Sémantique Mécanisée, Compilation Vérifiée et Compilation Certifiante pour Lustre

Séminaire SCALP

Lélio Brun[1]

20 mai 2021

[1]ISAE-SUPAERO – DISC – IpSC

Contexte

Systèmes embarqués

- systèmes informatiques au sein de systèmes physiques interagissant avec le monde réel, souvent sous des contraintes temps-réel
- logiciels habituellement développés avec des langages bas niveau : C, Ada, Assembleur

Systèmes embarqués

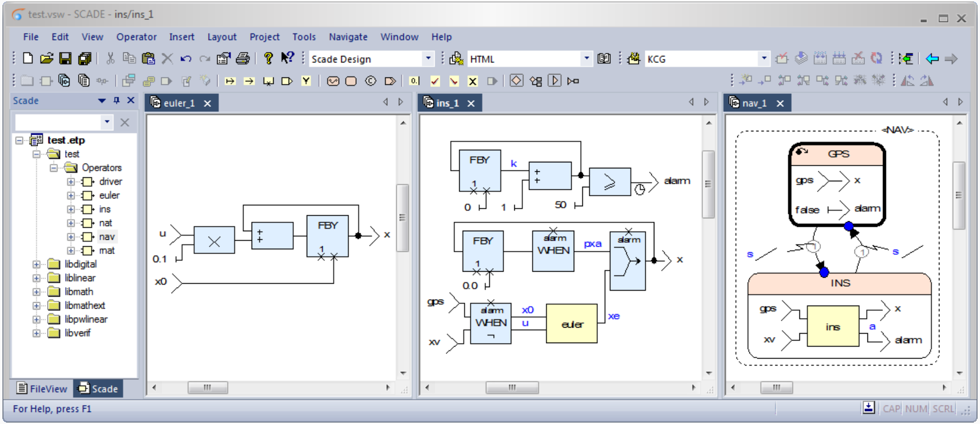
- systèmes informatiques au sein de systèmes physiques interagissant avec le monde réel, souvent sous des contraintes temps-réel
- logiciels habituellement développés avec des langages bas niveau : C, Ada, Assembleur

Model-Based Design

Spécifications abstraites de haut niveau exécutables

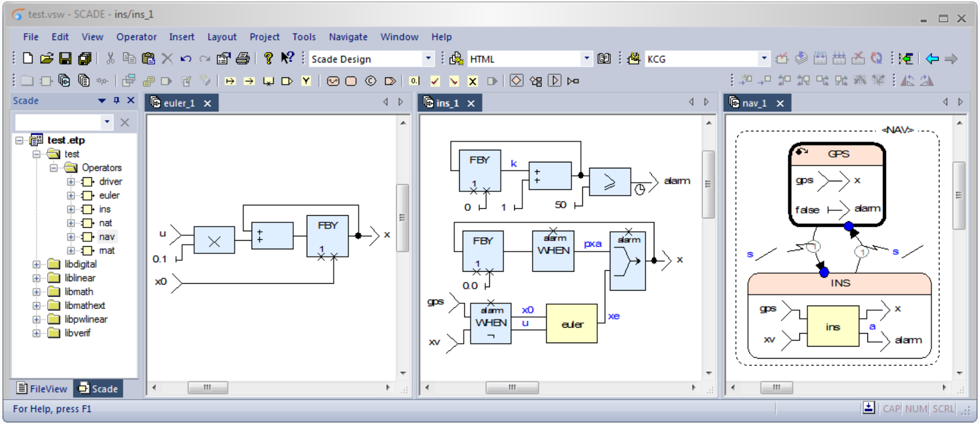
Model-Based Design dans SCADE Suite

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



Model-Based Design dans SCADE Suite

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

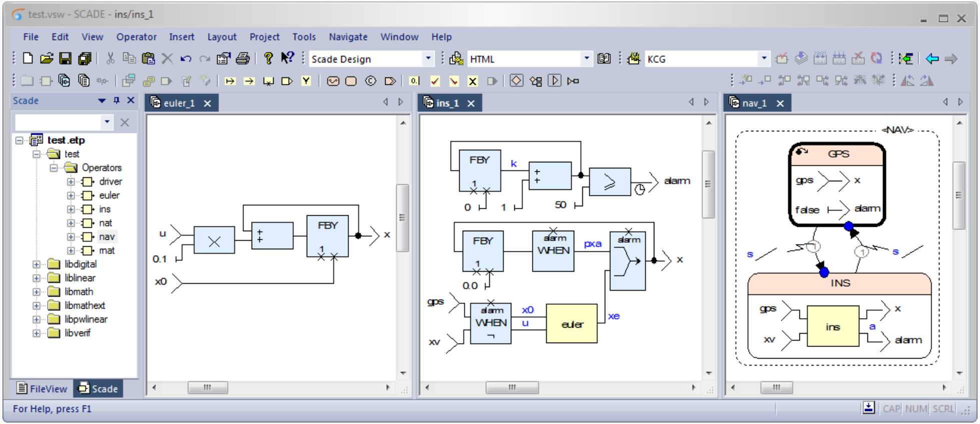


bloc / nœud = système

ligne = signal

Model-Based Design dans SCADE Suite

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

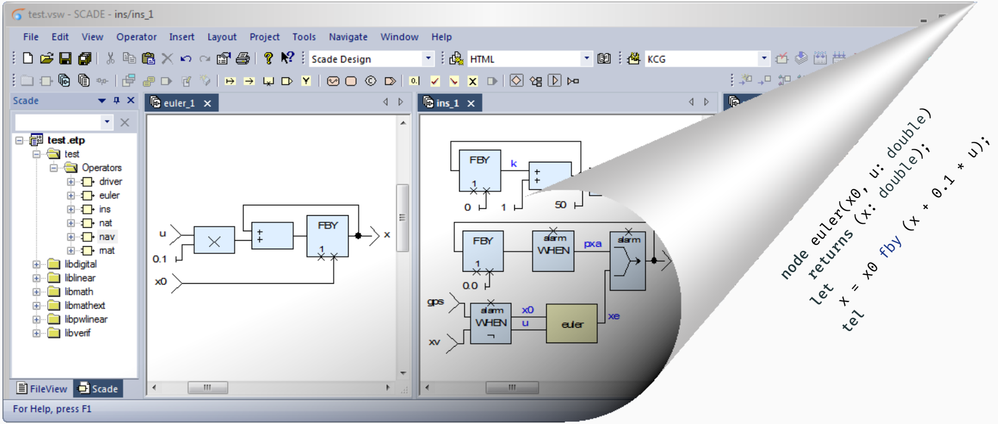


bloc / nœud = système = fonction de flots

ligne = signal = flot de valeurs

Model-Based Design dans SCADE Suite

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

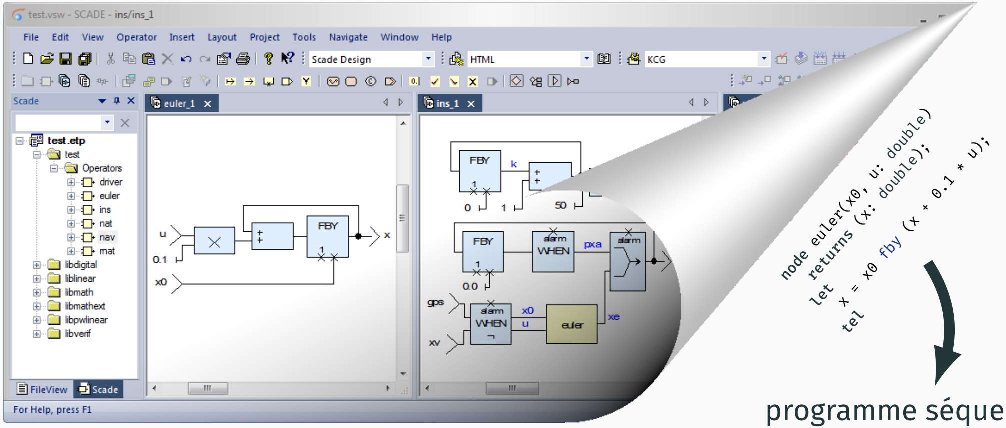


bloc / nœud = système = fonction de flots

ligne = signal = flot de valeurs

Model-Based Design dans SCADE Suite

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



programme séquentiel
(C, Ada, Assembleur)

bloc / nœud = système = fonction de flots

ligne = signal = flot de valeurs

Systèmes qui ne doivent pas échouer

- Systèmes de contrôle de vol
- Systèmes ferroviaires automatiques
- Systèmes de contrôle de centrales

CRITICITÉ

Systèmes qui ne doivent pas échouer

- Systèmes de contrôle de vol
- Systèmes ferroviaires automatiques
- Systèmes de contrôle de centrales

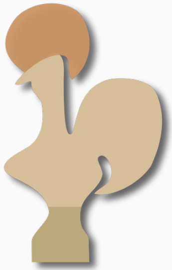
État de l'art : certification industrielle du processus de développement, parfois avec des *méthodes formelles*, ex. SCADE

Question scientifique : peut-on mécaniser les définitions formelles et produire une preuve de correction bout-à-bout ?

Formalisation Mécanisée

Assistant de Preuve

- Outils pour aider la formulation de théorèmes ainsi que le développement et la vérification de leurs preuves
- Mizar, Isabelle, HOL, Coq, ACL2, PVS, Agda, ...



Assistant de Preuve

- Outils pour aider la formulation de théorèmes ainsi que le développement et la vérification de leurs preuves
- Mizar, Isabelle, HOL, Coq, ACL2, PVS, Agda, ...

Formalisations mécanisées existantes

seL4 : un micro-noyau vérifié avec Isabelle

CakeML : un compilateur vérifié pour un langage fonctionnel avec HOL

CompCert : une étape clef

Formalisation mécanisée avec Coq du langage C et de la preuve de correction de sa compilation vers du code Assembleur.

Langages pour le Model-Based Design
Scade 6, Lustre

+

Assistants de Preuve
Coq

Défis

1. Mécaniser les sémantiques
2. Prouver la correction des algorithmes de compilation

Langages pour le Model-Based Design
Scade 6, Lustre

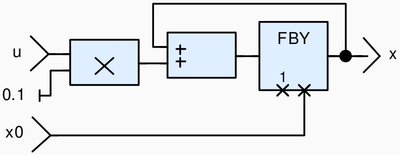
+

Assistants de Preuve
Coq

Défis

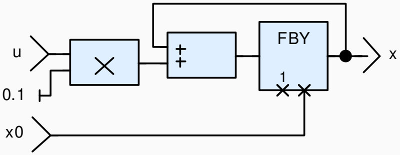
1. Mécaniser les sémantiques
2. Prouver la correction des algorithmes de compilation

Focus : réinitialisation modulaire (*modular reset*)




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

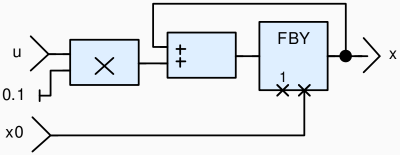
x_0	0.00	1.55	3.62	5.46	$\cdots$
u	15.00	20.00	17.00	12.00	$\cdots$
$x + 0.1 \times u$	1.50	3.50	5.20	6.70	$\cdots$
x	0.00	1.50	3.50	5.20	$\cdots$




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

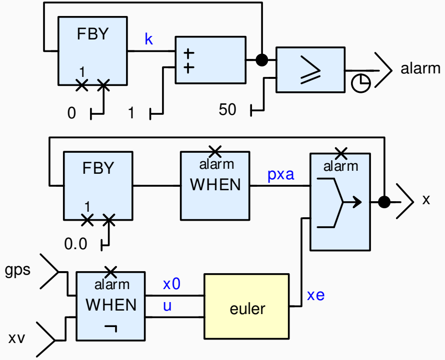
x_0	0.00	1.55	3.62	5.46	$\cdots$
u	15.00	20.00	17.00	12.00	$\cdots$
$x + 0.1 \times u$	1.50	3.50	5.20	6.70	$\cdots$
x	0.00	1.50	3.50	5.20	$\cdots$

Exemple



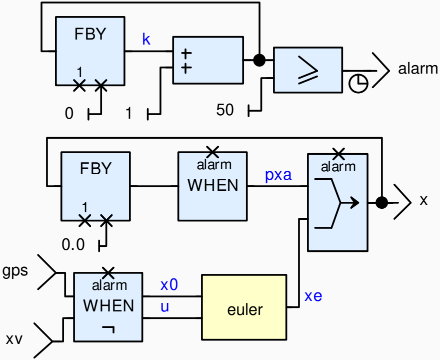

```
node euler(x0, u: double)
  returns (x: double);
let
  x = x0 fby (x + 0.1 * u);
tel
```

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




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

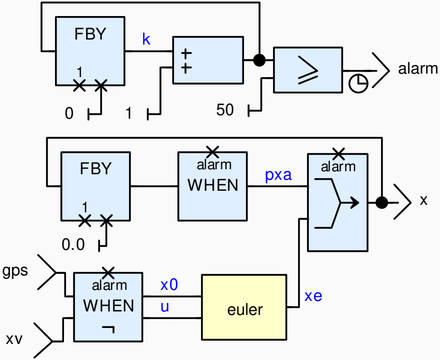
gps	0.00	1.55	3.62	5.46	...	86.52	88.40	90.91	...
xv	15.00	20.00	17.00	12.00	...	18.00	23.00	20.00	...
k	0	1	2	3	...	49	50	51	...
alarm	F	F	F	F	...	F	T	T	...
xe	0.00	1.50	3.50	5.20	...	77.35			...
pxa					...		77.35	77.35	...
x	0.00	1.50	3.50	5.20	...	77.35	77.35	77.35	...




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

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

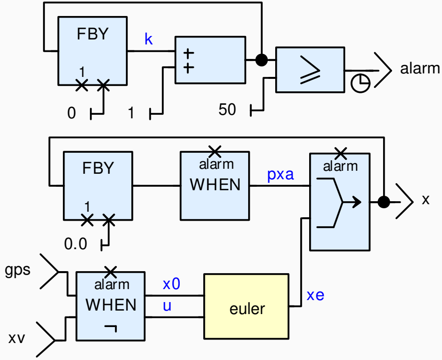
Exemple




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

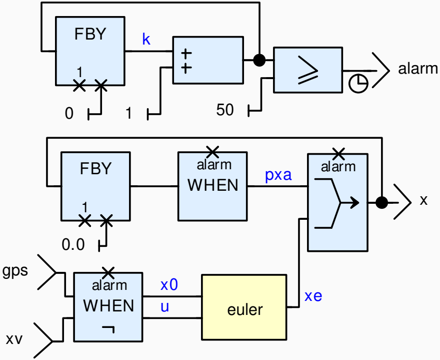
gps	0.00	1.55	3.62	5.46	···	86.52	88.40	90.91	···
xv	15.00	20.00	17.00	12.00	···	18.00	23.00	20.00	···
k	0	1	2	3	···	49	50	51	···
alarm	F	F	F	F	···	F	T	T	···
xe	0.00	1.50	3.50	5.20	···	77.35			···
pxa					···		77.35	77.35	···
x	0.00	1.50	3.50	5.20	···	77.35	77.35	77.35	···

Exemple




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

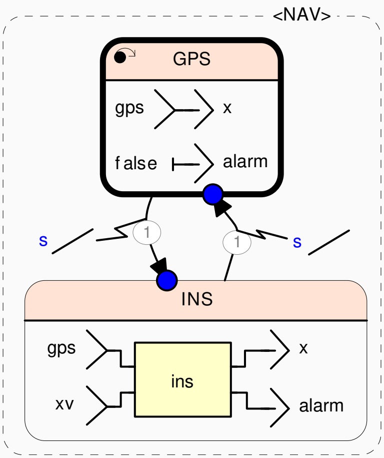
gps	0.00	1.55	3.62	5.46	···	86.52	88.40	90.91	···
xv	15.00	20.00	17.00	12.00	···	18.00	23.00	20.00	···
k	0	1	2	3	···	49	50	51	···
alarm	F	F	F	F	···	F	T	T	···
xe	0.00	1.50	3.50	5.20	···	77.35			···
pxa					···		77.35	77.35	···
x	0.00	1.50	3.50	5.20	···	77.35	77.35	77.35	···




```
node ins(gps, xv: double)
  returns (x: double, alarm: bool)
  var pxa, xe: double; k: int;
let
  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
```

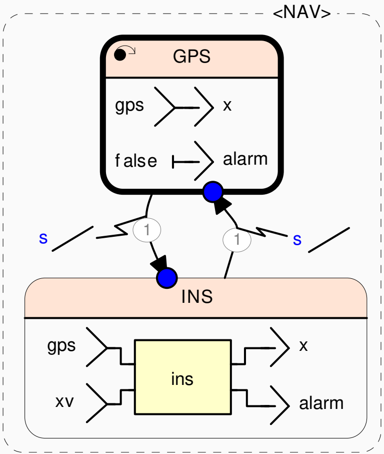
gps	0.00	1.55	3.62	5.46	...	86.52	88.40	90.91	...
xv	15.00	20.00	17.00	12.00	...	18.00	23.00	20.00	...
k	0	1	2	3	...	49	50	51	...
alarm	F	F	F	F	...	F	T	T	...
xe	0.00	1.50	3.50	5.20	...	77.35			...
pxa					...		77.35	77.35	...
x	0.00	1.50	3.50	5.20	...	77.35	77.35	77.35	...

Exemple




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

Exemple

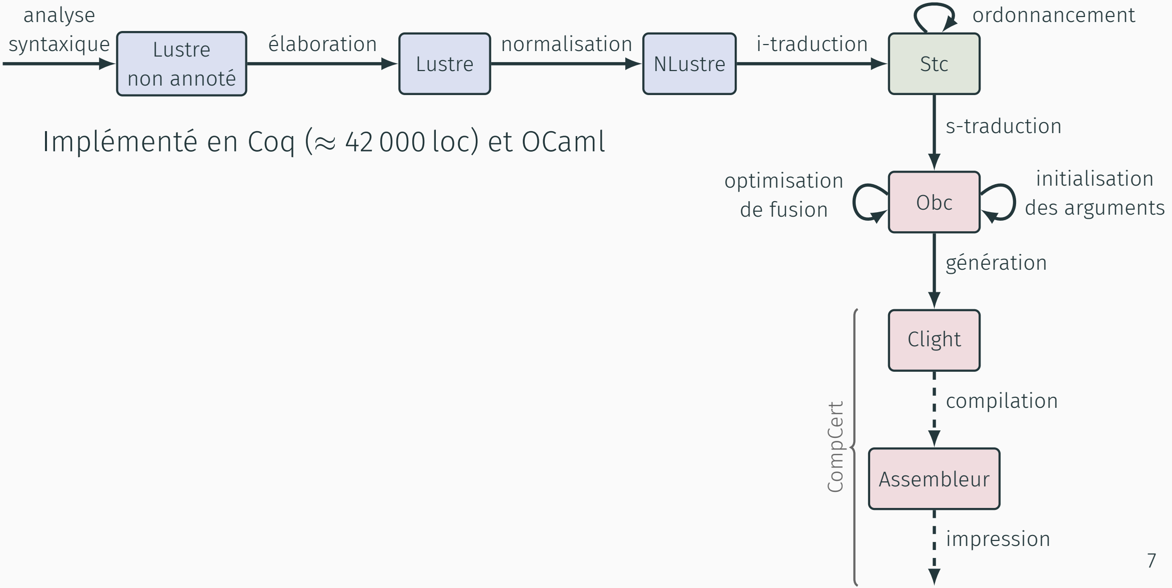



```
node nav(gps, xv: double, s: bool)
  returns (x: double, alarm: bool)
  var r, c: bool;
let
  (x, alarm) = merge c
                 (gps when c, false)
                 ((restart ins every r)
                     ((gps, xv) whenot c));
  c = true fby (merge c (not s when c)
                     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

Il faut un moyen de réinitialiser l'état d'un nœud

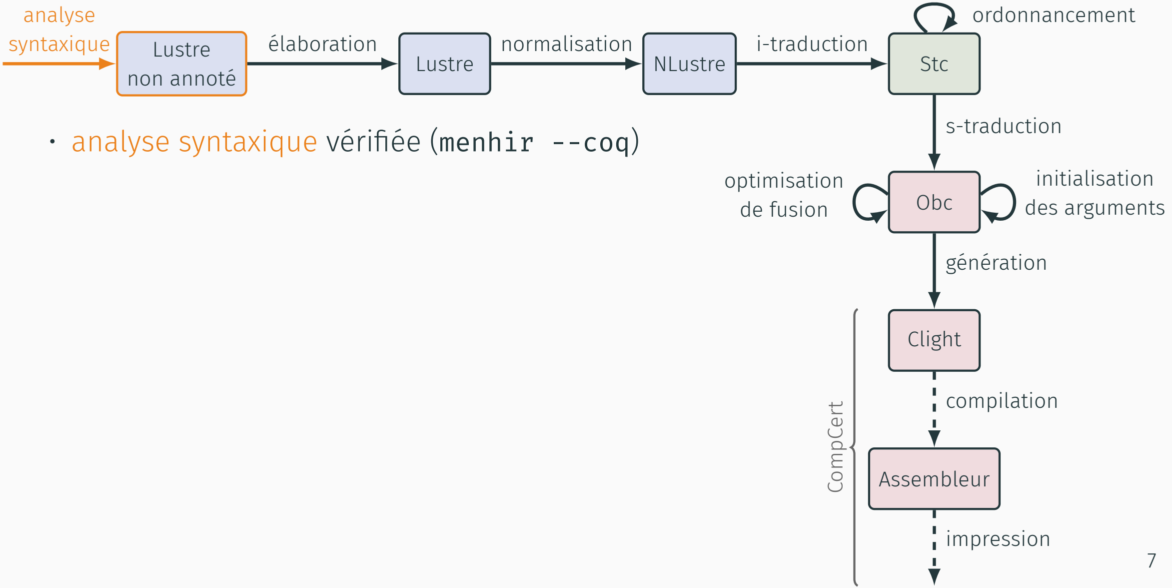
Vélus : un Compilateur Lustre Vérifié

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



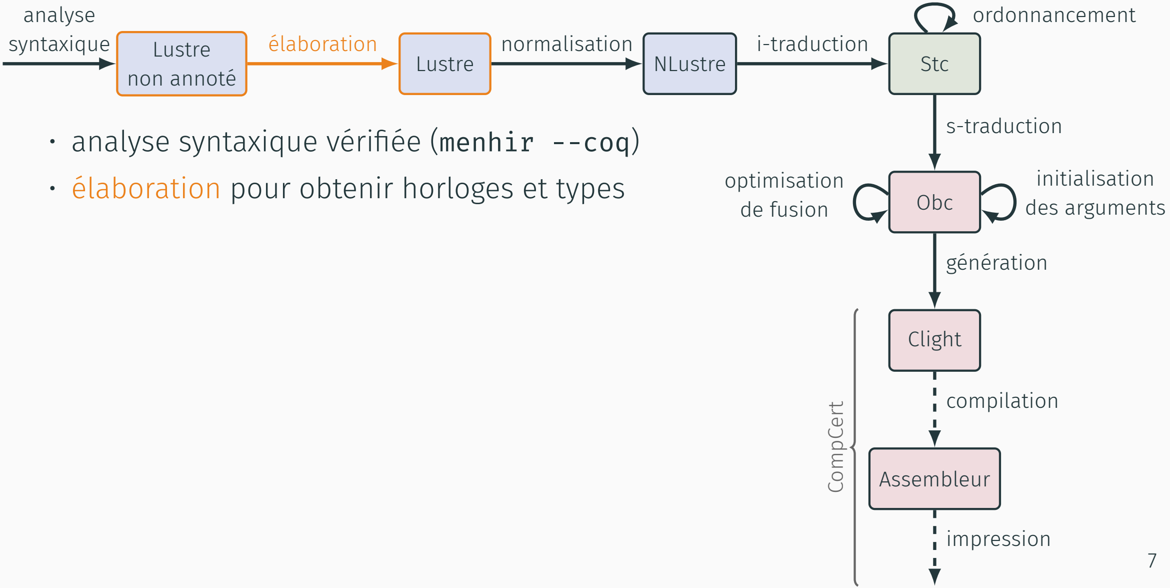
Implémenté en Coq ($\approx$ 42 000 loc) et OCaml

Vélus : un Compilateur Lustre Vérifié



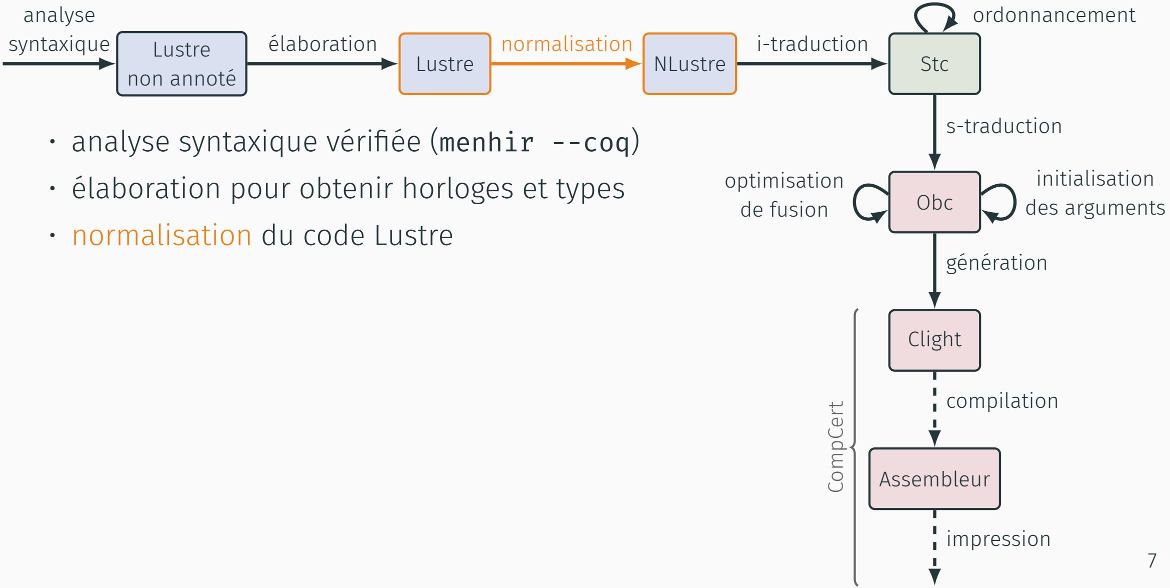
- analyse syntaxique vérifiée (`menhir --coq`)

Vélus : un Compilateur Lustre Vérifié



- analyse syntaxique vérifiée (`menhir --coq`)
- élaboration pour obtenir horloges et types

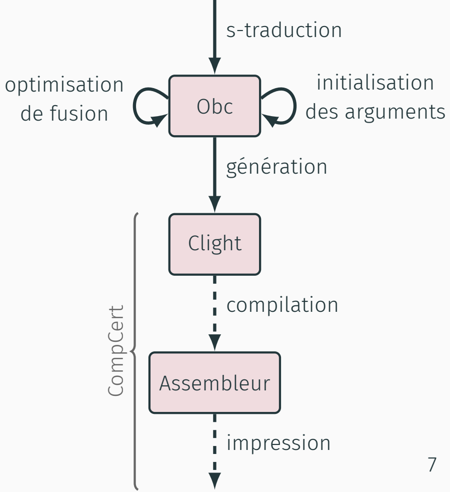
Vélus : un Compilateur Lustre Vérifié



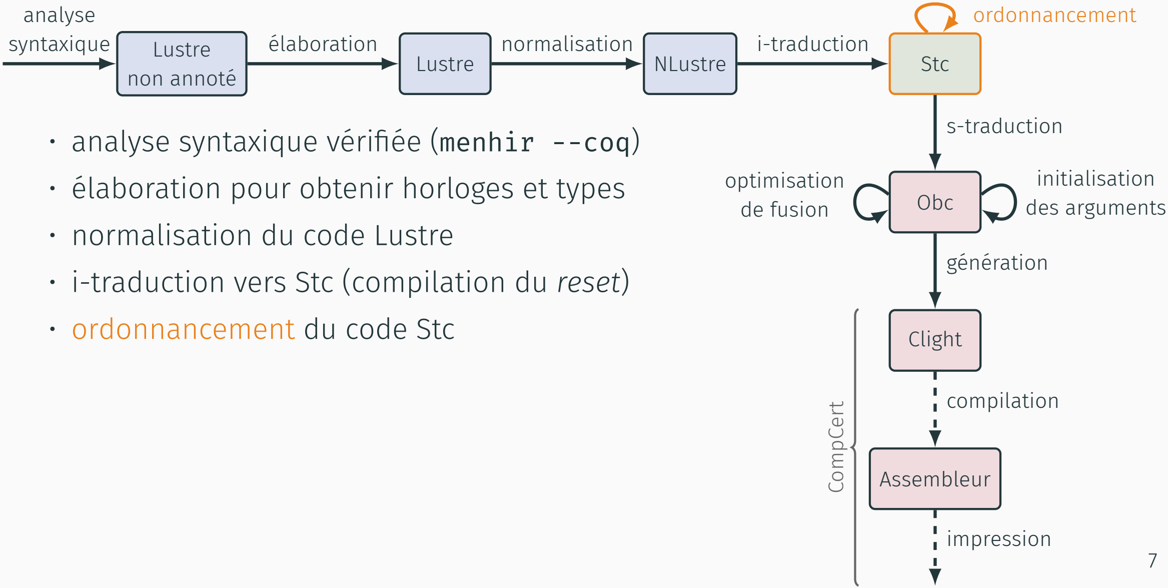
- analyse syntaxique vérifiée (`menhir --coq`)
- élaboration pour obtenir horloges et types
- normalisation du code Lustre

Vélus : un Compilateur Lustre Vérifié

- analyse syntaxique vérifiée (`menhir --coq`)
- élaboration pour obtenir horloges et types
- normalisation du code Lustre
- i-traduction vers Stc (compilation du *reset*)

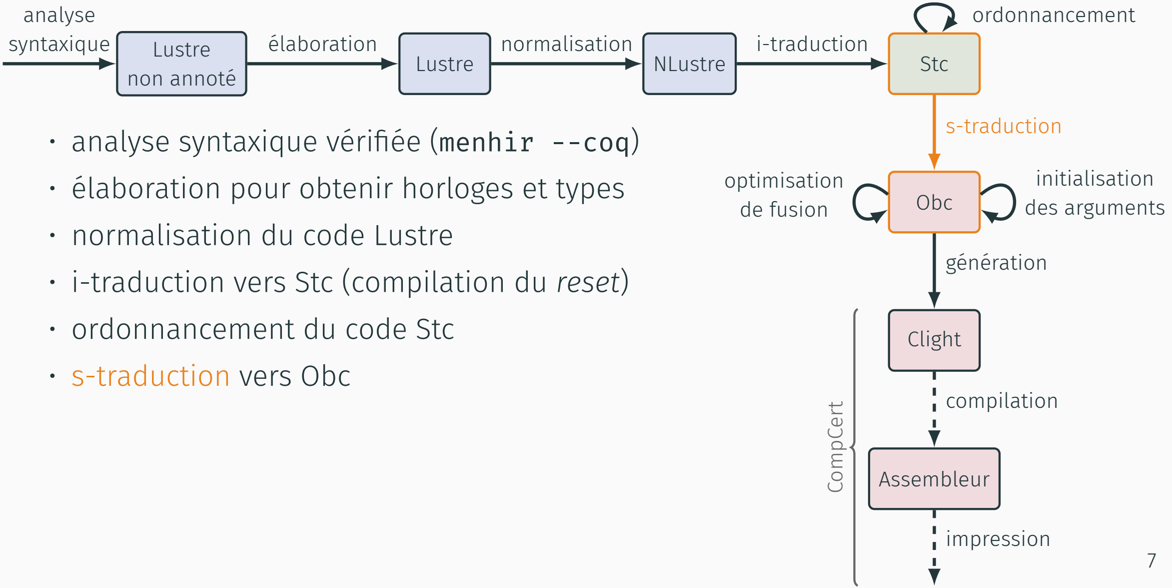


Vélus : un Compilateur Lustre Vérifié

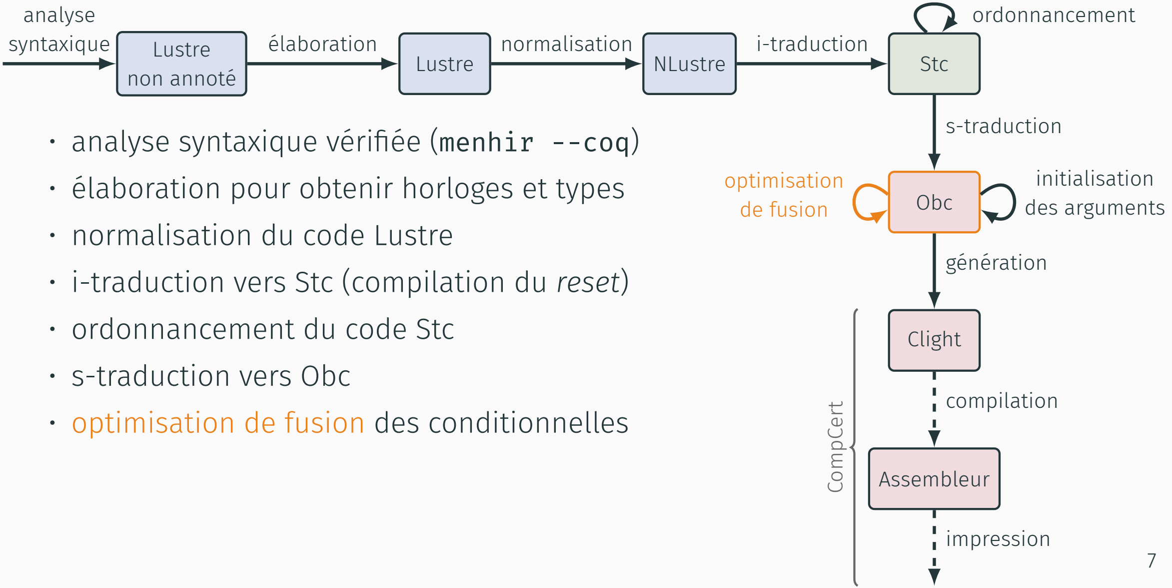


- analyse syntaxique vérifiée (`menhir --coq`)
- élaboration pour obtenir horloges et types
- normalisation du code Lustre
- i-traduction vers Stc (compilation du *reset*)
- ordonnancement du code Stc

Vélus : un Compilateur Lustre Vérifié

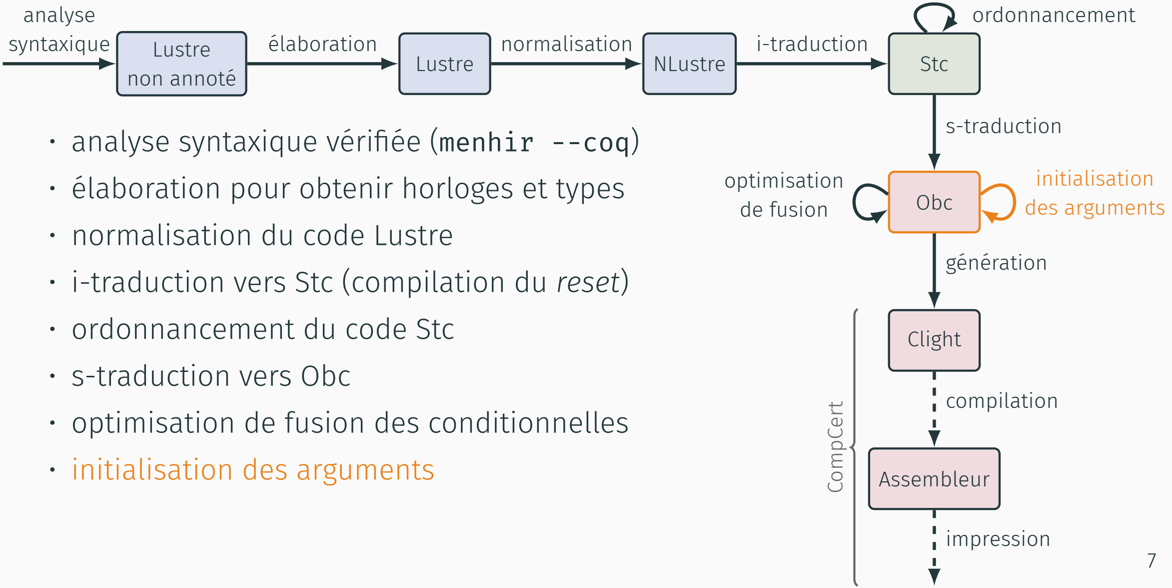


- analyse syntaxique vérifiée (`menhir --coq`)
- élaboration pour obtenir horloges et types
- normalisation du code Lustre
- i-traduction vers Stc (compilation du *reset*)
- ordonnancement du code Stc
- s-traduction vers Obc



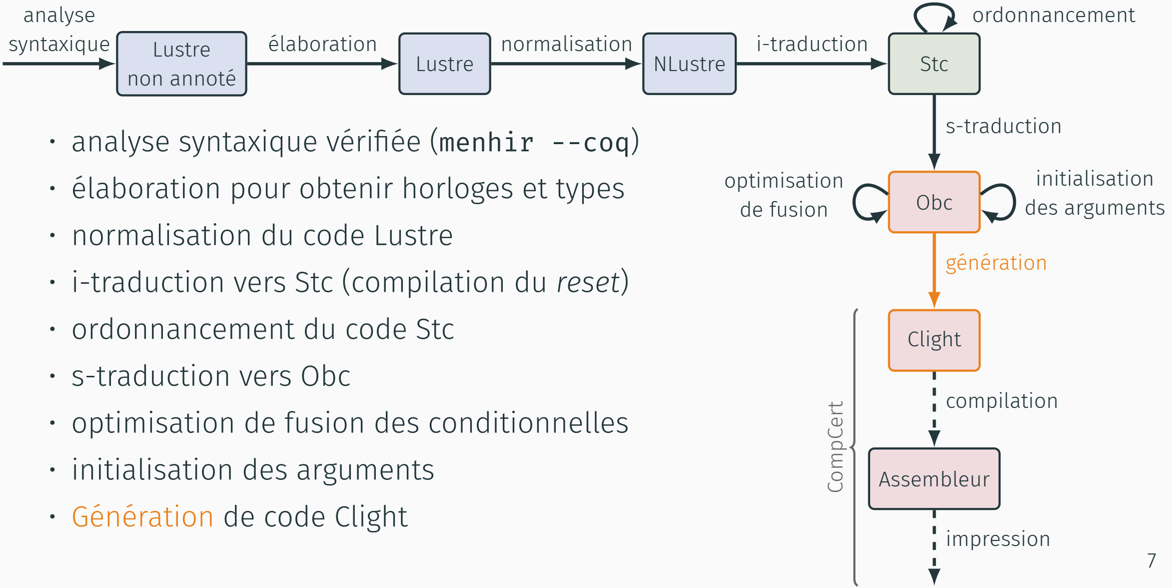
- analyse syntaxique vérifiée (`menhir --coq`)
- élaboration pour obtenir horloges et types
- normalisation du code Lustre
- i-traduction vers Stc (compilation du *reset*)
- ordonnancement du code Stc
- s-traduction vers Obc
- optimisation de fusion des conditionnelles

Vélus : un Compilateur Lustre Vérifié



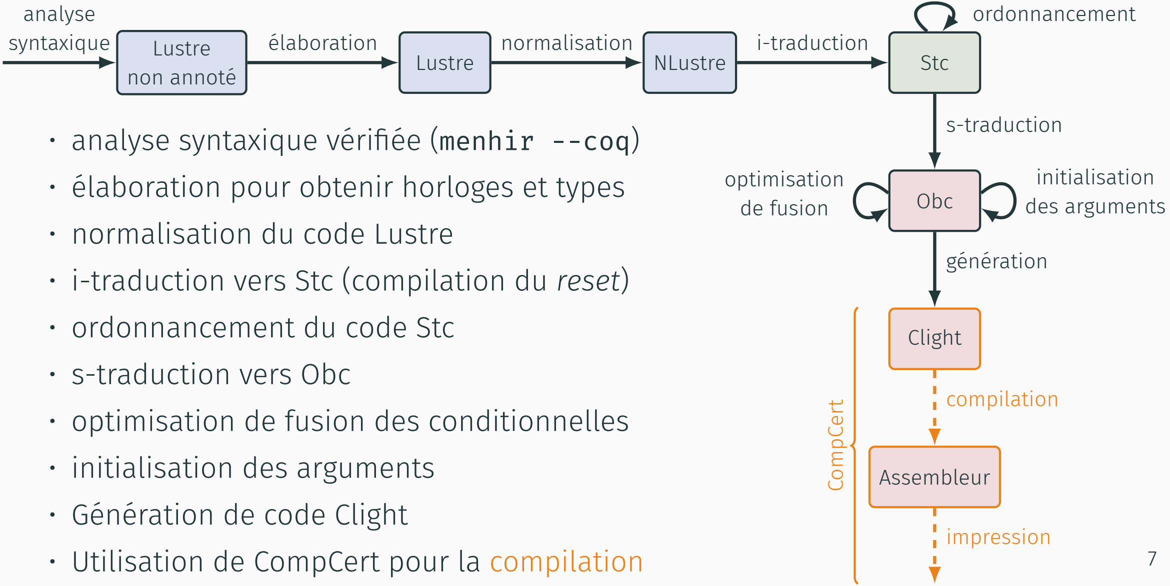
- analyse syntaxique vérifiée (`menhir --coq`)
- élaboration pour obtenir horloges et types
- normalisation du code Lustre
- i-traduction vers Stc (compilation du *reset*)
- ordonnancement du code Stc
- s-traduction vers Obc
- optimisation de fusion des conditionnelles
- initialisation des arguments

Vélus : un compilateur Lustre vérifié



- analyse syntaxique vérifiée (`menhir --coq`)
- élaboration pour obtenir horloges et types
- normalisation du code Lustre
- i-traduction vers Stc (compilation du *reset*)
- ordonnancement du code Stc
- s-traduction vers Obc
- optimisation de fusion des conditionnelles
- initialisation des arguments
- Génération de code Clight

Vélus : un Compilateur Lustre Vérifié



- analyse syntaxique vérifiée (`menhir --coq`)
- élaboration pour obtenir horloges et types
- normalisation du code Lustre
- i-traduction vers Stc (compilation du *reset*)
- ordonnancement du code Stc
- s-traduction vers Obc
- optimisation de fusion des conditionnelles
- initialisation des arguments
- Génération de code Clight
- Utilisation de CompCert pour la compilation

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

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

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

Lustre

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

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

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

C

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

double fun$euler$step(struct euler *self,
                      double x0, double u) {
  register double x;
  if (self->i) {
    x = x0;
  } else {
    x = self->px;
  }
  self->i = false;
  self->px = x + 0.100000000000000006 * u;
  return x;
}

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

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

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

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

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

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

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

  fun$nav$reset(&self$);

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

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

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

Lustre

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

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

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

C

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

double fun$euler$step(struct euler *self,
                      double x0, double u) {
  register double x;
  if (self->i) {
    x = x0;
  } else {
    x = self->px;
  }
  self->i = false;
  self->px = x + 0.100000000000000006 * u;
  return x;
}

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

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

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

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

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

code traduit

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

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

  fun$nav$reset(&self$);

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

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

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

Lustre

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

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

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

C

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

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

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

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

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

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

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

boucle principale

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

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

  fun$nav$reset(&self$);

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

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

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

Lustre

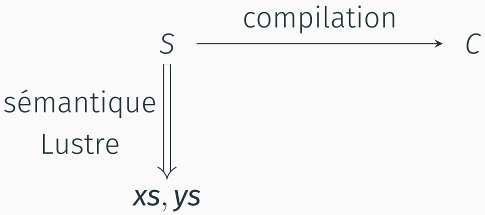
```
node euler(x0, u: double)
  returns (x: double);
let
  x = x0 fby (x + 0.1 * u);
tel

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

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

Assembleur

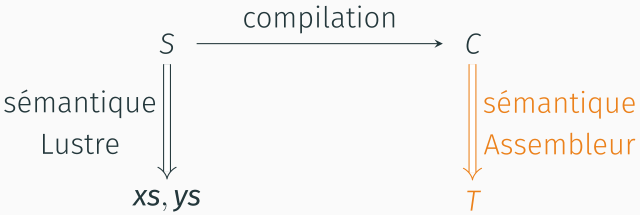
```
        .data
        .align  8
        .globl  self$
self$:
        .space  40
        .type   self$, @object
        .size   self$, . - self$
        .data
        .align  8
        .globl  x$
x$:
        .space  8
        .type   x$, @object
        .size   x$, . - x$
        .data
        .align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

Correction ?

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


compilation
S
C
sémantique
Lustre
xs, ys

Correction ?


compilation
S
C
sémantique Lustre
sémantique Assembleur
xs, ys
T

Correction ?


compilation
S
C
sémantique
Lustre
sémantique
Assembleur
xs, ys
T
« équivalence »

Remarque : on veut en réalité la direction opposée, appelée *raffinement*, c'est-à-dire les comportements observables de C sont aussi des comportements observables de S.

Mécanisation de Lustre Normalisé

4 types d'équations

$x = ce$	équation simple
$x = c$ **fby** e	équation **fby**
$\boldsymbol{x} = f(\boldsymbol{e})$	instanciation de nœud
$\boldsymbol{x} = ($**restart** f **every** $r)(\boldsymbol{e})$	instanciation avec *reset* modulaire

Sémantique

Flots comme fonctions $\mathbb{N} \mapsto$ value :

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

sémantique instantanée projetée

Flots comme coinductifs :

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

description coinductive de la sémantique

Sémantique Instantanée

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

Sémantique Projetée

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

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

Sémantique Instantanée

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

Sémantique Projetée

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

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

Sémantique Instantanée

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

Sémantique Projetée

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

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

Sémantique Instantanée

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

Sémantique Projetée

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

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

Sémantique Instantanée

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

Sémantique Projetée

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

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

Sémantique Instantanée

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

Sémantique Projetée

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

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

Sémantique Instantanée

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

Sémantique Projetée

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

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

Sémantique Instantanée

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

Sémantique Projetée

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

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

Sémantique Instantanée

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

Sémantique Projetée

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

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

Sémantique Instantanée

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

Sémantique Projetée

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

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

Sémantique Instantanée

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

Sémantique Projetée

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

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

Sémantique de Nœud

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

Sémantique de Nœud

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

Sémantique de Nœud

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

Sémantique de Nœud

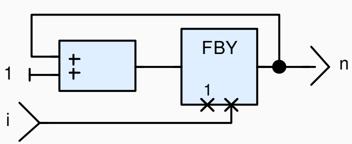
$$\frac{\text{node}(G, f) = n \quad \forall i,\ H_i(n.\textbf{in}) = xs_i \quad \forall i,\ H_i(n.\textbf{out}) = ys_i \qquad \forall eq \in n.\textbf{eqs},\ H \vdash eq}{\vdash f(xs) \Downarrow ys}$$

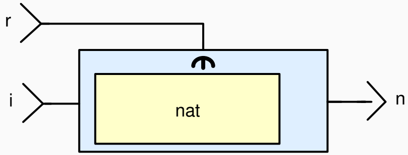
Sémantique de Nœud

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

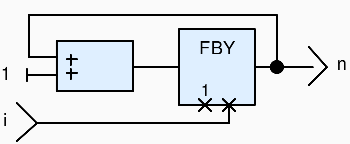
Exemple Plus Simple : Sémantique Intuitive du Reset Modulaire

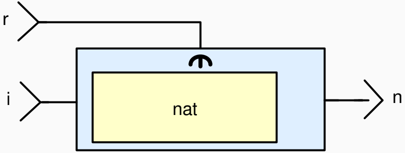
```
node nat(i: int)
  returns (n: int)
let
  n = i fby (n + 1);
tel
```





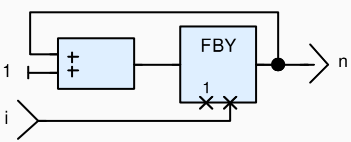
```
node nat(i: int)
  returns (n: int)
let
  n = i fby (n + 1);
tel
```

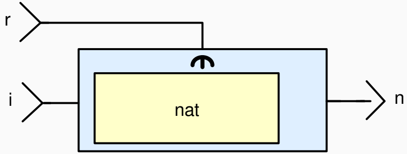




r	F
i	0
$nat(i)$	0
$(\texttt{restart}\ nat\ \texttt{every}\ r)(i)$	0

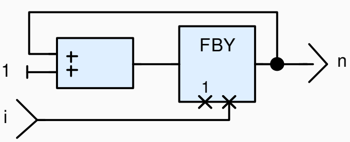
```
node nat(i: int)
  returns (n: int)
let
  n = i fby (n + 1);
tel
```

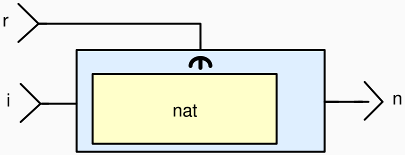




r	F	F
i	0	5
$nat(i)$	0	1
$(\texttt{restart}\ nat\ \texttt{every}\ r)(i)$	0	1

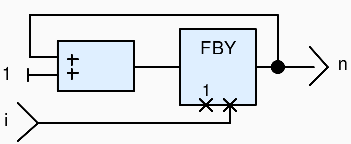
```
node nat(i: int)
  returns (n: int)
let
  n = i fby (n + 1);
tel
```

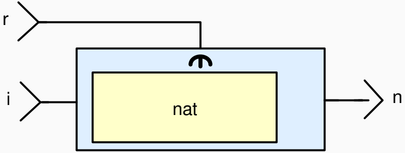




r	F	F	T
i	0	5	10
$nat(i)$	0	1	2
$(\texttt{restart}\ nat\ \texttt{every}\ r)(i)$	0	1	10

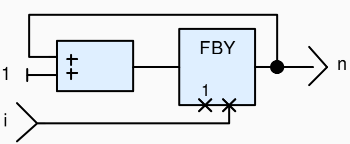
```
node nat(i: int)
  returns (n: int)
let
  n = i fby (n + 1);
tel
```

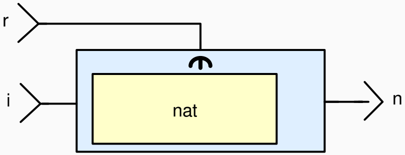




r	F	F	T	F
i	0	5	10	15
$nat(i)$	0	1	2	3
(**restart** nat **every** r)(i)	0	1	10	11

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

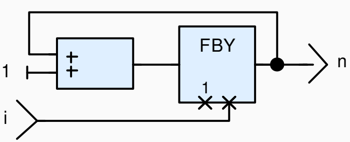


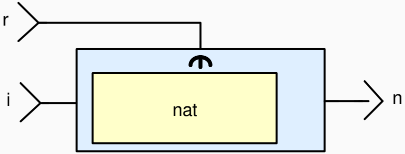


r	F	F	T	F	F
i	0	5	10	15	20
$nat(i)$	0	1	2	3	4
$($**restart** nat **every** $r)(i)$	0	1	10	11	12

Exemple Plus Simple : Sémantique Intuitive du Reset Modulaire

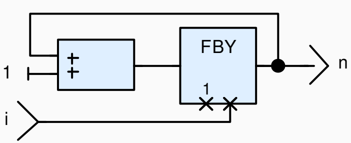
```
node nat(i: int)
  returns (n: int)
let
  n = i fby (n + 1);
tel
```

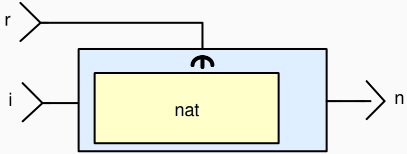




r	F	F	T	F	F	T
i	0	5	10	15	20	25
nat(*i*)	0	1	2	3	4	5
(**restart** *nat* **every** *r*)(*i*)	0	1	10	11	12	25

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

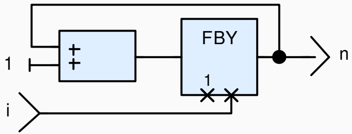


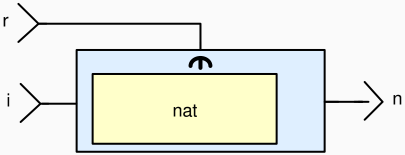


r	F	F	T	F	F	T	F
i	0	5	10	15	20	25	30
$nat(i)$	0	1	2	3	4	5	6
(**restart** nat **every** r)(i)	0	1	10	11	12	25	26

Exemple Plus Simple : Sémantique Intuitive du Reset Modulaire

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



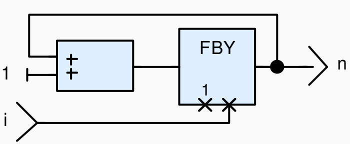


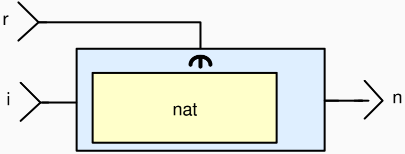
r	F	F	T	F	F	T	F	$\cdots$
i	0	5	10	15	20	25	30	$\cdots$
$nat(i)$	0	1	2	3	4	5	6	$\cdots$
(**restart** nat **every** r)(i)	0	1	10	11	12	25	26	$\cdots$

Exemple Plus Simple : Sémantique Intuitive du Reset Modulaire

Caspi 1994
Hamon et Pouzet 2000

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



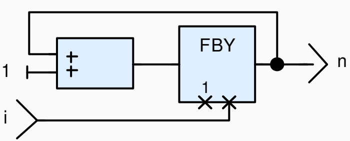


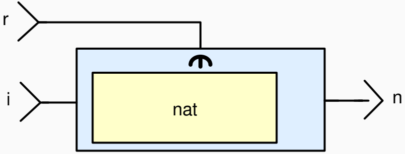
r	F	F	T	F	F	T	F	...
i	0	5	10	15	20	25	30	...
nat(*i*)	0	1	2	3	4	5	6	...
(**restart** *nat* **every** *r*)(*i*)	0	1	10	11	12	25	26	...

Peut être implémenté dans un langage récursif d'ordre supérieur

Exemple Plus Simple : Sémantique Intuitive du Reset Modulaire

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





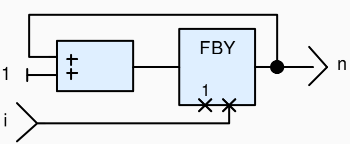
r	F	F	T	F	F	T	F	...
i	0	5	10	15	20	25	30	...
$nat(i)$	0	1	2	3	4	5	6	...
(**restart** nat **every** r)(i)	0	1	10	11	12	25	26	...

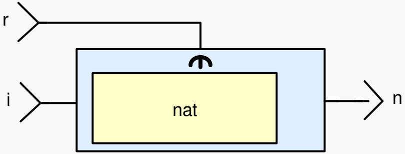
Peut être implémenté dans un langage récursif d'ordre supérieur

Exemple Plus Simple : Sémantique Intuitive du Reset Modulaire

Caspi 1994
Hamon et Pouzet 2000

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





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

Peut être implémenté dans un langage récursif d'ordre supérieur

Exemple Plus Simple : Sémantique Intuitive du Reset Modulaire

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

Exemple Plus Simple : Sémantique Intuitive du Reset Modulaire

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

Exemple Plus Simple : Sémantique Intuitive du Reset Modulaire

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

Exemple Plus Simple : Sémantique Intuitive du Reset Modulaire

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

Exemple Plus Simple : Sémantique Intuitive du Reset Modulaire

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

Exemple Plus Simple : Sémantique Intuitive du Reset Modulaire

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

Exemple Plus Simple : Sémantique Intuitive du Reset Modulaire

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

Exemple Plus Simple : Sémantique Intuitive du Reset Modulaire

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

Exemple Plus Simple : Sémantique Intuitive du Reset Modulaire

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

Instanciation de nœud

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

Instanciation de nœud

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

Reset modulaire

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

Instanciation de nœud

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

Reset modulaire

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

Instanciation de nœud

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

Reset modulaire

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

Instanciation de nœud

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

Reset modulaire

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

Relation universellement quantifiée : nombre non borné de contraintes

Compilation du Reset Modulaire : de NLustre vers Stc

Un Problème avec la Compilation de NLustre vers Obc

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

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

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

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

Un Problème avec la Compilation de NLustre vers Obc

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

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

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

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

Un Problème avec la Compilation de NLustre vers Obc

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

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

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

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

Un Problème avec la Compilation de NLustre vers Obc

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

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

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

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

Un Problème avec la Compilation de NLustre vers Obc

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

ordonnancer *et* introduire l'état

VS

introduire l'état *puis* ordonnancer

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

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

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

STC : SYNCHRONOUS TRANSITION CODE

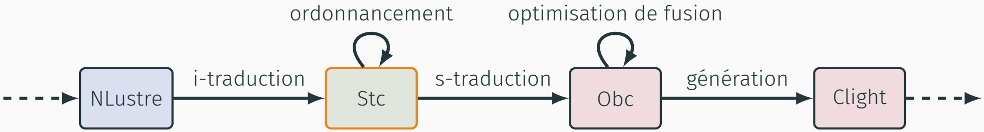
Proposer un nouveau langage intermédiaire

- Sémantique invariante par permutation
- Reset séparé
- Variables d'état et instances explicites

Stc : Synchronous Transition Code

Proposer un nouveau langage intermédiaire

- Sémantique invariante par permutation
- Reset séparé
- Variables d'état et instances explicites



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

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

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

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

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

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

NLustre

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

introduire l'état uniquement

Stc

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

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

NLustre

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

introduire l'état uniquement

Stc

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

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

NLustre

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

introduire l'état uniquement

Stc

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

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

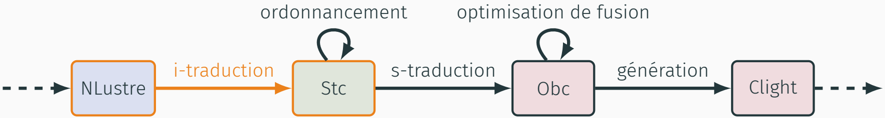
## NLustre	Stc

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

introduire l'état uniquement

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

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

Compilation de l'Exemple du Reset

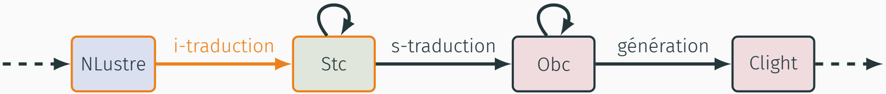



```
node driver(x0, y0, u, v: double, r: bool)
  returns (x, y: double)
  var ax, ay: bool;
let
  x, ax = (restart ins every r)(x0, u);
  y, ay = (restart ins every r)(y0, v);
tel
```

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

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

introduire l'état uniquement

Compilation de l'Exemple du Reset

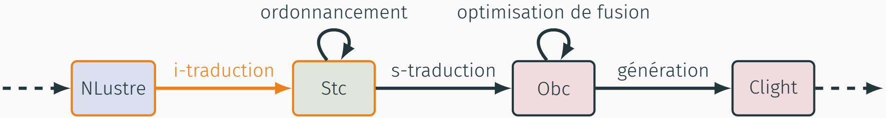



```
node driver(x0, y0, u, v: double, r: bool)
  returns (x, y: double)
  var ax, ay: bool;
let
  x, ax = (restart ins every r)(x0, u);
  y, ay = (restart ins every r)(y0, v);
tel
```

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

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

introduire l'état uniquement

Compilation de l'Exemple du Reset

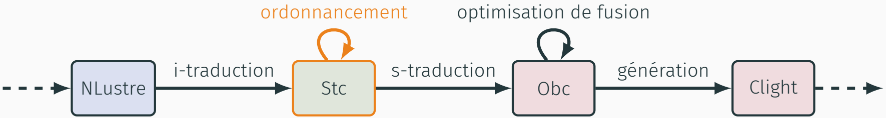



```
node driver(x0, y0, u, v: double, r: bool)
  returns (x, y: double)
  var ax, ay: bool;
let
  x, ax = (restart ins every r)(x0, u);
  y, ay = (restart ins every r)(y0, v);
tel
```

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

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

introduire l'état uniquement

Compilation de l'Exemple du Reset




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

ordonnancer uniquement

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

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

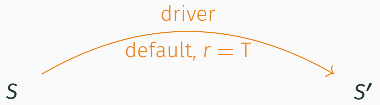
Système de transitions

- États de départ S, d'arrivée S'
- Contraintes de transition
- État intermédiaire I

Système de transitions

- États de départ S, d'arrivée S'
- Contraintes de transition
- État intermédiaire I

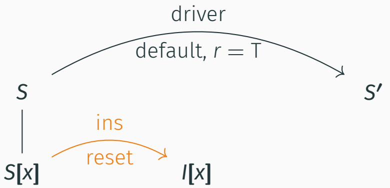
```
system driver {
  sub x: ins, y: ins;

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



Système de transitions

- États de départ S, d'arrivée S'
- Contraintes de transition
- État intermédiaire I

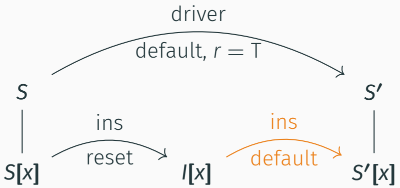
```
system driver {
  sub x: ins, y: ins;

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



Système de transitions

- États de départ S, d'arrivée S'
- Contraintes de transition
- État intermédiaire I

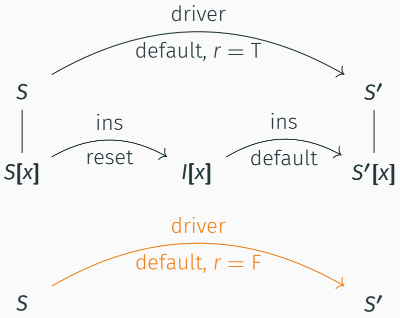
```
system driver {
  sub x: ins, y: ins;

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



Système de transitions

- États de départ S, d'arrivée S'
- Contraintes de transition
- État intermédiaire I

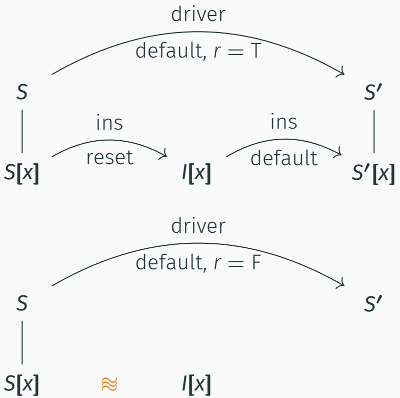
```
system driver {
  sub x: ins, y: ins;

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



Système de transitions

- États de départ S, d'arrivée S'
- Contraintes de transition
- État intermédiaire I

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

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

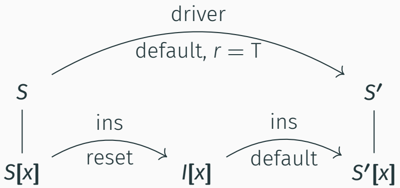


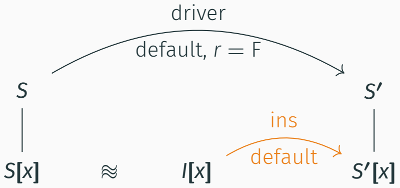
Sémantique Intuitive de Stc

Système de transitions

- États de départ S, d'arrivée S'
- Contraintes de transition
- État intermédiaire I

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

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





Contrainte de transition basique

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

Contrainte de transition *next*

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

Contrainte de transition basique

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

Contrainte de transition *next*

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

Contrainte de transition basique

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

Contrainte de transition *next*

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

Contrainte de transition basique

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

Contrainte de transition *next*

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

Contrainte de transition basique

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

Contrainte de transition *next*

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

Contrainte de transition basique

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

Contrainte de transition *next*

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

Contrainte de transition basique

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

Contrainte de transition *next*

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

Contrainte de transition basique

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

Contrainte de transition *next*

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

Transition par défaut

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

Transition *reset*

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

Transition par défaut

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

Transition *reset*

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

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

Transition par défaut

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

Transition *reset*

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

Transition par défaut

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

Transition *reset*

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

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

Transition par défaut

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

Transition *reset*

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

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

Transition par défaut

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

Transition *reset*

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

Transition par défaut

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

Transition *reset*

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

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

Transition par défaut

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

Transition *reset*

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

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

Système

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

Système

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

Système

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

Système

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

Système

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

Exécution en boucle

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

Exécution en boucle

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

Exécution en boucle

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

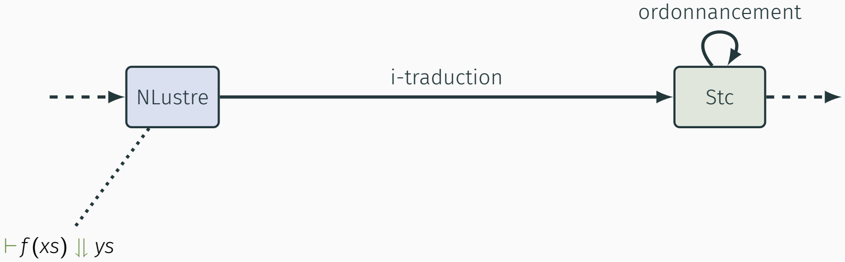
Exécution en boucle

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

Exécution en boucle

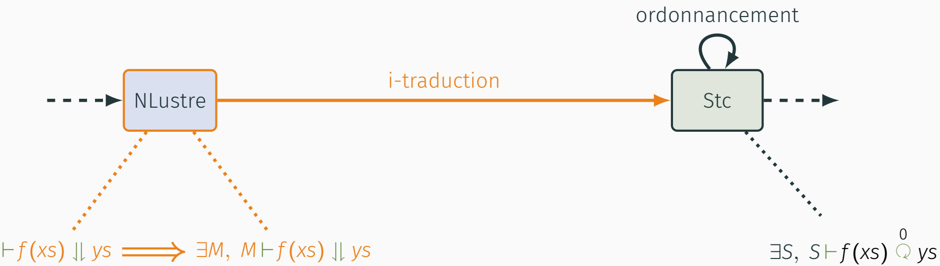
$$\dfrac{S, S' \vdash f(xs_n) \downdownarrows ys_n \qquad \dfrac{S', S'' \vdash f(xs_{n+1}) \downdownarrows ys_{n+1} \qquad \dfrac{S'', S''' \vdash f(xs_{n+2}) \downdownarrows ys_{n+2} \qquad \dfrac{\vdots}{S''' \vdash f(xs) \overset{n+3}{\circlearrowright} ys}}{S'' \vdash f(xs) \overset{n+2}{\circlearrowright}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

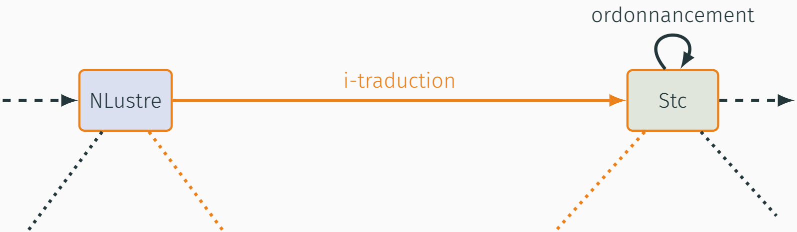
Correction : Préservation de la Sémantique



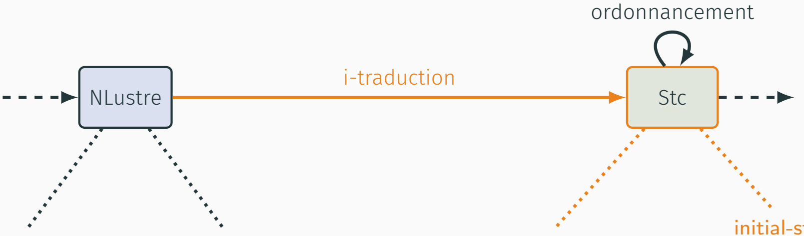
Correction : Préservation de la Sémantique

Correction : Préservation de la Sémantique

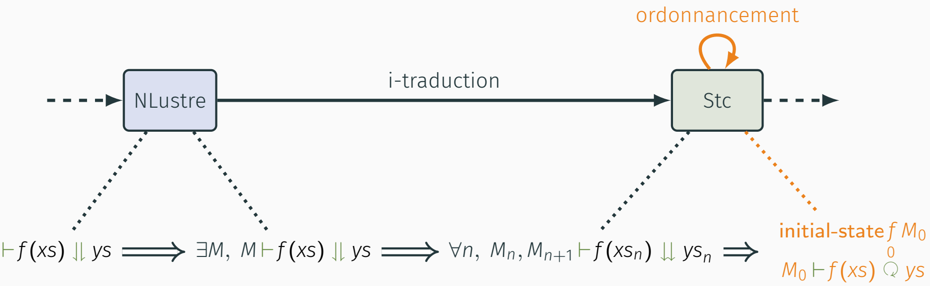




$\vdash f(xs) \Downarrow ys \implies \exists M,\ M \vdash f(xs) \Downarrow ys \implies \forall n,\ M_n, M_{n+1} \vdash f(xs_n) \Downarrow ys_n$ $\qquad \exists S,\ S \vdash f(xs) \overset{0}{\circlearrowleft} ys$



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


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

Production de Code Impératif : de Stc vers Obc

Stc

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

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

Obc

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

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

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

Stc

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

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

Obc

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

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

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

Stc

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

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

Obc

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

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

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

Stc

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

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

Obc

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

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

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

Stc

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

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

Obc

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

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

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

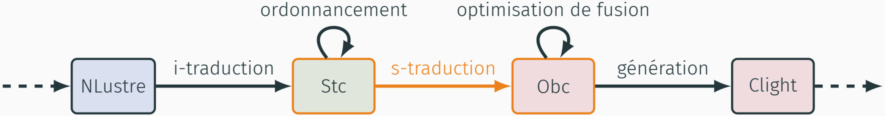
Stc

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

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

Obc

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

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

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

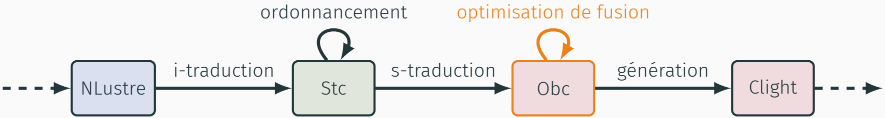
Compilation de l'Exemple du Reset




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

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

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

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

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

Compilation de l'Exemple du Reset




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

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

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

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

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

Expressions

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

Instructions

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

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

Expressions

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

Instructions

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

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

Expressions

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

Instructions

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

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

Expressions

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

Instructions

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

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

Expressions

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

Instructions

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

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

Règles Sémantiques

Expressions

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

Instructions

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

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

Expressions

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

Instructions

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

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

Expressions

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

Instructions

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

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

Expressions

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

Instructions

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

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

Expressions

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

Instructions

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

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

Expressions

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

Instructions

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

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

Expressions

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

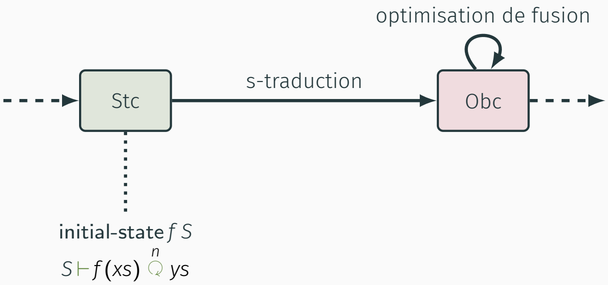
Instructions

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

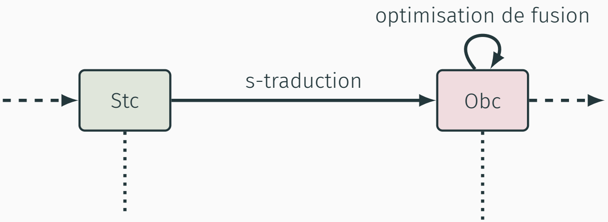
$$\frac{\begin{array}{c} me, ve \vdash s_1 \Downarrow (me_1, ve_1) \\ me_1, ve_1 \vdash s_2 \Downarrow (me_2, ve_2) \end{array}}{me, ve \vdash s_1 \; ; \; s_2 \Downarrow (me_2, ve_2)} \qquad \frac{me, ve \vdash \mathbf{e} \Downarrow \mathbf{v} \quad me[i] \vdash c.f(\mathbf{v}) \overset{\mathbf{w}}{\Downarrow}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

Exécution en boucle

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


optimisation de fusion
s-traduction
Stc
Obc
initial-state f S
S ⊢ f (xs) ↻ⁿ ys

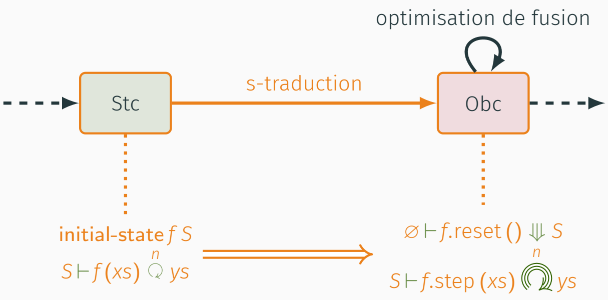
Correction

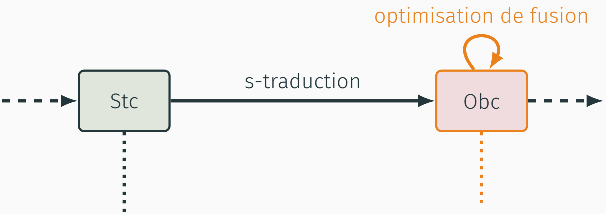


initial-state f S

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

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


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



$$\begin{array}{c} \textbf{initial-state}\ f\ S \\ S \vdash f\,(xs) \overset{n}{\circlearrowleft} ys \end{array} \Longrightarrow \begin{array}{c} \varnothing \vdash f.\mathrm{reset}\ () \Downarrow S \\ S \vdash f.\mathrm{step}\ (xs) \overset{n}{\circlearrowleft} ys \end{array}$$

Generation de code Clight

CompCert

Mécanisation en Coq de la syntaxe, de la sémantique et des algorithmes de compilation du langage C.

Clight

- langage intermédiaire de CompCert
- très proche de C
- opérations de bas niveau (adresses, structures, ...)

Obc

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

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

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

Clight

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

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

Obc

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

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

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

Clight

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

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

Obc

Clight

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

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

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

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

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

Obc

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

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

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

Clight

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

Obc

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

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

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

Clight

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

Boucle Principale

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

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

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

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

  fun$nav$reset(&self$);

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

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

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

Boucle Principale

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

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

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

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

  fun$nav$reset(&self$);

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

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

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

Boucle Principale

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

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

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

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

  fun$nav$reset(&self$);

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

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

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

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

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

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

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

  fun$nav$reset(&self$);

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

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

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

Boucle Principale

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

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

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

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

  fun$nav$reset(&self$);

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

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

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

Boucle Principale

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

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

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

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

  fun$nav$reset(&self$);

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

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

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

Boucle Principale

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

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

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

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

  fun$nav$reset(&self$);

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

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

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

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

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

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

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

  fun$nav$reset(&self$);

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

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

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

Boucle Principale

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

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

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

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

  fun$nav$reset(&self$);

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

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

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

Boucle Principale

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

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

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

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

  fun$nav$reset(&self$);

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

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

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

Modèle Sémantique de Clight

- modèle mémoire : blocs contigus
- variables et registres
- état sémantique (E, L, M)

  E environnement de variables : identifiants vers adresses mémoire

  L environnement de registres : identifiants vers valeurs

  M mémoire : adresses vers octets

Conséquences du modèle mémoire de CompCert

- *aliasing*
- alignement
- permissions
- tailles de types

Manipulation de structures et de pointeurs

Conséquences du modèle mémoire de CompCert

- *aliasing*
- alignement
- permissions
- tailles de types

Manipulation de structures et de pointeurs

Solution : utiliser des assertions de Logique de Séparation

Logique de Séparation

Une extension de la logique de Hoare pour raisonner sur des programmes qui manipulent des structures et des pointeurs.

Logique de Séparation

Une extension de la logique de Hoare pour raisonner sur des programmes qui manipulents des structures et des pointeurs.

Le prédicat $M \models P * Q$ stipule que M peut être partitionnée en deux zones distinctes sur lesquelles P et Q sont vraies respectivement.

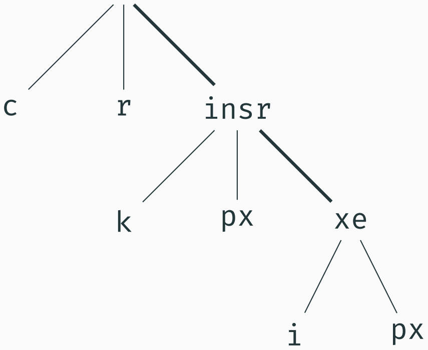
Logique de Séparation

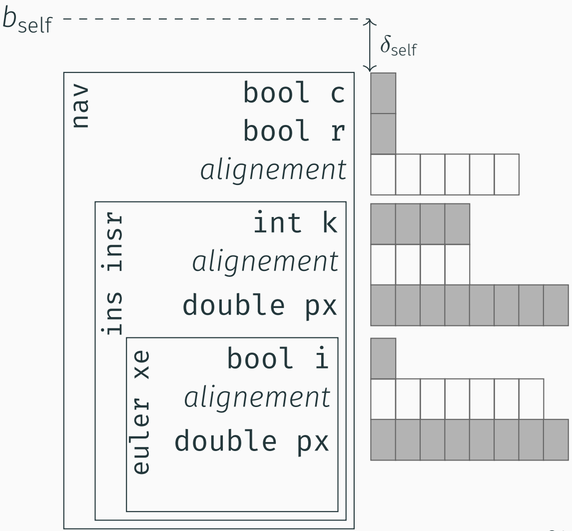
Une extension de la logique de Hoare pour raisonner sur des programmes qui manipulents des structures et des pointeurs.

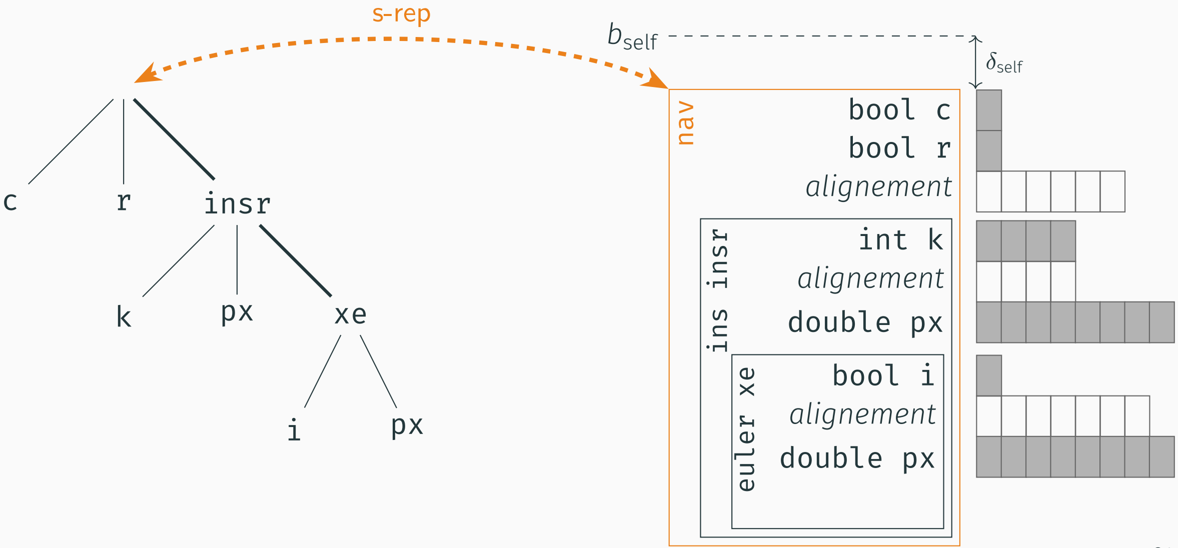
Le prédicat $M \vDash P * Q$ stipule que M peut être partitionnée en deux zones distinctes sur lesquelles P et Q sont vraies respectivement.

CompCert utilise déjà une librairie légère de Logique de Séparation pour l'une de ses passes.

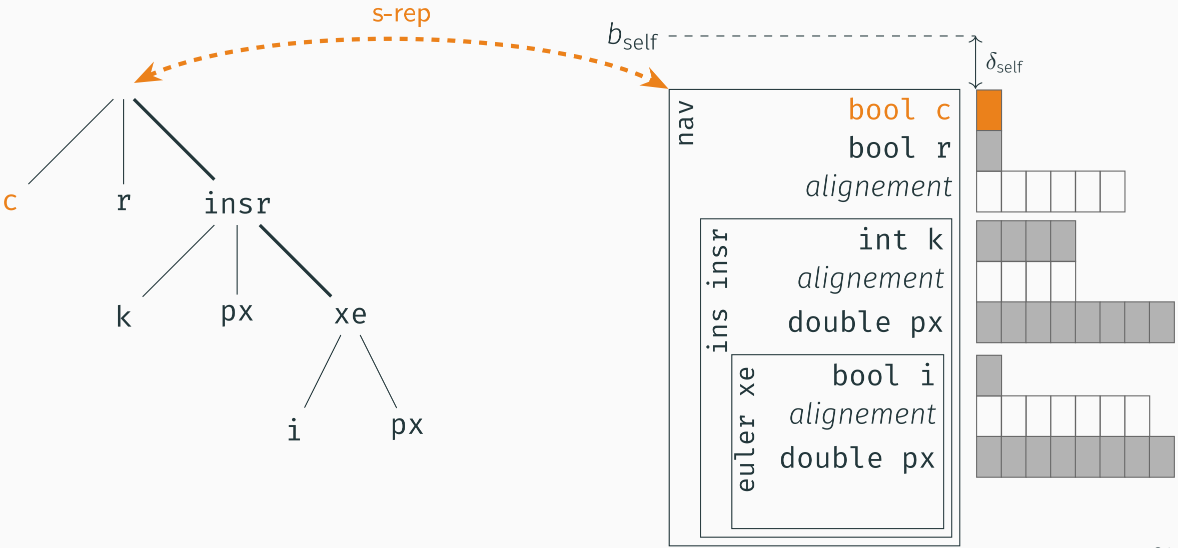
Prédicat de Correspondance d'État

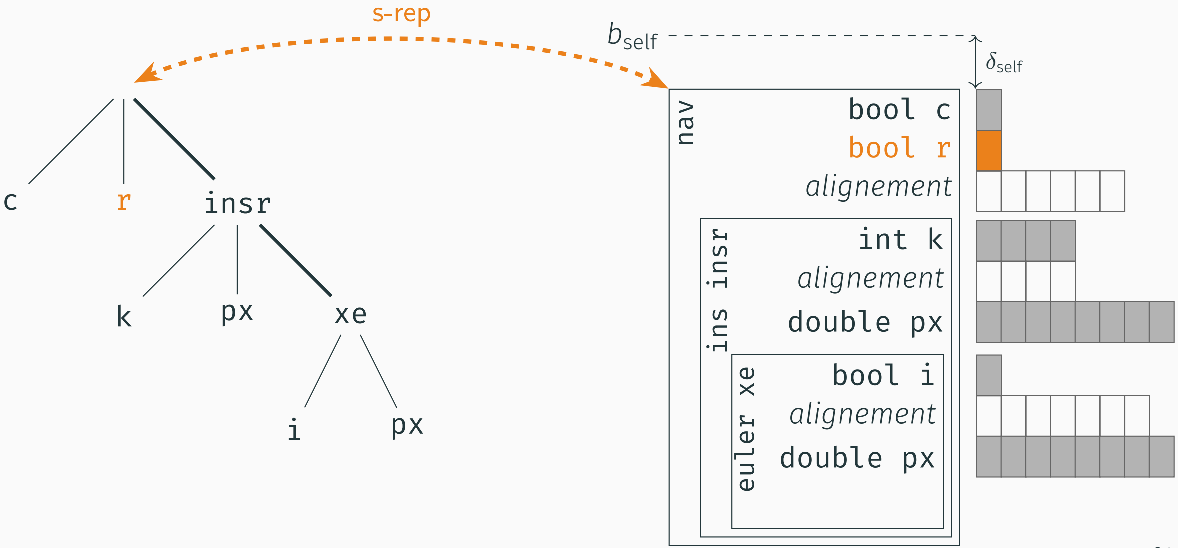


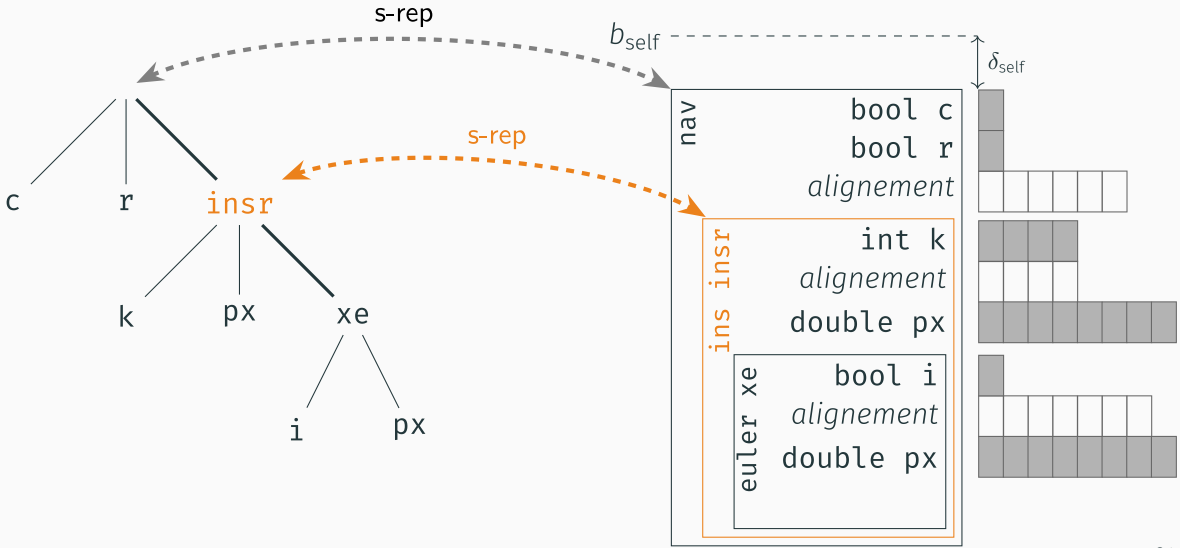



s-rep
b_{self}
δ_{self}
c
r
insr
k
px
xe
i
px
nav
bool c
bool r
alignement
ins insr
int k
alignement
double px
euler xe
bool i
alignement
double px

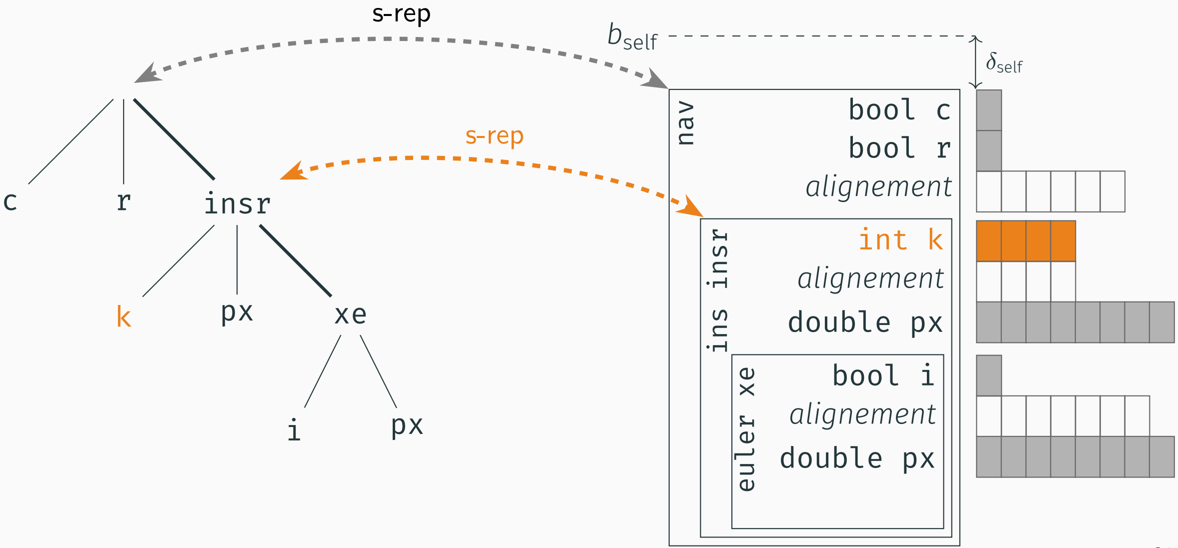
Prédicat de Correspondance d'État



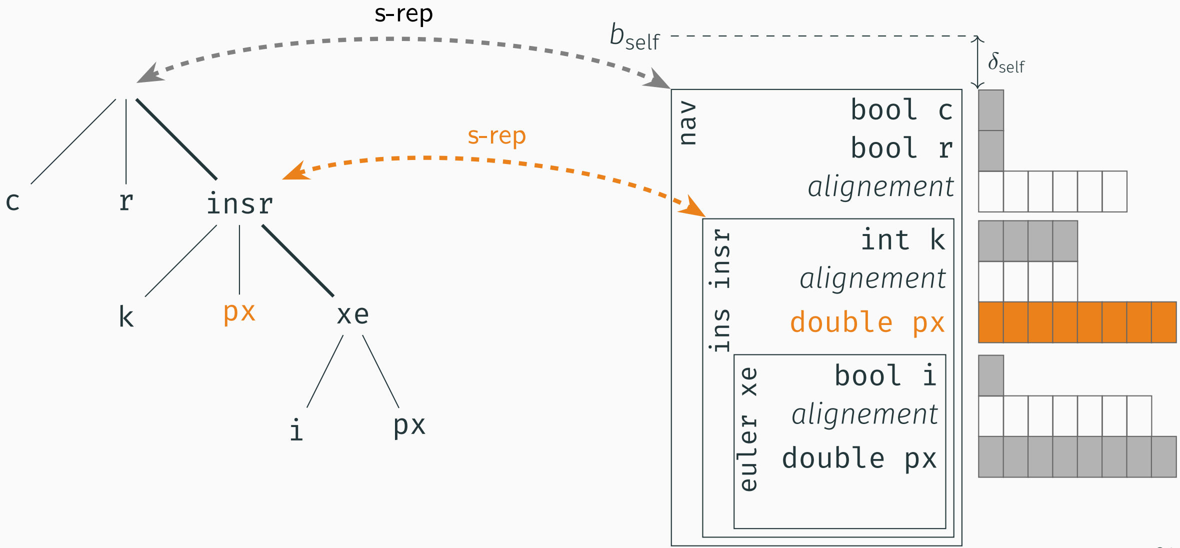

s-rep
c
r
insr
k
px
xe
i
px
b_{self}
δ_{self}
nav
bool c
bool r
alignement
ins insr
int k
alignement
double px
euler xe
bool i
alignement
double px


s-rep
b_{self}
δ_{self}
s-rep
c
r
insr
k
px
xe
i
px
nav
bool c
bool r
alignement
ins insr
int k
alignement
double px
euler xe
bool i
alignement
double px

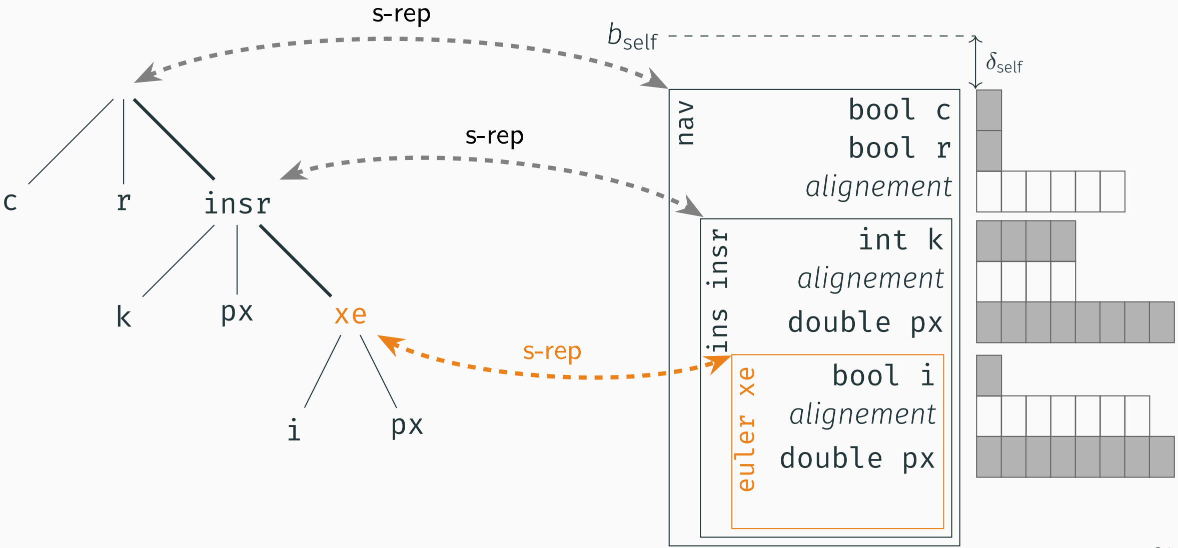
Prédicat de Correspondance d'État



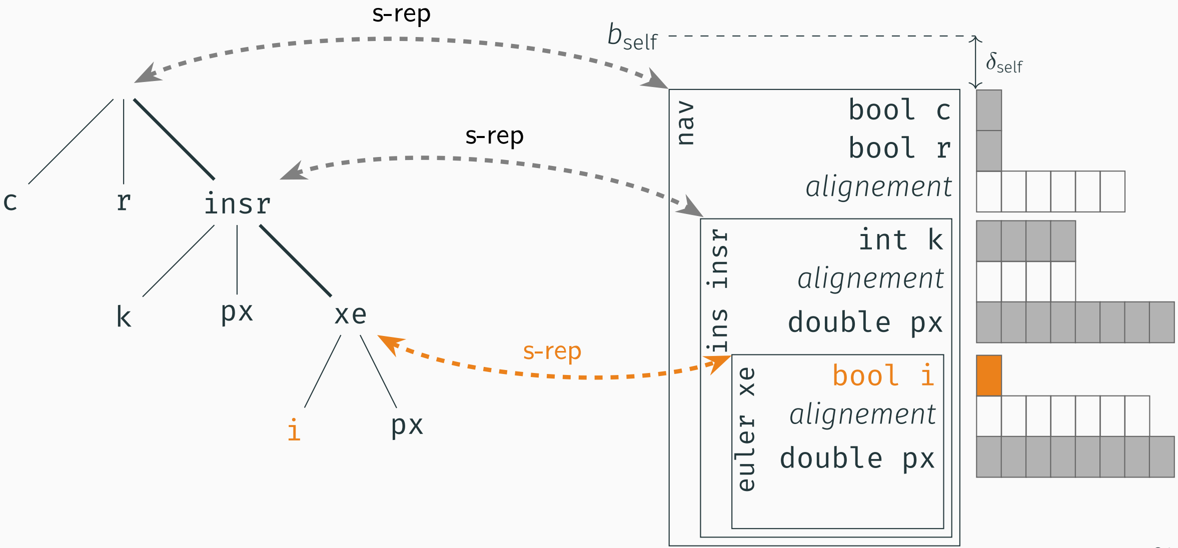
Prédicat de Correspondance d'État



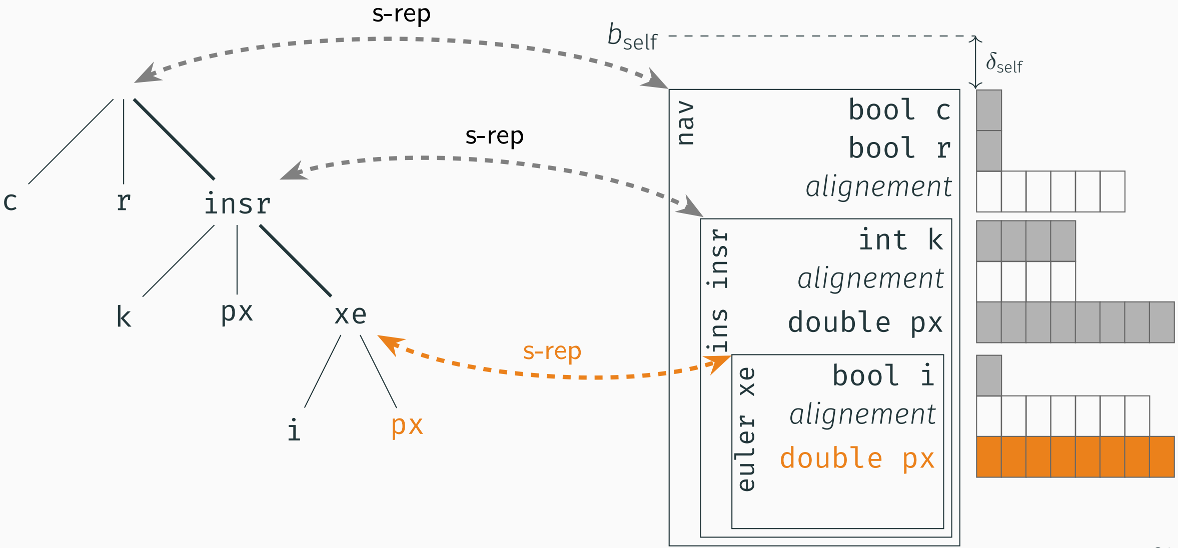
Prédicat de Correspondance d'État

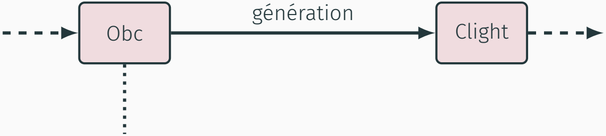


Prédicat de Correspondance d'État



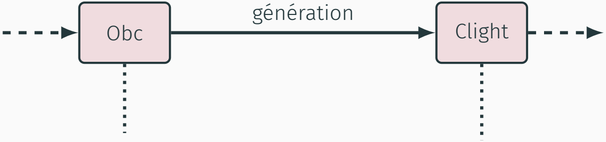
Prédicat de Correspondance d'État





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

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

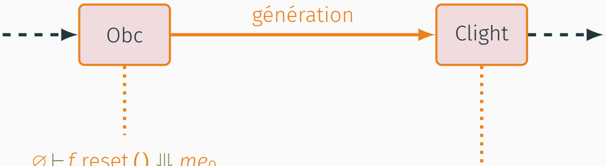


$\varnothing \vdash f.\mathrm{reset}\,() \Downarrow me_0$

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

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

T et (xs, ys) sont « équivalents »



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

Correction de Vélus

Théorème (correction de Vélus)

Soient une liste de déclarations D, un nom f, deux listes de flots de valeurs $\boldsymbol{xs}$ *et* $\boldsymbol{ys}$*, un programme NLustre G et un programme Assembleur P tels que* $\mathsf{compile}\, D\, f = \mathsf{OK}\,(G, P)$ *et* $G \vdash f\,(\boldsymbol{xs}) \Downarrow \boldsymbol{ys}$*, alors, il existe une trace infinie d'événements T telle que*

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

Théorème (correction de Vélus)

Soient une liste de déclarations D, un nom f, deux listes de flots de valeurs $\boldsymbol{xs}$ *et* $\boldsymbol{ys}$, *un programme NLustre G et un programme Assembleur P tels que* $\mathsf{compile}\, D\, f = \mathsf{OK}\,(G, P)$ *et* $G \vdash f\,(\boldsymbol{xs}) \Downarrow \boldsymbol{ys}$, *alors, il existe une trace infinie d'événements T telle que*

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

Théorème (correction de Vélus)

Soient une liste de déclarations D, un nom f, deux listes de flots de valeurs $\mathbf{xs}$ *et* $\mathbf{ys}$*, un programme NLustre G et un programme Assembleur P tels que* $\mathsf{compile}\, D\, f = \mathsf{OK}\,(G, P)$ *et* $G \vdash f(\mathbf{xs}) \Downarrow \mathbf{ys}$*, alors, il existe une trace infinie d'événements T telle que*

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

Théorème Final

Théorème (correction de Vélus)

Soient une liste de déclarations D, un nom f, deux listes de flots de valeurs $\mathbf{xs}$ *et* $\mathbf{ys}$*, un programme NLustre G et un programme Assembleur P tels que* $\mathsf{compile}\,D\,f = \mathsf{OK}\,(G, P)$ *et* $G \vdash f\,(\mathbf{xs}) \Downarrow \mathbf{ys}$*, alors, il existe une trace infinie d'événements T telle que*

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

Théorème (correction de Vélus)

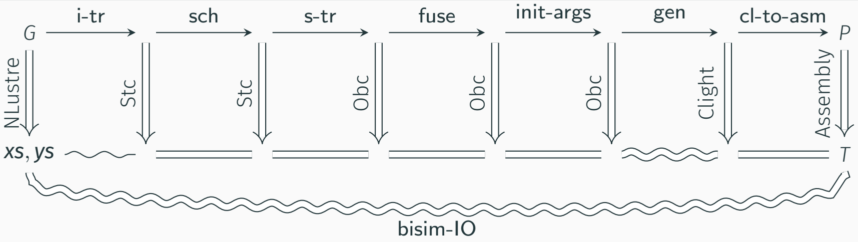
Soient une liste de déclarations D, un nom f, deux listes de flots de valeurs $\mathbf{xs}$ *et* $\mathbf{ys}$, *un programme NLustre G et un programme Assembleur P tels que* $\mathsf{compile}\, D\ f = \mathsf{OK}\,(G, P)$ *et* $G \vdash f\,(\mathbf{xs}) \Downarrow \mathbf{ys}$, *alors, il existe une trace infinie d'événements T telle que*

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

Théorème (correction de Vélus)

Soient une liste de déclarations D, un nom f, deux listes de flots de valeurs **xs** *et* **ys**, *un programme NLustre G et un programme Assembleur P tels que* compile $D\ f = \mathrm{OK}\,(G, P)$ *et* $G \vdash f\,(\mathbf{xs}) \Downarrow \mathbf{ys}$, *alors, il existe une trace infinie d'événements T telle que*

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



Conclusion

Résumé

- Un compilateur vérifié Lustre vers Assembleur
- Une seule règle sémantique pour le *reset*
- Un langage de systèmes de transitions intermédiaire : Stc

`velus.inria.fr`
`github.com/INRIA/velus`

Futur

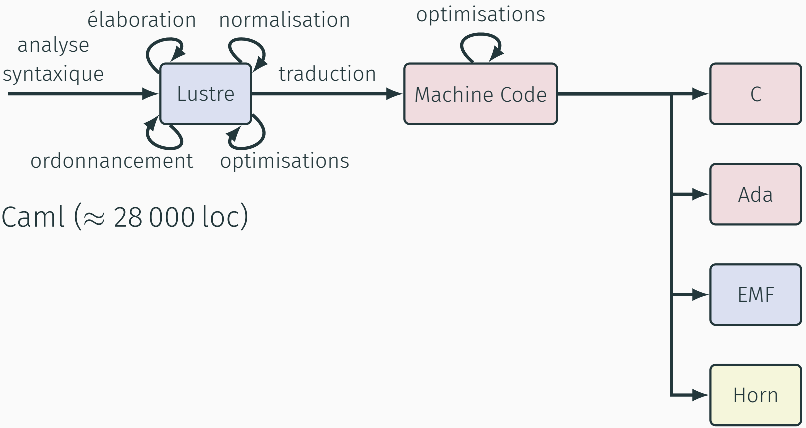
- Normalisation (fait !)
- Machines à états (en cours !)
- *Raffinement*
- Optimisations

Perspectives et discussion

- 42 000 loc et 3% de code fonctionnel
- Extensibilité
- Maintenance
- Axiomes
- Applicabilité industrielle

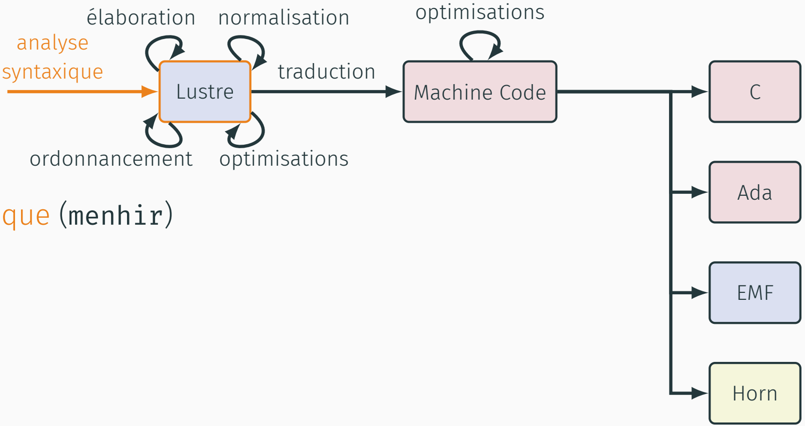
LustreC et compilation certifiante

LustreC : un Compilateur Lustre



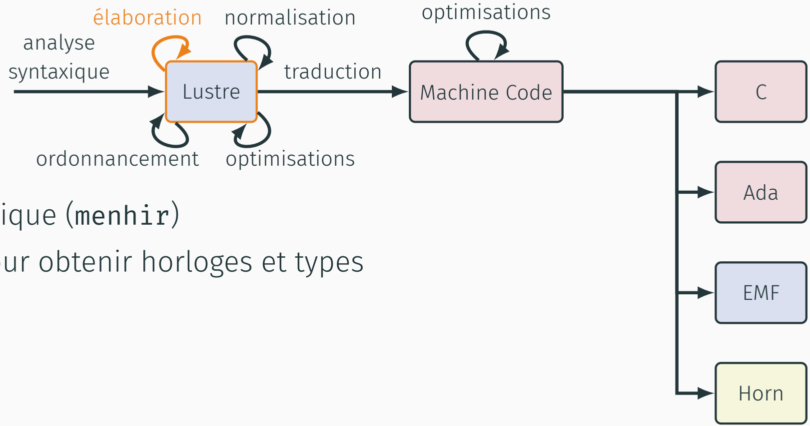
Implémenté en OCaml (≈ 28 000 loc)

LustreC : un Compilateur Lustre



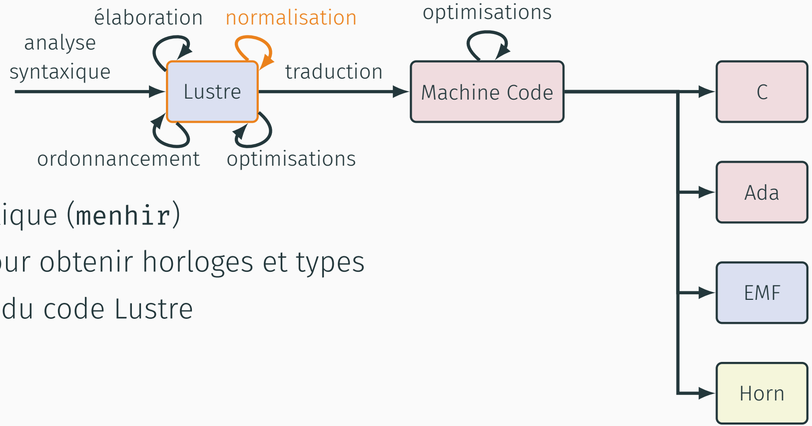
- analyse syntaxique (`menhir`)

LustreC : un Compilateur Lustre

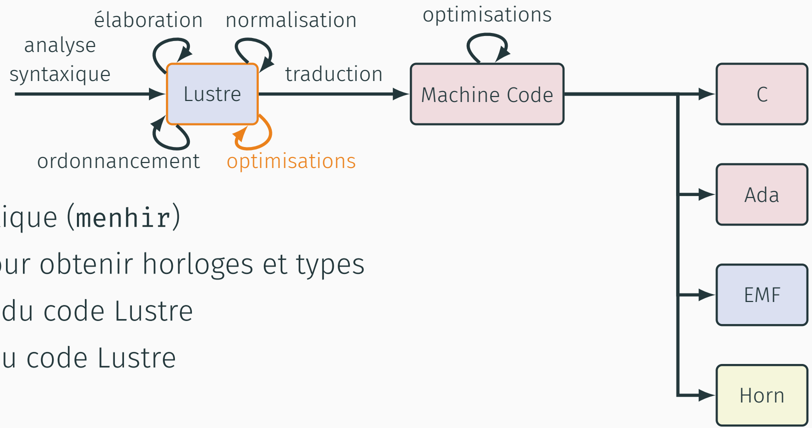


- analyse syntaxique (`menhir`)
- élaboration pour obtenir horloges et types

LustreC : un Compilateur Lustre

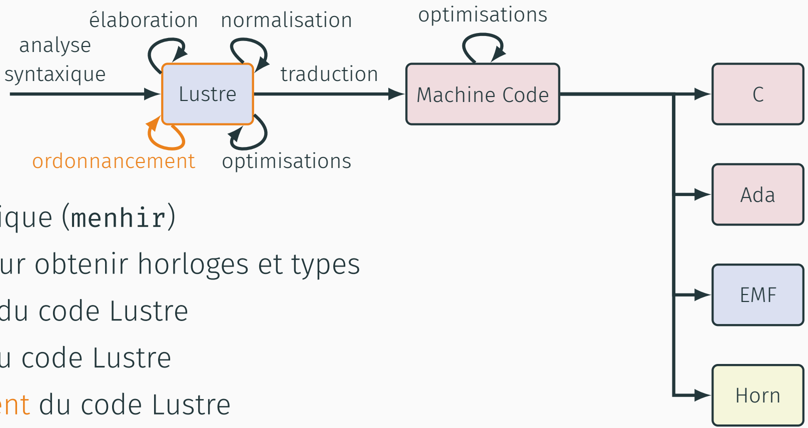


- analyse syntaxique (`menhir`)
- élaboration pour obtenir horloges et types
- normalisation du code Lustre



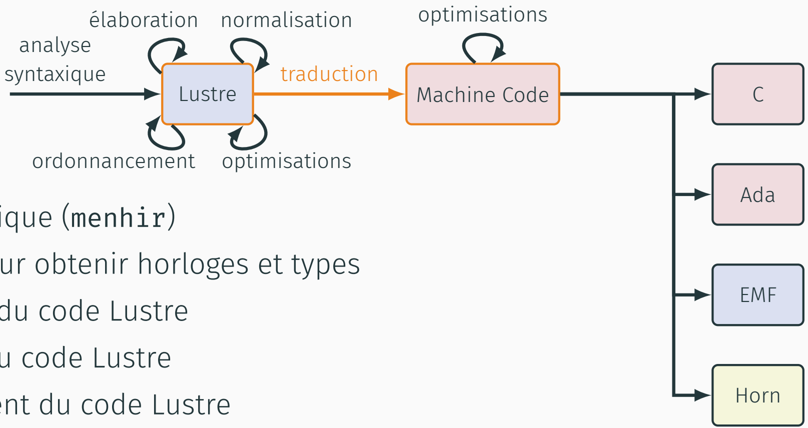
- analyse syntaxique (`menhir`)
- élaboration pour obtenir horloges et types
- normalisation du code Lustre
- optimisation du code Lustre

LustreC : un Compilateur Lustre



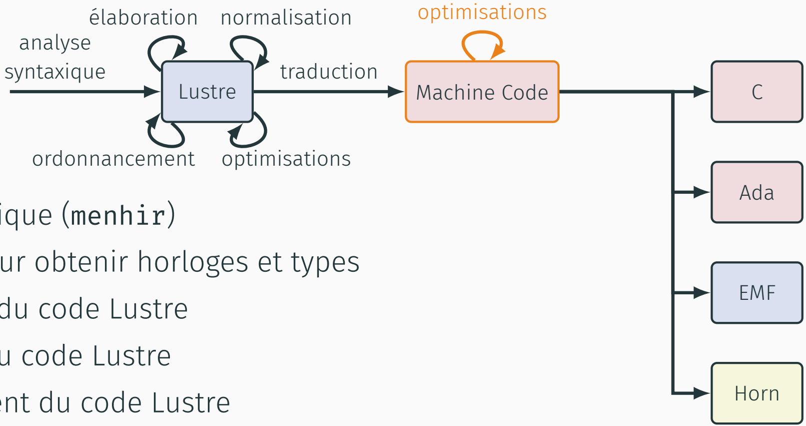
- analyse syntaxique (`menhir`)
- élaboration pour obtenir horloges et types
- normalisation du code Lustre
- optimisation du code Lustre
- ordonnancement du code Lustre

LustreC : un Compilateur Lustre



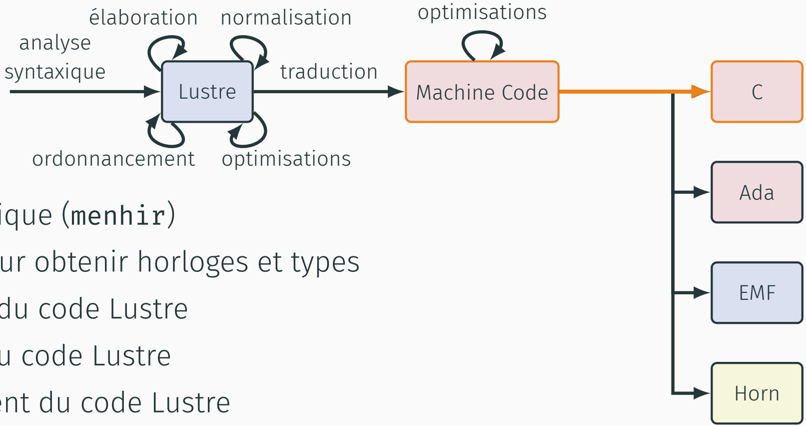
- analyse syntaxique (`menhir`)
- élaboration pour obtenir horloges et types
- normalisation du code Lustre
- optimisation du code Lustre
- ordonnancement du code Lustre
- traduction vers le Machine Code

LustreC : un Compilateur Lustre



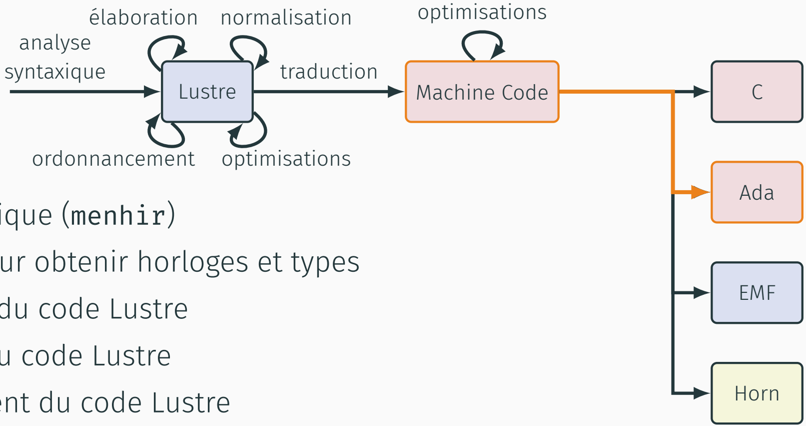
- analyse syntaxique (`menhir`)
- élaboration pour obtenir horloges et types
- normalisation du code Lustre
- optimisation du code Lustre
- ordonnancement du code Lustre
- traduction vers le Machine Code
- optimisations du Machine Code

LustreC : un Compilateur Lustre

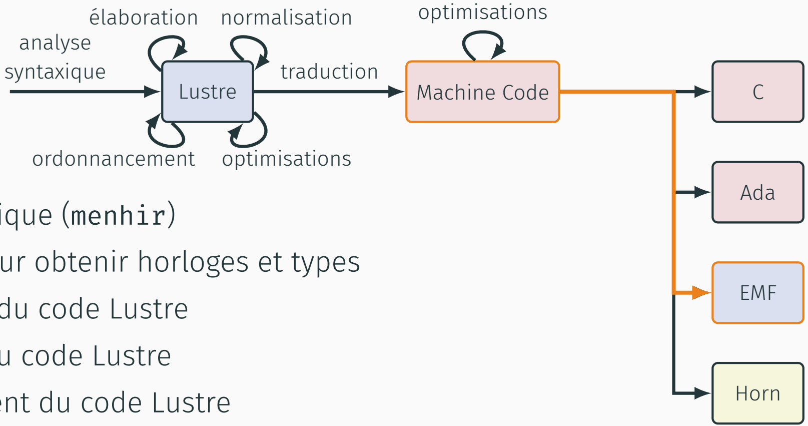


- analyse syntaxique (`menhir`)
- élaboration pour obtenir horloges et types
- normalisation du code Lustre
- optimisation du code Lustre
- ordonnancement du code Lustre
- traduction vers le Machine Code
- optimisations du Machine Code
- génération de code

LustreC : un Compilateur Lustre

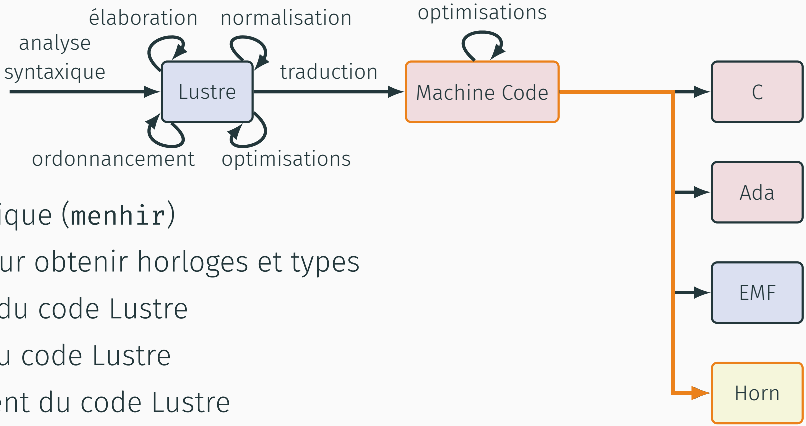


- analyse syntaxique (`menhir`)
- élaboration pour obtenir horloges et types
- normalisation du code Lustre
- optimisation du code Lustre
- ordonnancement du code Lustre
- traduction vers le Machine Code
- optimisations du Machine Code
- génération de code



- analyse syntaxique (`menhir`)
- élaboration pour obtenir horloges et types
- normalisation du code Lustre
- optimisation du code Lustre
- ordonnancement du code Lustre
- traduction vers le Machine Code
- optimisations du Machine Code
- génération de code

LustreC : un Compilateur Lustre



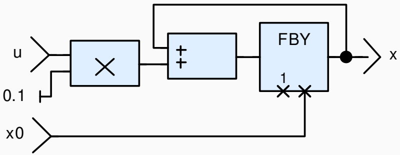
- analyse syntaxique (`menhir`)
- élaboration pour obtenir horloges et types
- normalisation du code Lustre
- optimisation du code Lustre
- ordonnancement du code Lustre
- traduction vers le Machine Code
- optimisations du Machine Code
- génération de code

Objectif : Compilateur Certifiant

- Génération de spécification ACSL
- Encodage du résultat de correction
- À terme : transport automatique de spécification haut niveau

frama C
Software Analyzers

- Approche plus souple
- Autres *backends* envisageables (ex. SPARK / Ada)
- Incomparables en termes de fonctionnalités (2K loc vs 28K loc)
- SCADE vs Simulink
- Transport de spécifications et vérification




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

Exemple

```c
struct euler {
  bool i;
  double px;
};
```

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

void _arrow_reset(struct _arrow_mem *self) {
  self->_reg._first = true;
  return;
}

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

Exemple

```c
struct euler {
  bool i;
  double px;
};
```

```c
struct euler_mem {
  _Bool euler_reset;
  struct euler_reg {double __euler_2;} _reg;
  struct _arrow_mem *ni_9;
};
```

Exemple

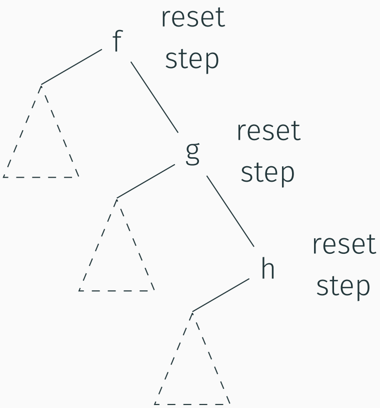
```
void fun$euler$reset(struct euler *self) {
  self->i = true;
  self->px = 0;
  return;
}
```

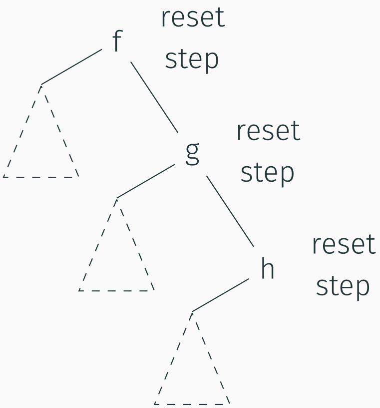
```
void euler_set_reset(struct euler_mem *self) {
  self->euler_reset = 1;
  return;
}

void euler_clear_reset(struct euler_mem *self) {
  if (self->euler_reset) {
    self->euler_reset = 0;
    _arrow_reset(self->ni_9);
  }
  return;
}
```

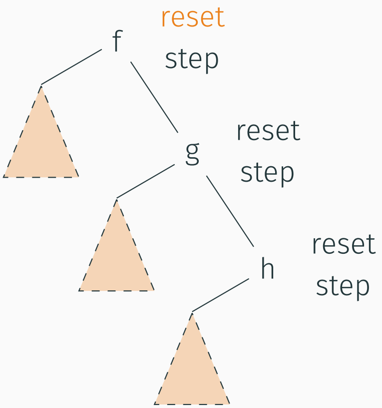
```c
double fun$euler$step(struct euler *self,
                      double x0, double u) {
  register double x;
  if (self->i) {
    x = x0;
  } else {
    x = self->px;
  }
  self->i = false;
  self->px = x + 0.10000000000000006 * u;
  return x;
}
```

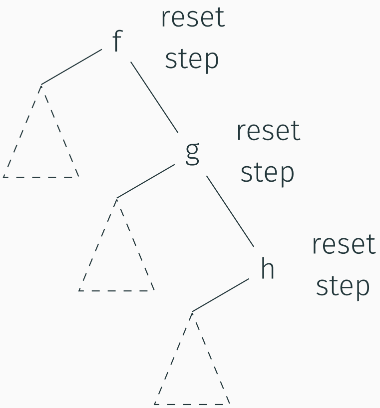
```c
void euler_step(double x0, double u,
                double (*x),
                struct euler_mem *self) {
  _Bool __euler_1;
  euler_clear_reset(self);
  __euler_1 = _arrow_step(self->ni_9);
  if (__euler_1) {
    *x = x0;
  } else {
    *x = self->_reg.__euler_2;
  }
  self->_reg.__euler_2 = (*x + (0.1 * u));
  return;
}
```

Compilation du reset

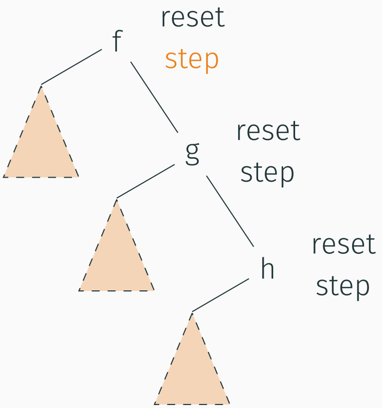


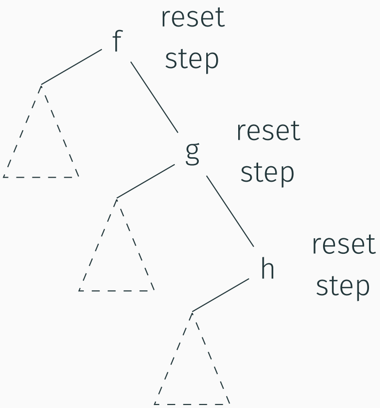


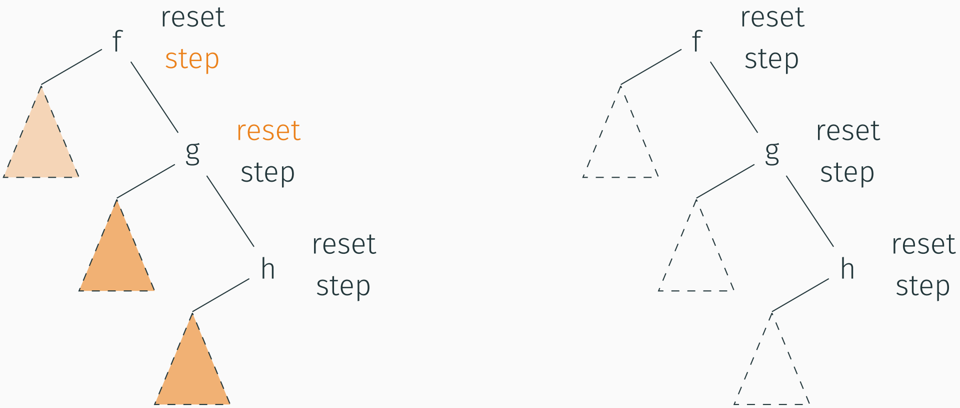

reset
f
step
reset
g
step
reset
h
step


reset
f
step
reset
g
step
reset
h
step

Compilation du reset

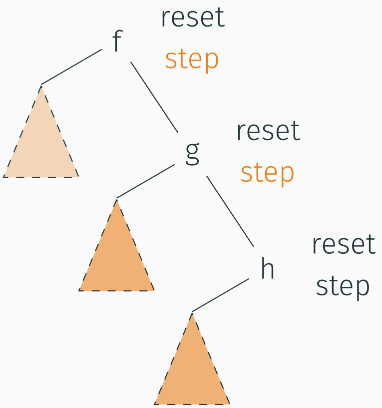


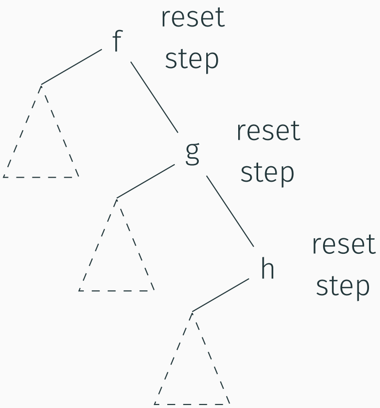


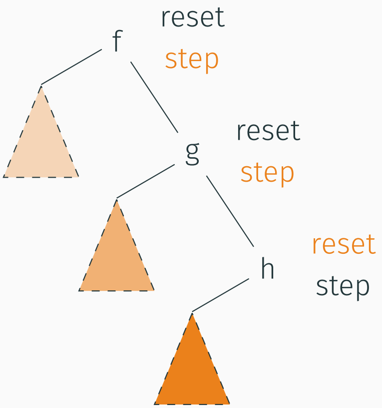


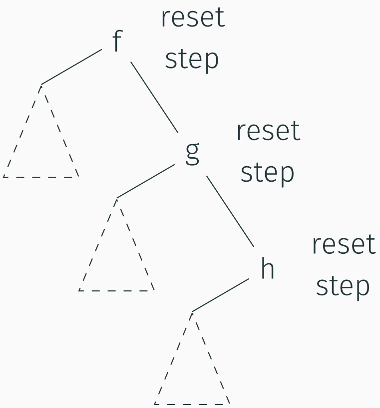
f
reset
step
g
reset
step
h
reset
step
f
reset
step
g
reset
step
h
reset
step

Compilation du reset

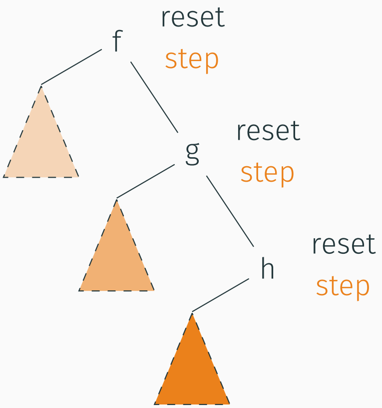


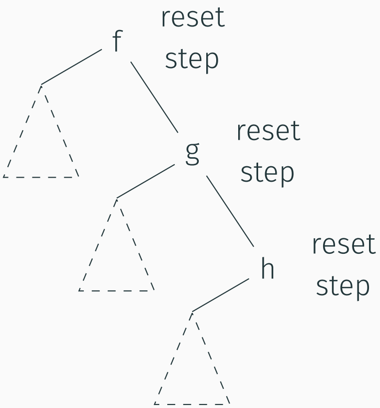



f
reset
step
g
reset
step
h
reset
step

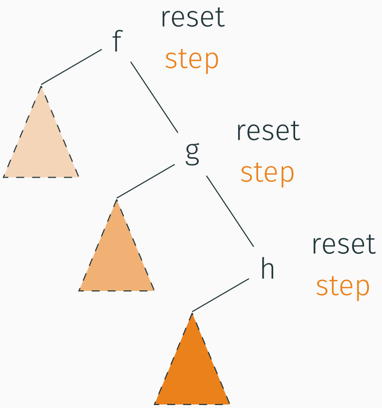

f
reset
step
g
reset
step
h
reset
step

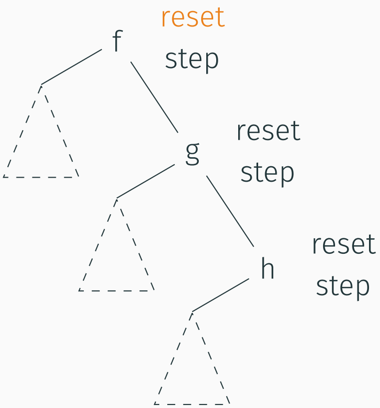
Compilation du reset



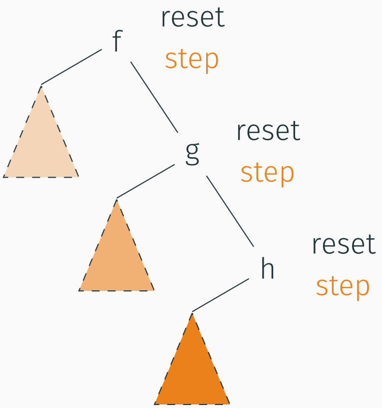


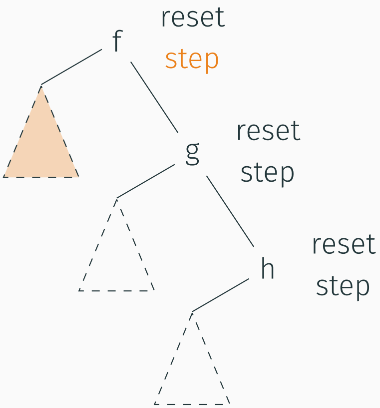
Compilation du reset

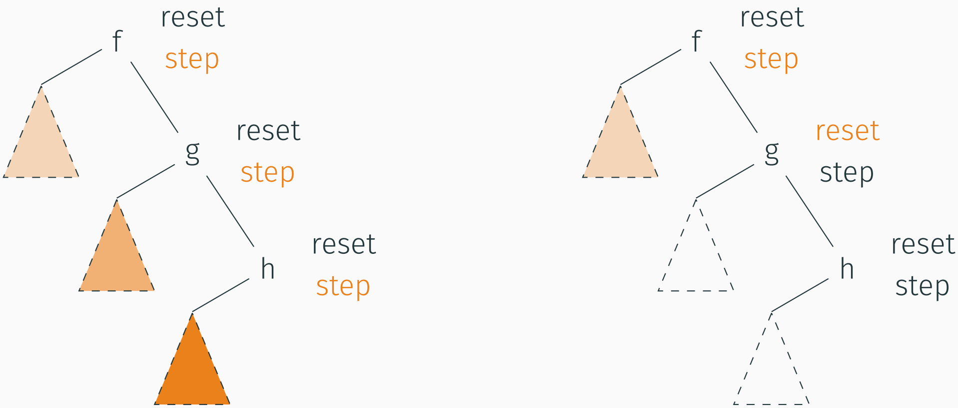




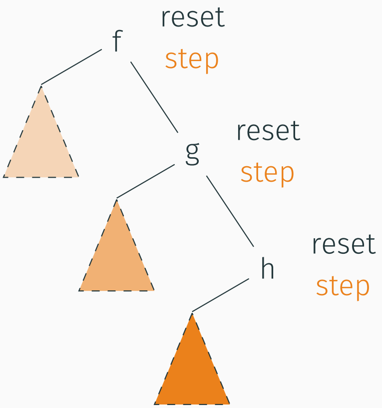
Compilation du reset

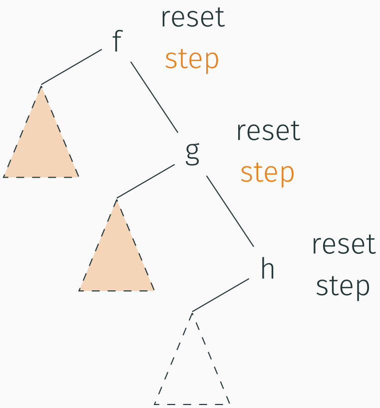




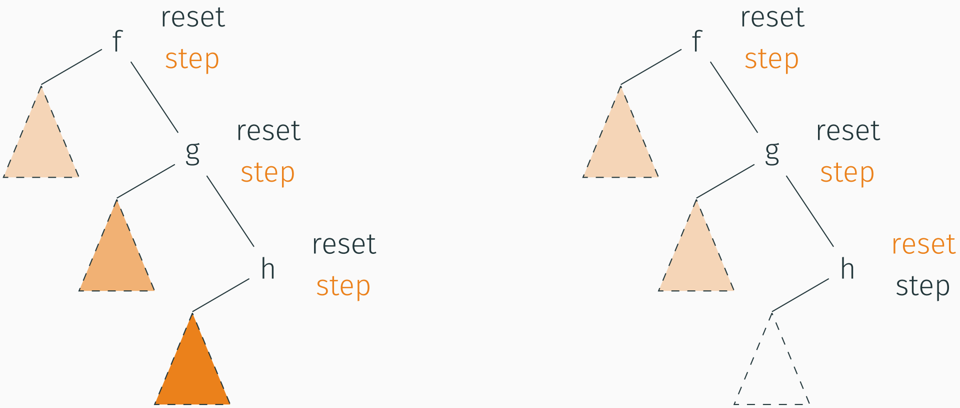

reset
step
f
reset
step
g
reset
step
h
reset
step
f
reset
step
g
reset
step
h

Compilation du reset

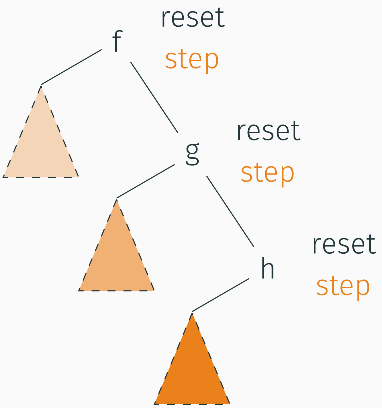


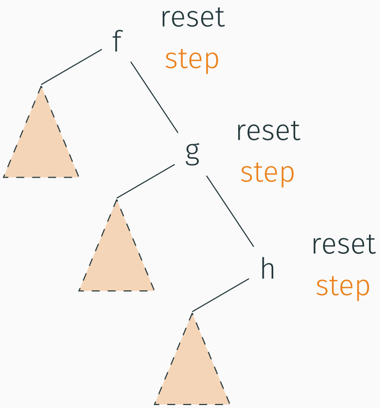


Compilation du reset



Compilation du reset





Principe de la spécification

Prédicats ACSL

- correspondance d'état (mémoire fantôme, principe de `s-rep`)
- relations de transition

Exemple avec Spécification

```
/*@ requires \separated(self, mem, self->ni_9, x);
    requires euler_ghost(*mem, self);
    ensures euler_ghost(*mem, self);
    ensures euler_transition(\old(*mem), x0, u, *mem, *x);
    assigns *x;
    assigns self->_reg.__euler_2;
    assigns self->ni_9->_reg._first;
    assigns self->euler_reset;
    assigns mem->_reg.__euler_2;
    assigns mem->ni_9._reg._first;
    assigns mem->euler_reset;
*/
void euler_step(double x0, double u,
                 double (*x),
                 struct euler_mem *self)
      /*@ ghost (struct euler_mem_ghost \ghost *mem) */ {
```

Exemple avec Spécification

```
  _Bool __euler_1;
  euler_clear_reset(self) /*@ ghost (mem) */;
  //@ assert euler_ghost0(*mem, self);
  //@ assert euler_transition0(*mem, x0, u, *mem);
  Reset:

  __euler_1 = _arrow_step(self->ni_9) /*@ ghost (&mem->ni_9) */;
  //@ assert euler_ghost1(*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_ghost2(*mem, self);
  //@ assert euler_transition2(\at(*mem, Reset), x0, u, *mem, *x);

  self->_reg.__euler_2 = (*x + (0.1 * u));
  //@ ghost mem->_reg.__euler_2 = (*x + (0.1 * u));
  //@ assert euler_ghost3(*mem, self);
  //@ assert euler_transition3(\at(*mem, Reset), x0, u, *mem, *x);

  return;
}
```

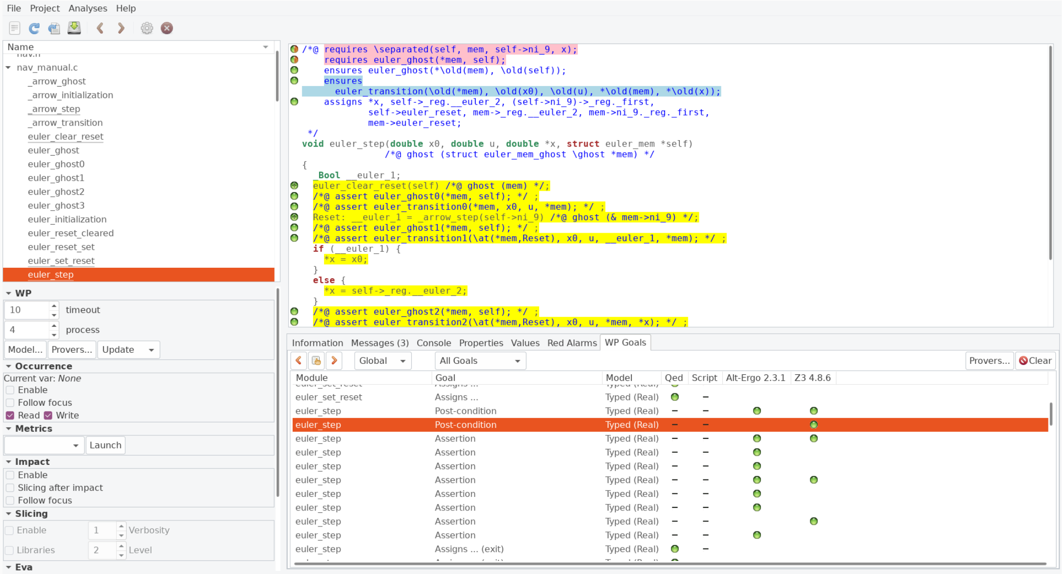
Exemple avec Spécification

```
/*@ ghost struct euler_mem_ghost {
          _Bool euler_reset;
          struct euler_reg _reg;
          struct _arrow_mem_ghost ni_9;
   };
*/
/*@ predicate euler_ghost0(struct euler_mem_ghost mem, struct euler_mem *self) =
      mem.euler_reset == self->euler_reset && mem.euler_reset == \false;
*/
/*@ predicate euler_ghost1(struct euler_mem_ghost mem, struct euler_mem *self) =
      euler_ghost0(mem, self)
      && _arrow_ghost(mem.ni_9, self->ni_9);
*/
/*@ predicate euler_ghost2(struct euler_mem_ghost mem, struct euler_mem *self) =
      euler_ghost1(mem, self);
*/
/*@ predicate euler_ghost3(struct euler_mem_ghost mem, struct euler_mem *self) =
      euler_ghost2(mem, self)
      && (!_arrow_initialization(mem.ni_9) ==> mem._reg.__euler_2 == self->_reg.__euler_2);
*/
/*@ predicate euler_ghost(struct euler_mem_ghost mem, struct euler_mem *self) =
      mem.euler_reset ? (mem.euler_reset == self->euler_reset) : (euler_ghost3(mem, self));
*/
```

Exemple avec Spécification

```
/*@ predicate euler_transition0(struct euler_mem_ghost mem_in,
                                double x0, double u,
                                struct euler_mem_ghost mem_out) =
      \true;
*/
/*@ predicate euler_transition1(struct euler_mem_ghost mem_in,
                                double x0, double u, _Bool __euler_1,
                                struct euler_mem_ghost mem_out) =
      euler_transition0(mem_in, x0, u, mem_out)
      && _arrow_transition(mem_in.ni_9, mem_out.ni_9, __euler_1);
*/
/*@ predicate euler_transition2(struct euler_mem_ghost mem_in,
                                double x0, double u,
                                struct euler_mem_ghost mem_out, double x) =
      \exists _Bool __euler_1;
        euler_transition1(mem_in, x0, u, __euler_1, mem_out)
        && (__euler_1 ? (x == x0) : (x == mem_in._reg.__euler_2));
*/
/*@ predicate euler_transition3(struct euler_mem_ghost mem_in,
                                double x0, double u,
                                struct euler_mem_ghost mem_out, double x) =
      euler_transition2(mem_in, x0, u, mem_out, x)
      && mem_out._reg.__euler_2 == (x + 0.1 * u);
*/
/*@ predicate euler_transition(struct euler_mem_ghost mem_in,
                               double x0, double u,
                               struct euler_mem_ghost mem_out, double x) =
      \exists struct euler_mem_ghost mem_reset;
        euler_reset_cleared(mem_in, mem_reset)
        && euler_transition3(mem_reset, x0, u, mem_out, x);
*/
```

Frama-C



File Project Analyses Help
Name
nav_manual.c
_arrow_ghost
_arrow_initialization
_arrow_step
_arrow_transition
euler_clear_reset
euler_ghost
euler_ghost0
euler_ghost1
euler_ghost2
euler_ghost3
euler_initialization
euler_reset_cleared
euler_reset_set
euler_set_reset
euler_step
WP
10 timeout
4 process
Model... Provers... Update
Occurrence
Current var: None
Enable
Follow focus
Read Write
Metrics
Launch
Impact
Enable
Slicing after impact
Follow focus
Slicing
Enable 1 Verbosity
Libraries 2 Level
Eva
/*@ requires \separated(self, mem, self->ni_9, x);
requires euler_ghost(*mem, self);
ensures euler_ghost(*\old(mem), \old(self));
ensures
euler_transition(\old(*mem), \old(x0), \old(u), *\old(mem), *\old(x));
assigns *x, self->_reg.__euler_2, (self->ni_9)->_reg._first,
self->euler_reset, mem->_reg.__euler_2, mem->ni_9._reg._first,
mem->euler_reset;
*/
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) */;
/*@ assert euler_ghost0(*mem, self); */ ;
/*@ assert euler_transition0(*mem, x0, u, *mem); */ ;
Reset: __euler_1 = _arrow_step(self->ni_9) /*@ ghost (& mem->ni_9) */;
/*@ assert euler_ghost1(*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_ghost2(*mem, self); */ ;
/*@ assert euler_transition2(\at(*mem,Reset), x0, u, *mem, *x); */ ;
Information Messages (3) Console Properties Values Red Alarms WP Goals
Global All Goals Provers... Clear
Module Goal Model Qed Script Alt-Ergo 2.3.1 Z3 4.8.6
euler_set_reset Assigns ... Typed (Real)
euler_step Post-condition Typed (Real)
euler_step Post-condition Typed (Real)
euler_step Assertion Typed (Real)
euler_step Assigns ... (exit) Typed (Real)

Conclusion

Résumé

- Un compilateur certifiant Lustre vers C / Frama-C
- Une compilation différente du *reset*
- Un raisonnement basé sur des simulations

Futur

- Prototype
- *reset* monolithique ?
- Optimisations
- Réels
- SPARK / Ada
- Transport de spécifications
- Comparaison avec la sortie SMT

Références I

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

Références II

- Jean-Louis Colaço, Bruno Pagano et 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, p. 173-182.
- Dariusz Biernacki, Jean-Louis Colaço, Gregoire Hamon et Marc Pouzet (2008). « Clock-Directed Modular Code Generation for Synchronous Data-Flow Languages ». In : *Proceedings of the 2008 ACM SIGPLAN-SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*. LCTES '08. New York, NY, USA : ACM, p. 121-130.
- Sandrine Blazy et Xavier Leroy (1er oct. 2009). « Mechanized Semantics for the Clight Subset of the C Language ». In : *Journal of Automated Reasoning* 43.3, p. 263-288.
- Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch et Simon Winwood (11 oct. 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, p. 207-220.

- Xavier Leroy (juill. 2009). « Formal Verification of a Realistic Compiler ». In : *Communications of the ACM* 52.7, p. 107-115.
- Jacques-Henri Jourdan, François Pottier et Xavier Leroy (2012). « Validating LR(1) Parsers ». In : *Programming Languages and Systems*. Sous la dir. d'Helmut Seidl. Lecture Notes in Computer Science. Springer Berlin Heidelberg, p. 397-416.
- Ramana Kumar, Magnus O. Myreen, Michael Norrish et Scott Owens (jan. 2014). « CakeML : A Verified Implementation of ML ». In : *Principles of Programming Languages (POPL)*. ACM Press, p. 179-191.
- Xavier Thirioux (19 sept. 2016). « Verifying Embedded Systems ». Habilitation. Institut National Polytechnique de Toulouse. 187 p.
- Timothy Bourke, Lélio Brun, Pierre-Évariste Dagand, Xavier Leroy, Marc Pouzet et Lionel Rieg (juin 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, p. 586-601.

- Jean-Louis Colaço, Bruno Pagano et 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)*, p. 1-11.
- Timothy Bourke, Lélio Brun et Marc Pouzet (jan. 2020). « Mechanized Semantics and Verified Compilation for a Dataflow Synchronous Language with Reset ». In : *Proceedings of the 47th ACM SIGPLAN Symposium on Principles of Programming Languages*. Principles Of Programming Languages. T. 4. POPL'20. New Orleans, LA, USA : Association for Computing Machinery, p. 29.