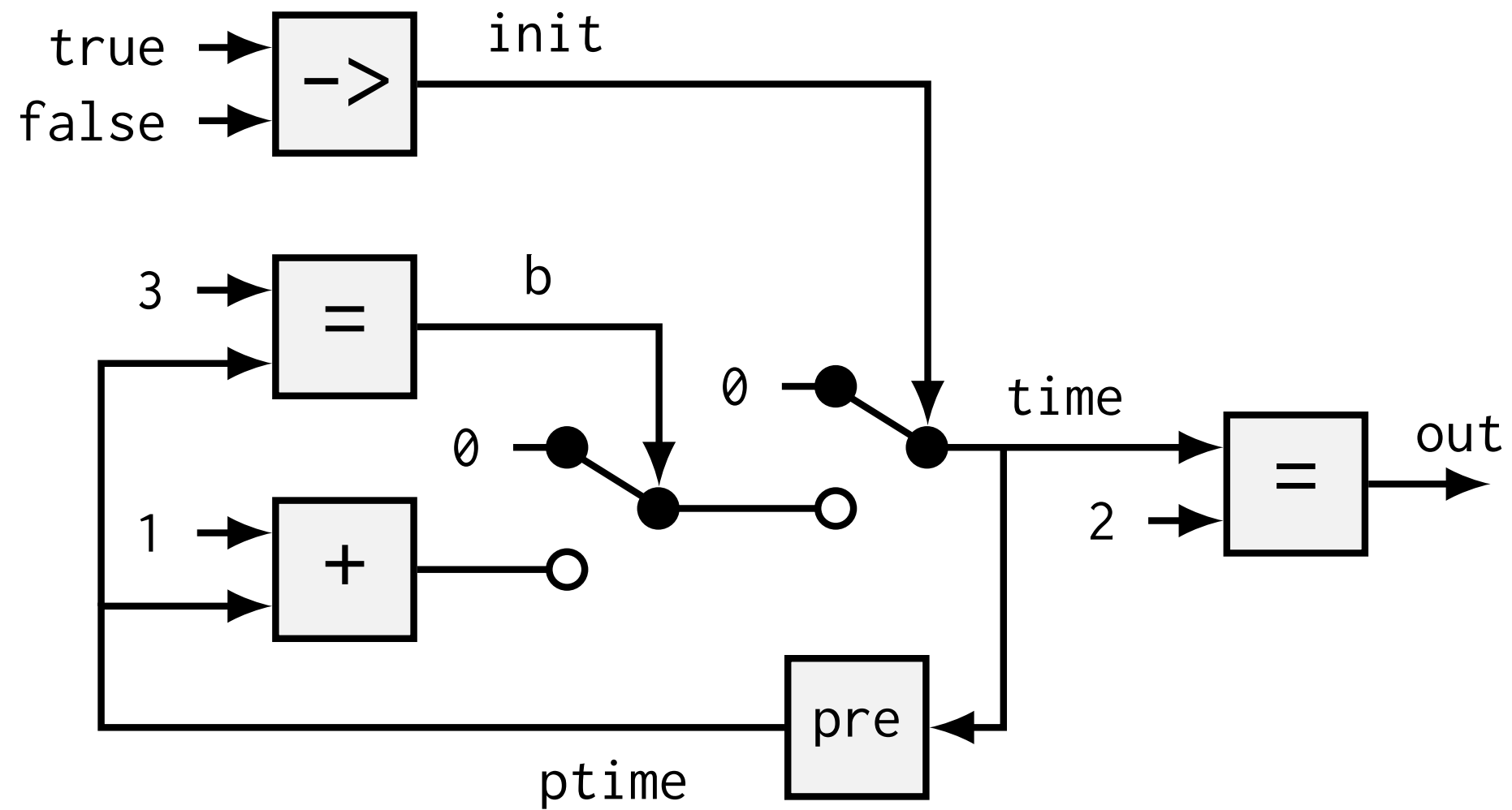
```
node count() returns (out: bool)
var time, ptime: int;
    init, b: bool;
let
    init = true -> false;
    b = (ptime = 3);
    time = if init then 0
           else if b then 0
           else ptime + 1;
    ptime = pre time;
    out = (time = 2);
tel
```


EXAMPLE



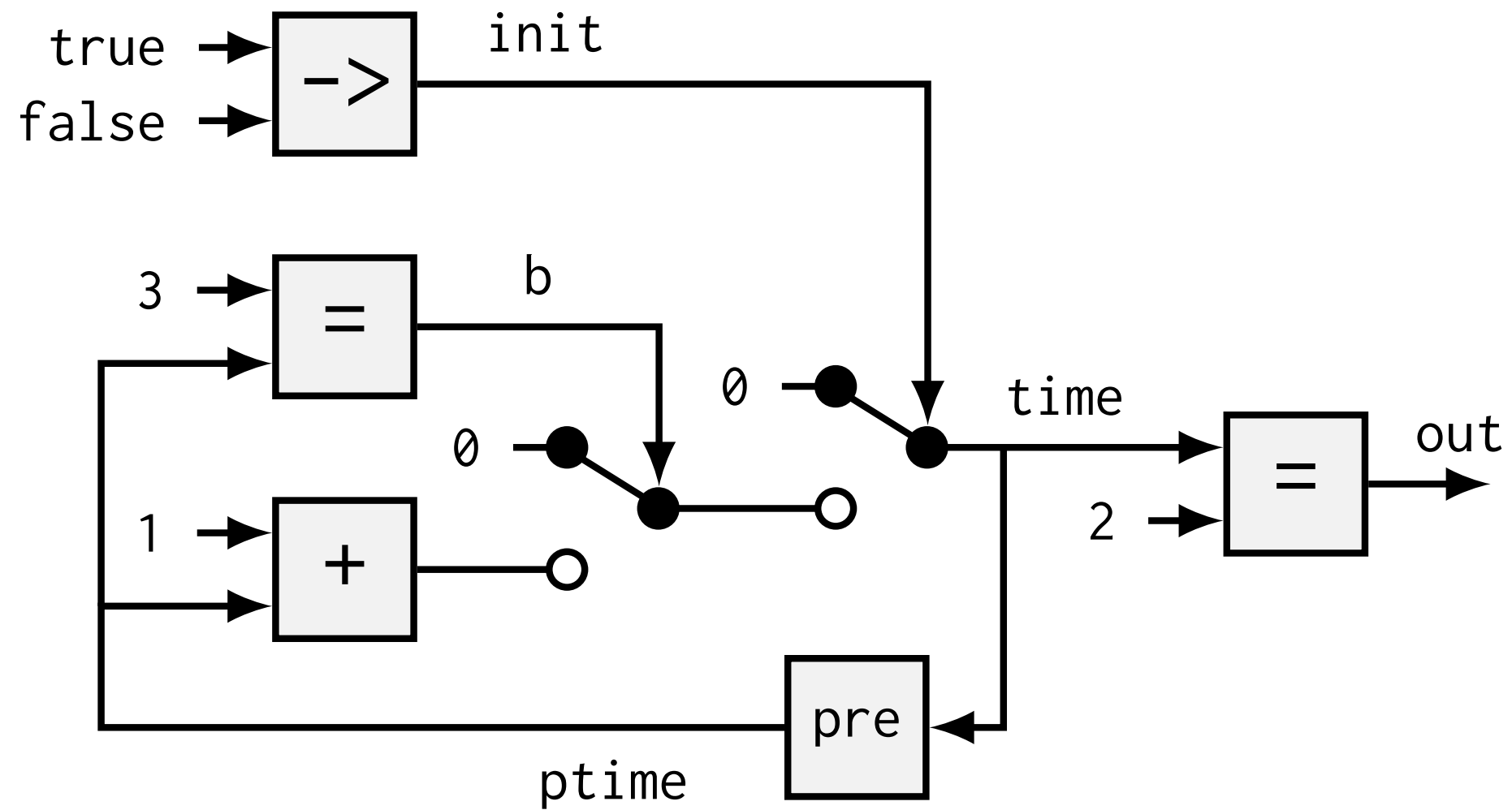
```
node count() returns (out: bool)
var time, ptime: int;
    init, b: bool;
let
    init = true -> false;
    b = (ptime = 3);
    time = if init then 0
           else if b then 0
           else ptime + 1;
    ptime = pre time;
    out = (time = 2);
tel
```

EXAMPLE



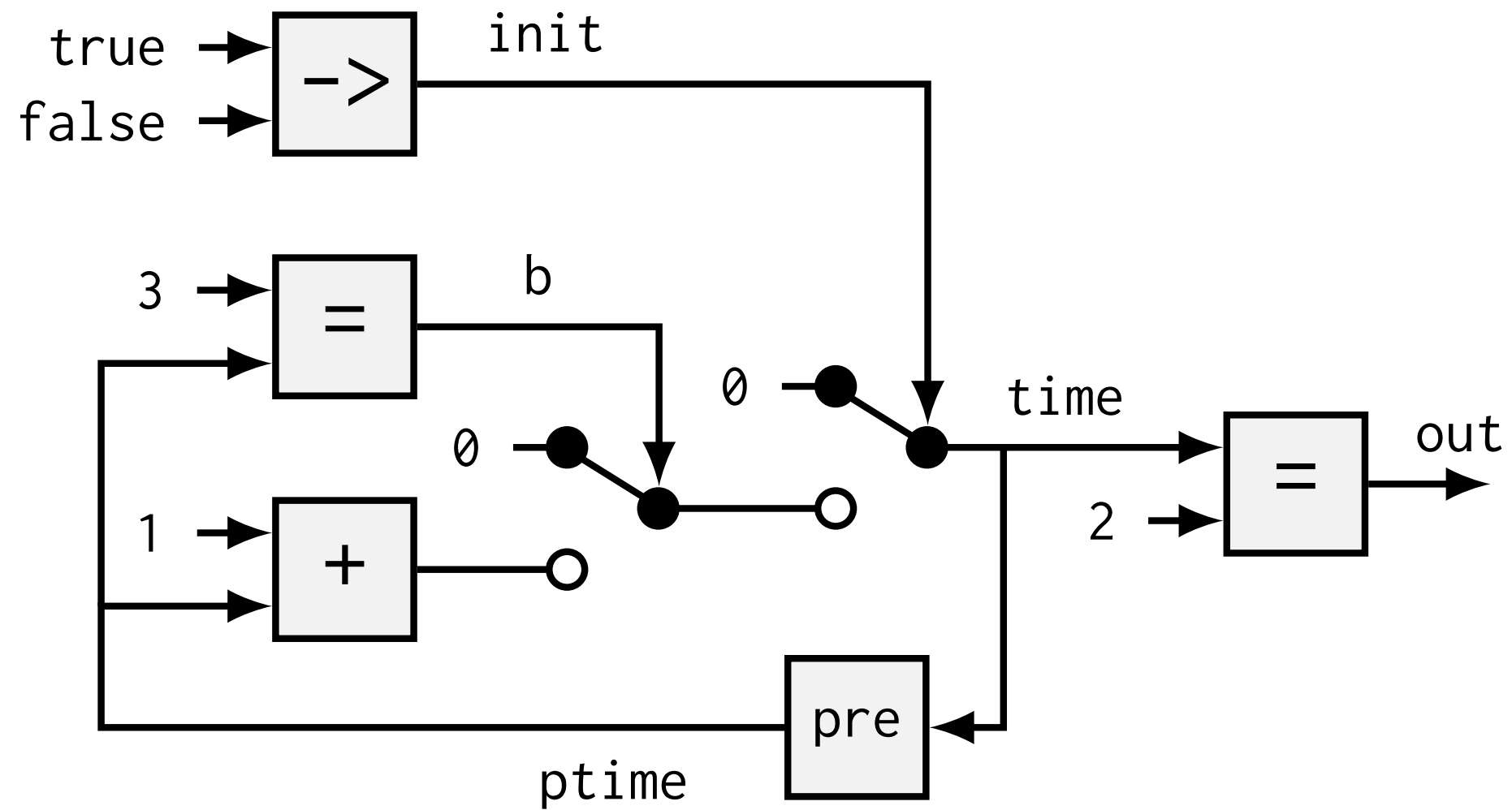
```
node count() returns (out: bool)
var time, ptime: int;
    init, b: bool;
let
    init = true -> false;
    b = (ptime = 3);
    time = if init then 0
           else if b then 0
           else ptime + 1;
    ptime = pre time;
    out = (time = 2);
tel
```

EXAMPLE



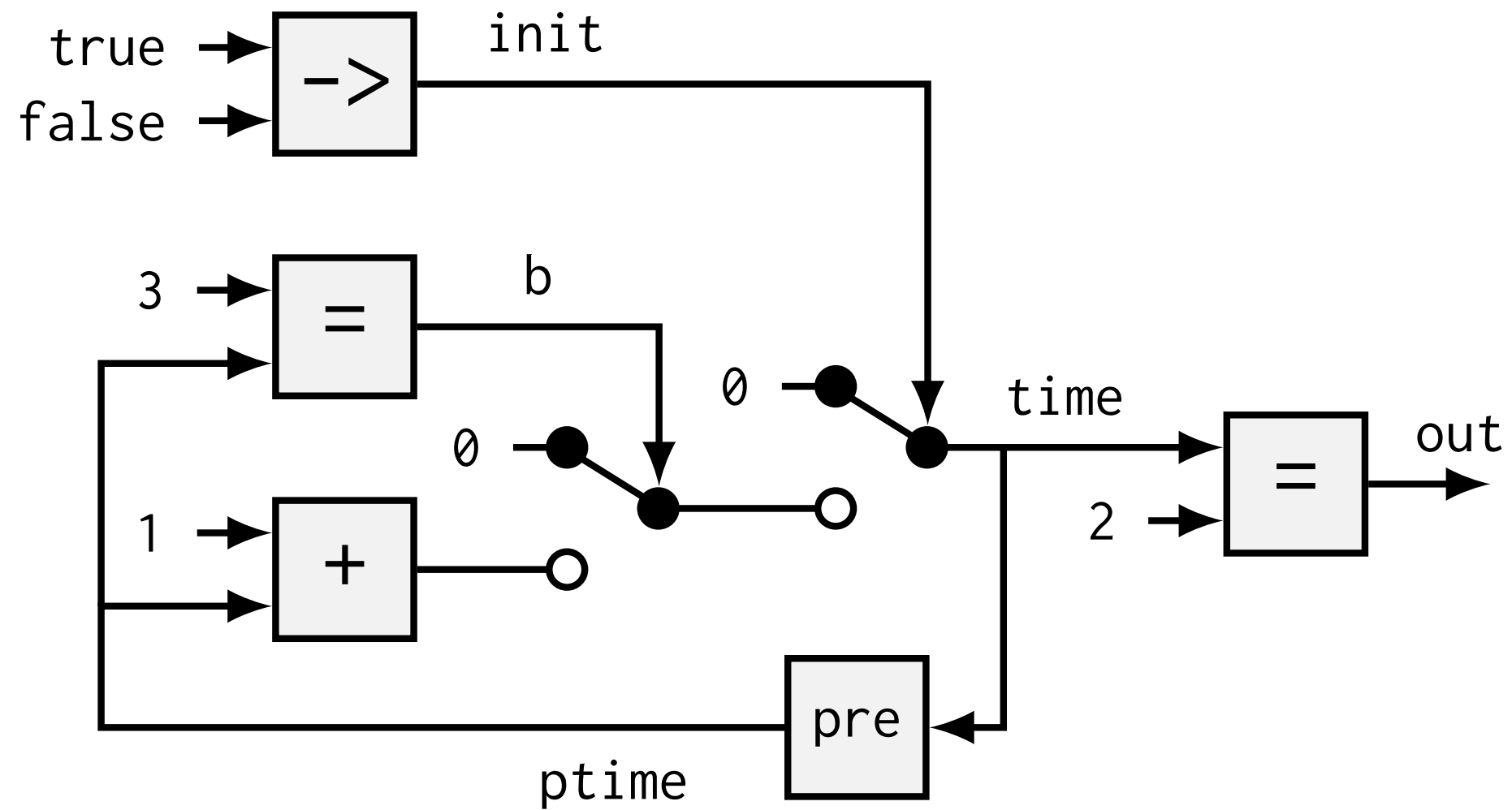
```
node count() returns (out: bool)
var time, ptime: int;
    init, b: bool;
let
    init = true -> false;
    b = (ptime = 3);
    time = if init then 0
           else if b then 0
           else ptime + 1;
    ptime = pre time;
    out = (time = 2);
tel
```

EXAMPLE

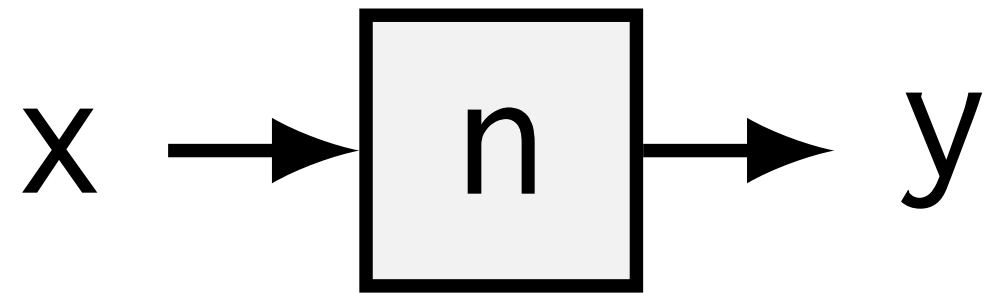
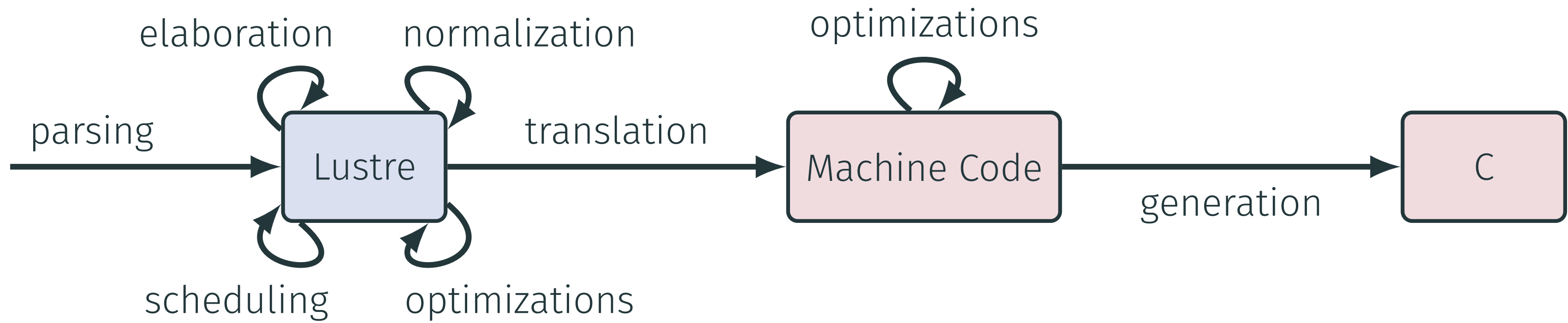


```
node count() returns (out: bool)
var time, ptime: int;
    init, b: bool;
let
    init = true -> false;
    b = (ptime = 3);
    time = if init then 0
           else if b then 0
           else ptime + 1;
    ptime = pre time;
    out = (time = 2);
tel
```

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
```

```

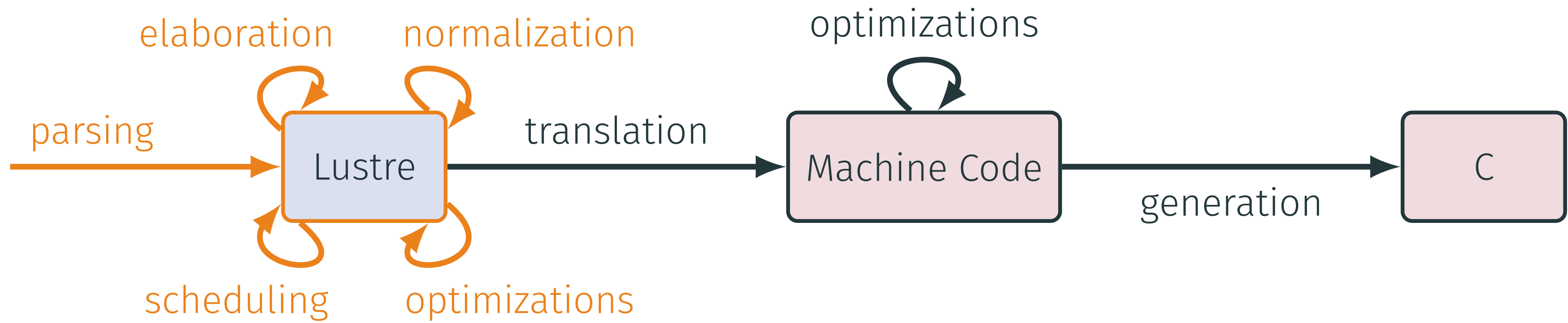
// state initialization
n_reset(&self);

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

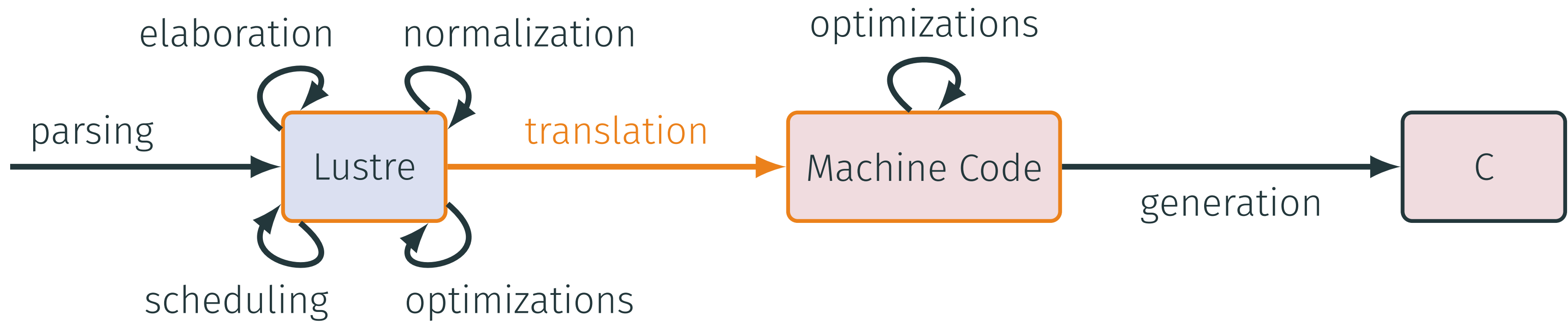
    // execution step
    n_step(x, &y, &self);

    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
```

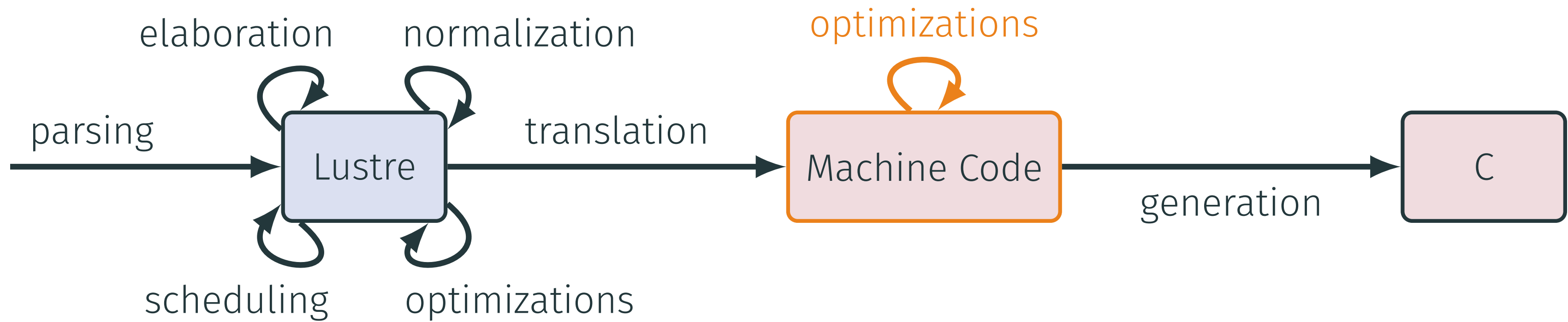


```

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);
  }
}
  
```

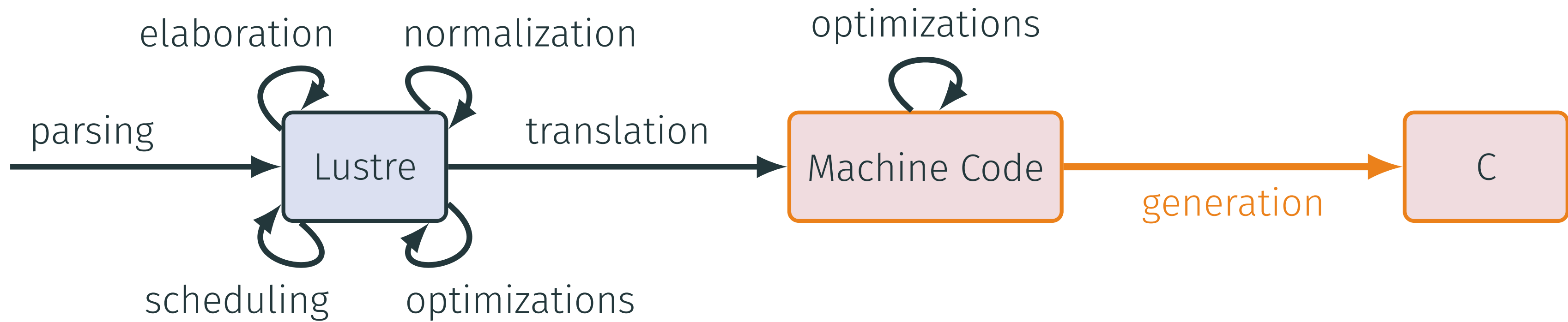


```

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);
  }
}
  
```



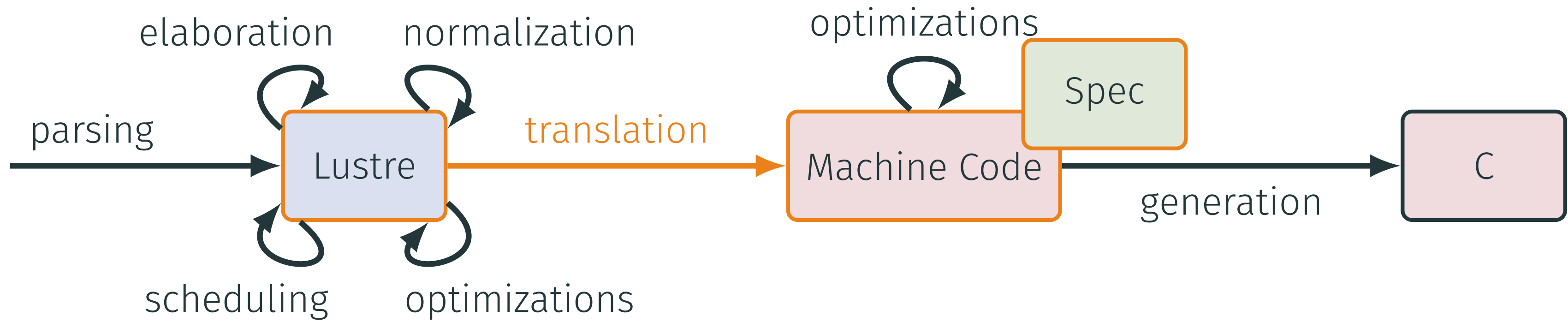
```

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;
}

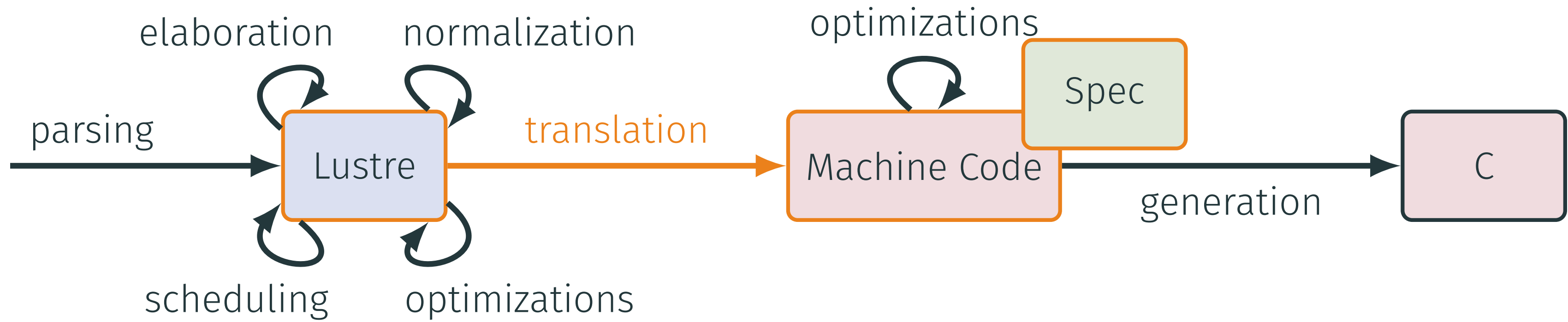
```



```

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
  
```

$$count_tr(S, out, S') \triangleq count_tr_5(S, out, S')$$



```

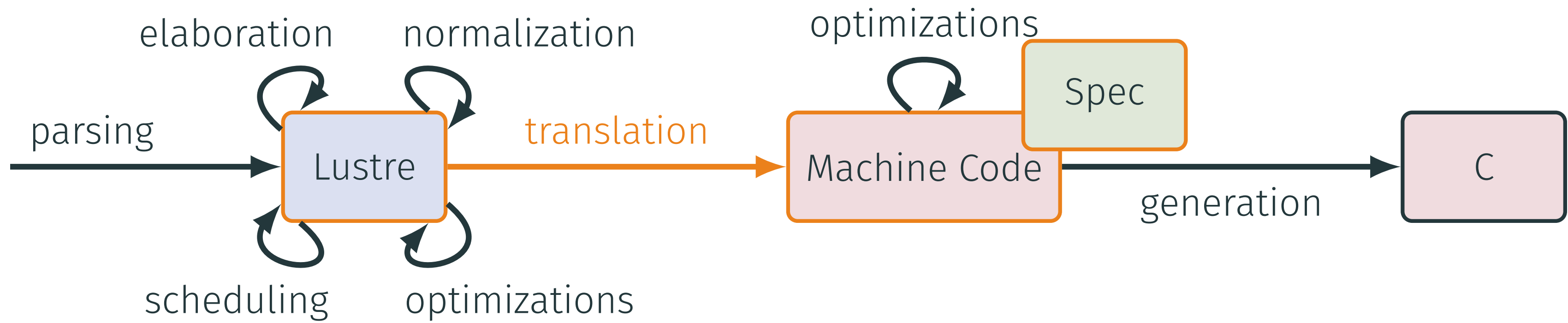
node count() returns (out: bool)
var time, ptime: int;
    init, b: bool;
let
  init = true -> false;
  b = (ptime = 3);
  time = if init then 0
         else if b then 0
         else ptime + 1;
  ptime = pre time;
  out = (time = 2);
tel
  
```

$$count_tr(S, out, S') \triangleq$$

$$\exists time,$$

$$count_tr_4(S, time, S')$$

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



```

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

```

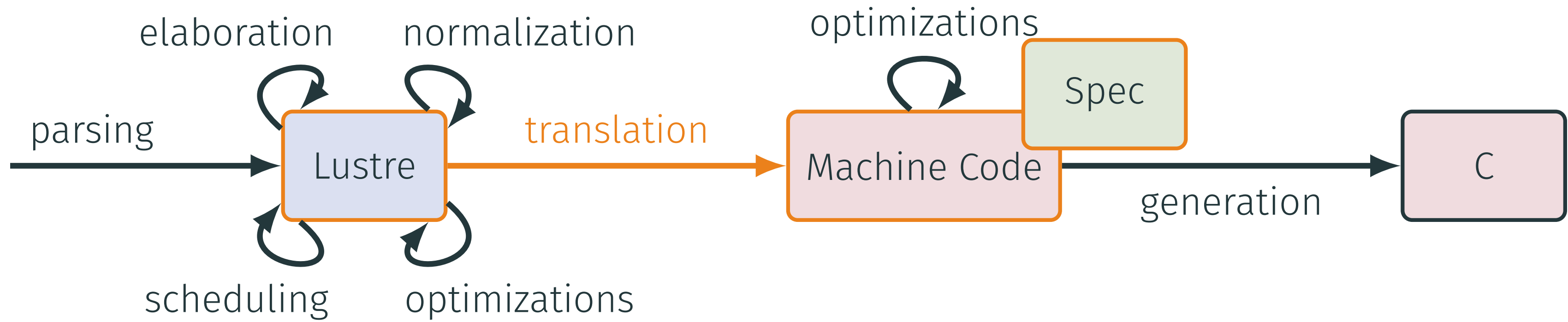
$$count_tr(S, out, S') \triangleq$$

$$\exists time,$$

$$count_tr_3(S, time, S')$$

$$\wedge S'(ptime) = time$$

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



```

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

```

$$\text{count_tr}(S, \text{out}, S') \triangleq$$

$$\exists \text{time},$$

$$\exists b, \text{init},$$

$$\text{count_tr}_2(S, \text{time}, S')$$

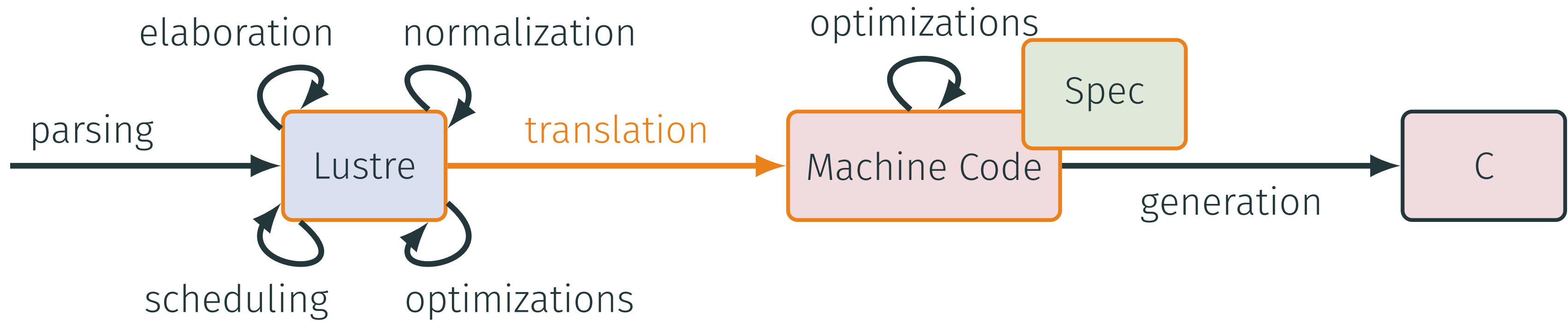
$$\wedge \text{init} \implies \text{time} = 0$$

$$\wedge (\neg \text{init} \wedge b) \implies \text{time} = 0$$

$$\wedge (\neg \text{init} \wedge \neg b) \implies \text{time} = S(\text{ptime}) + 1$$

$$\wedge S'(\text{ptime}) = \text{time}$$

$$\wedge \text{out} = (\text{time} = 2)$$



```

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

```

$$\text{count_tr}(S, \text{out}, S') \triangleq$$

$$\exists \text{time},$$

$$\exists b, \text{init},$$

$$\text{count_tr}_1(S, \text{time}, S')$$

$$\wedge b = (S(\text{ptime}) = 3)$$

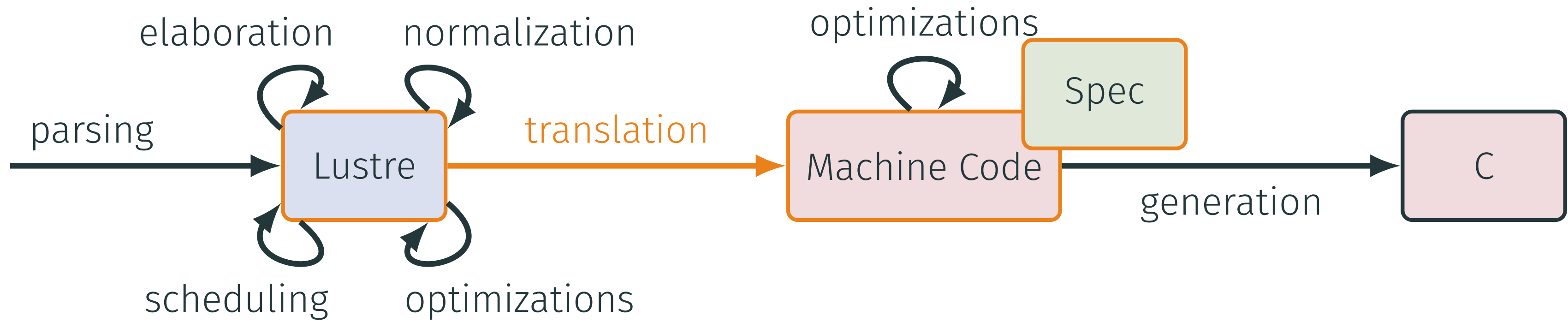
$$\wedge \text{init} \implies \text{time} = 0$$

$$\wedge (\neg \text{init} \wedge b) \implies \text{time} = 0$$

$$\wedge (\neg \text{init} \wedge \neg b) \implies \text{time} = S(\text{ptime}) + 1$$

$$\wedge S'(\text{ptime}) = \text{time}$$

$$\wedge \text{out} = (\text{time} = 2)$$



```

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

```

$$\text{count_tr}(S, \text{out}, S') \triangleq$$

$$\exists \text{time},$$

$$\exists b, \text{init},$$

$$\text{arrow_tr}(S[a], \text{init}, S'[a])$$

$$\wedge b = (S(\text{ptime}) = 3)$$

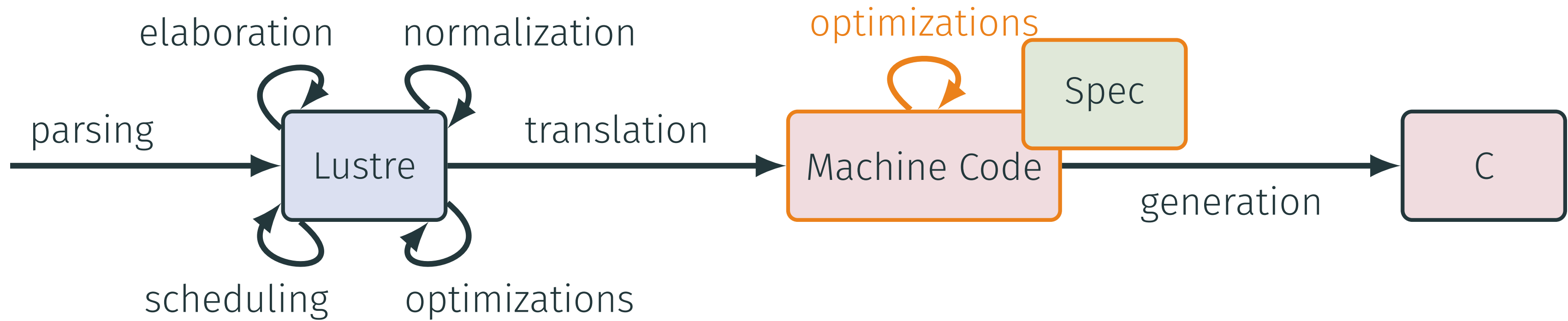
$$\wedge \text{init} \implies \text{time} = 0$$

$$\wedge (\neg \text{init} \wedge b) \implies \text{time} = 0$$

$$\wedge (\neg \text{init} \wedge \neg b) \implies \text{time} = S(\text{ptime}) + 1$$

$$\wedge S'(\text{ptime}) = \text{time}$$

$$\wedge \text{out} = (\text{time} = 2)$$



```

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

```

$$\text{count_tr}(S, \text{out}, S') \triangleq$$

$$\exists \text{time},$$

$$\exists b, \text{init},$$

$$\text{arrow_tr}(S[a], \text{init}, S'[a])$$

$$\wedge b = (S(\text{ptime}) = 3)$$

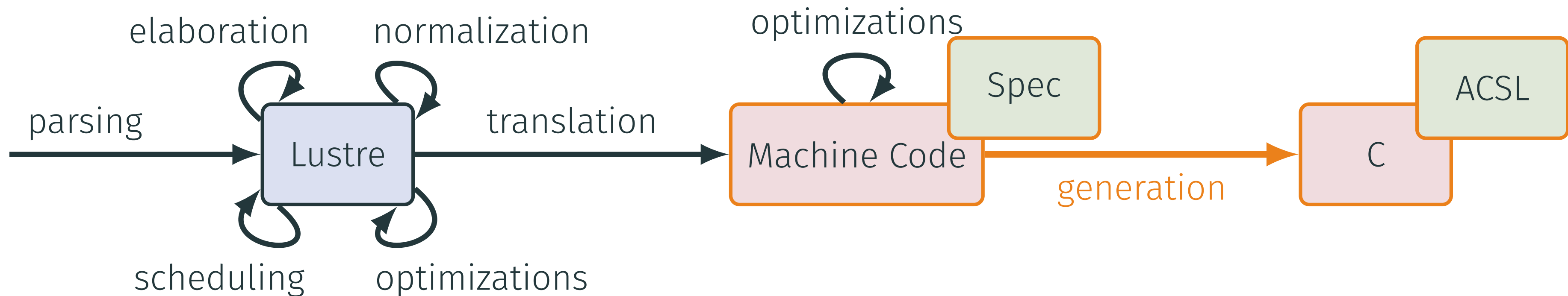
$$\wedge \text{init} \implies \text{time} = 0$$

$$\wedge (\neg \text{init} \wedge b) \implies \text{time} = 0$$

$$\wedge (\neg \text{init} \wedge \neg b) \implies \text{time} = S(\text{ptime}) + 1$$

$$\wedge S'(\text{ptime}) = \text{time}$$

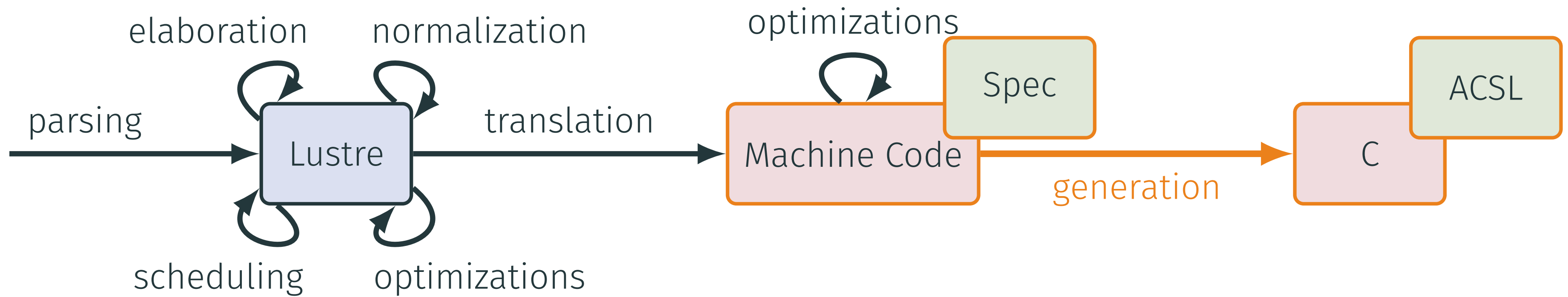
$$\wedge \text{out} = (\text{time} = 2)$$



```

/*@ 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);
}

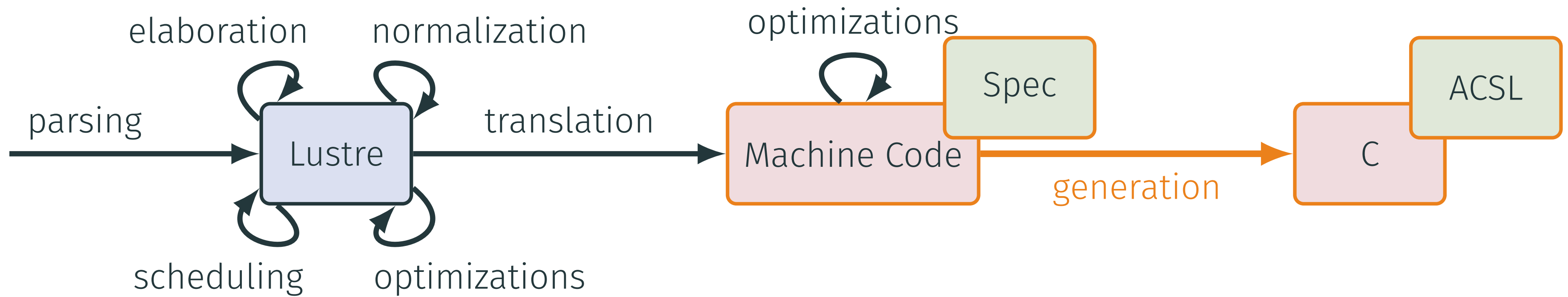
```



```

/*@ 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);
}

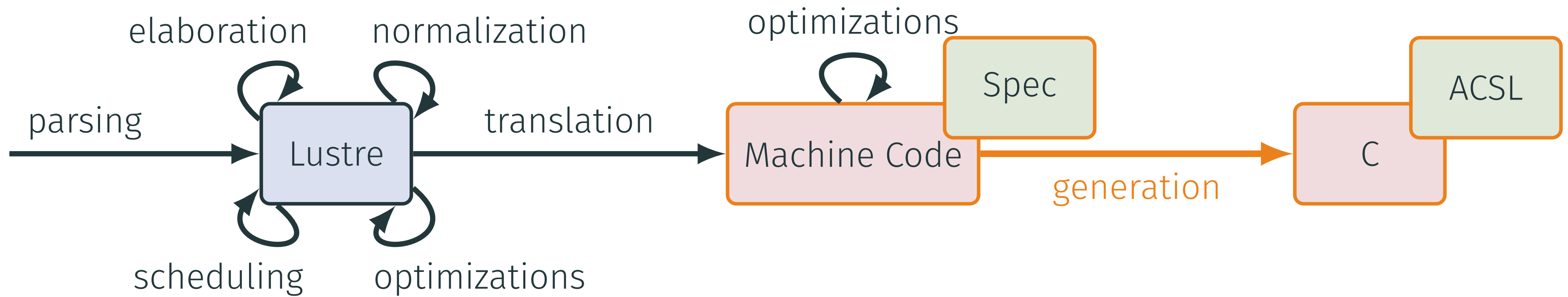
```



```

/*@ 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);
}

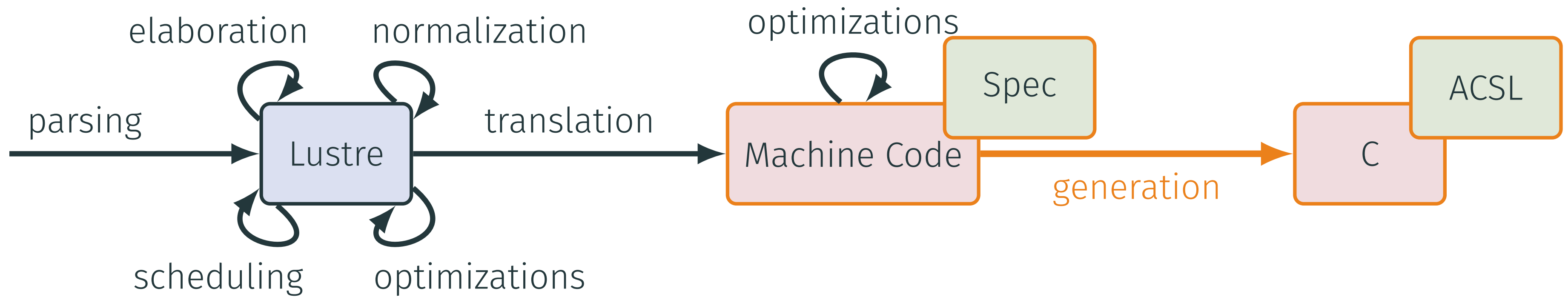
```



```

/*@ 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);
}

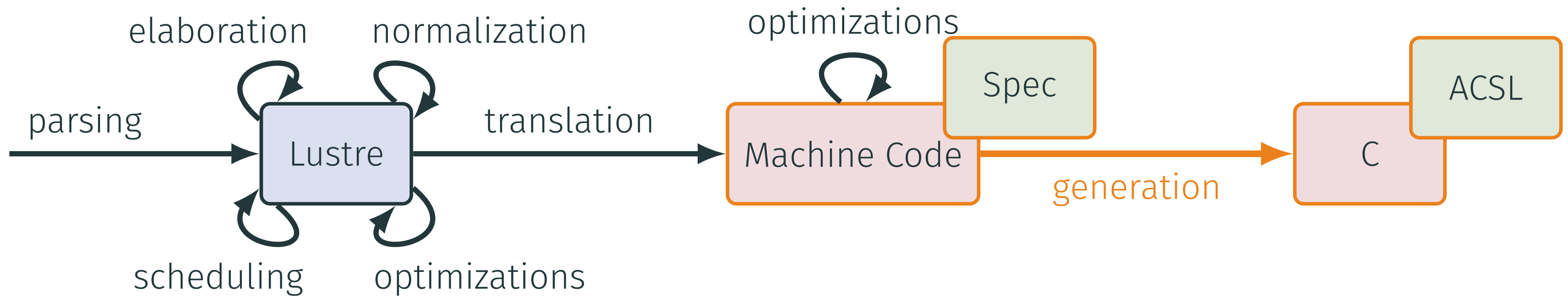
```

```

/*@ 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);
}

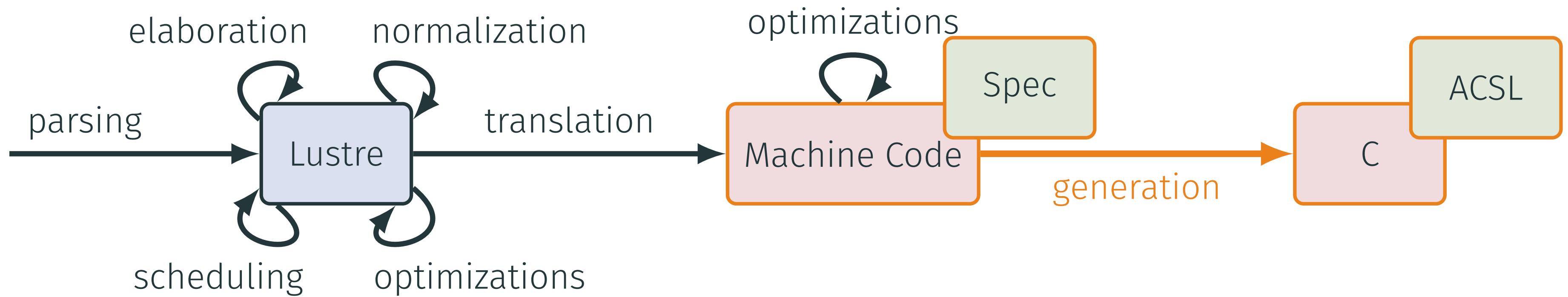
```



```

/*@ 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);
}

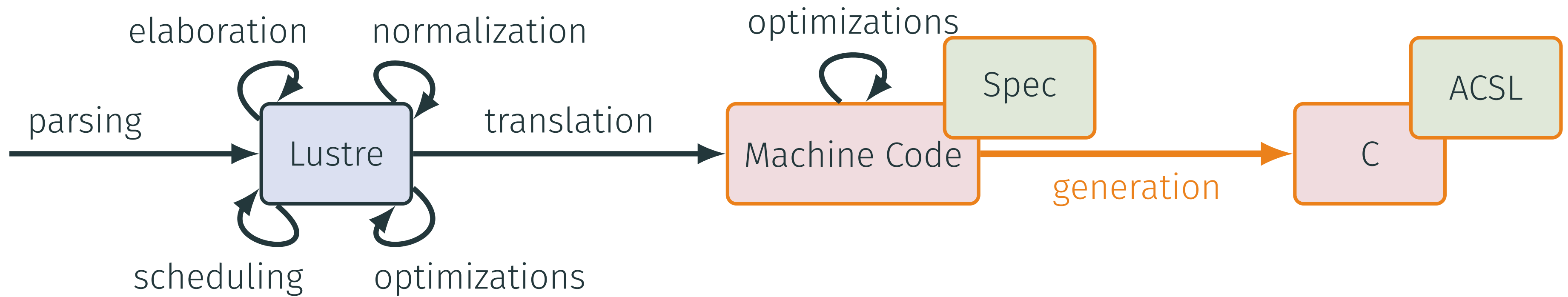
```



```

/*@ 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

Name

- ▶ _build/install/default/lib/lustrec/include/arrow_spec.h
- ▶ _opam/share/frama-c/share/lib/assert.h
- ▼ count.c
 - count_clear_reset
 - count_step**
 - count.h
 - count_memory.h
 - ▶ count_spec.h

WP

30 - + timeout

8 - + process

Model... Provers... Update ▶

Slicing

Enable 1 - + Verbose

Libraries 2 - + Level

Occurrence

Current var: None

Enable

Follow focus

Read Write

Metrics

▶ Launch

Impact

Enable

Slicing after impact

Follow focus

Eva

Run

-1 - + precision (meta-option)

0 - + slevel

main main

Show list of red alarms

```

1  /*@ requires count_valid(self);
2  requires \separated(self, mem, self->ni_0, out);
3  requires count_pack(*mem, self);
4  ensures count_pack(*old(mem), \old(self));
5  ensures count_transition(\old(*mem), \old(x), *\old(out), *\old(mem));
6  assigns *out, self->_reg, self->_reset, (self->ni_0)->_reg, mem->_reg,
   mem->ni_0._reg;
7  */
8  void count_step(_Bool x, _Bool *out, struct count_mem *self)
9      /*@ ghost (struct count_mem_ghost \ghost *mem) */
10 {
11     int time;
12     _Bool init;
13     _Bool b;
14     count_clear_reset(self) /*@ ghost (mem) */;
15     /*@ assert count_pack0(\at(*mem,Pre), *mem, self); */;
16     /*@ assert count_transition0(\at(*mem,Pre), x, *mem); */;
17     init = _arrow_step_fc_inline(self->ni_0) /*@ ghost (& mem->ni_0) */;
18     /*@ assert count_pack1(\at(*mem,Pre), *mem, self); */;
19     /*@ assert count_transition1(\at(*mem,Pre), x, init, *mem); */;
20     b = (_Bool)(self->_reg.ptime == 3);
21     /*@ assert count_transition2(\at(*mem,Pre), x, init, b, *mem); */;
22     if (init) {
23         time = 0;
24     }
25     else {
26         if (b) {
27             time = 0;
28         }
29         else {
30             time = self->_reg.ptime + 1;
31         }
32     }
33 }

```

count.c

```

33 requires count_pack(*mem, self);
34 ensures count_pack(*mem, self);
35 ensures count_transition(\old(*mem), x, *out, *mem);
36 assigns *out;
37 assigns self->_reg;
38 assigns self->_reset;
39 assigns self->ni_0->_reg;
40 assigns mem->_reg;
41 assigns mem->ni_0._reg;
42
43 */
44 void count_step(_Bool x,
45                 _Bool (*out),
46                 struct count_mem *self)
47 /*@ ghost (struct count_mem_ghost \ghost *mem) */ {
48     int time;
49     _Bool init;
50     _Bool b;
51     count_clear_reset(self) /*@ ghost (mem) */;
52     /*@ assert count_pack0(\at(*mem, Pre), (*mem), self); */
53     /*@ assert count_transition0(\at(*mem, Pre), x, (*mem)); */
54     init = _arrow_step(self->ni_0) /*@ ghost (& mem->ni_0) */;
55     /*@ assert count_pack1(\at(*mem, Pre), (*mem), self); */
56     /*@ assert count_transition1(\at(*mem, Pre), x, init, (*mem)); */
57     b = (self->_reg.ptime == 3);
58     /*@ assert count_transition2(\at(*mem, Pre), x, init, b, (*mem)); */
59     if (init) {
60         time = 0;
61     } else {
62         if (b) {
63             time = 0;
64         } else {

```

Information Messages(1) Console Properties Values Red Alarms WP Goals

< > Module All Goals Provers... Clear

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)	-	-	●		
count_step	Assertion	Typed (Ref) (Real)	-	-	●		

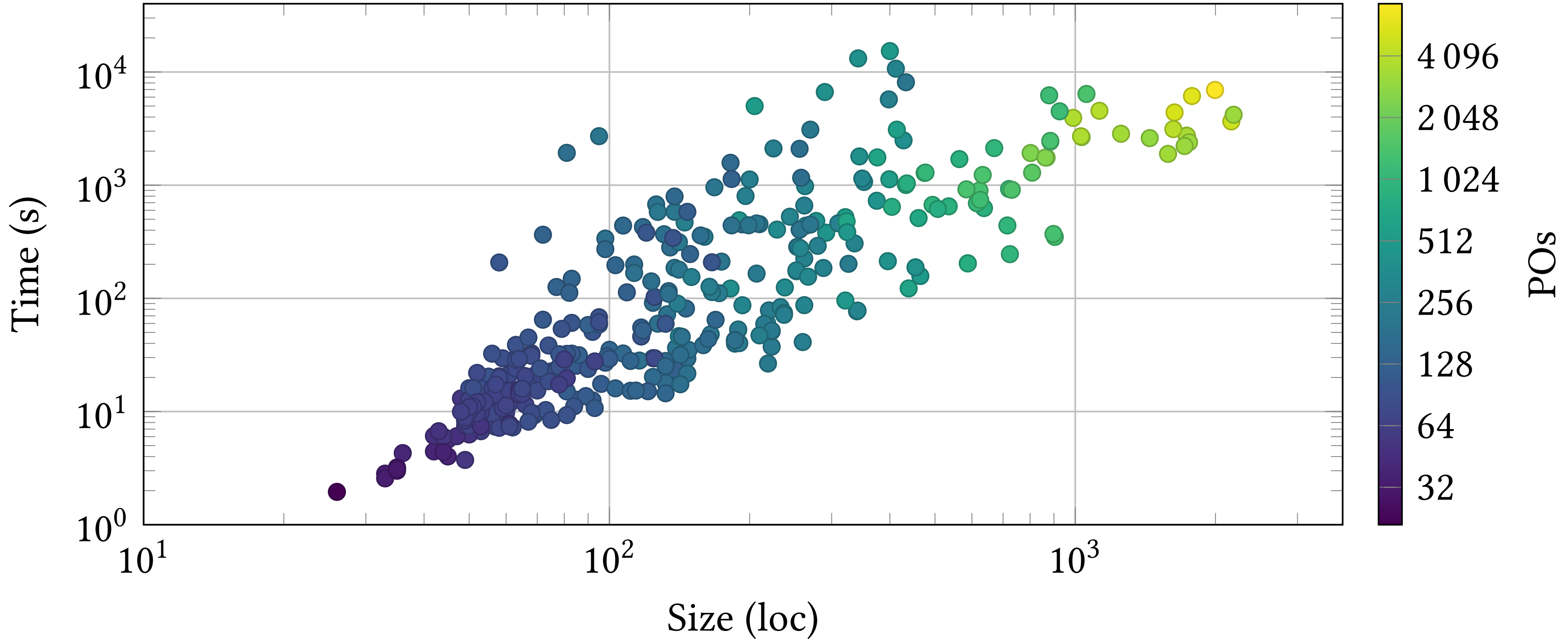
RESULTS

~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