Sémantique Mécanisée et Compilation Vérifiée pour un Langage Synchrone à Flots de Données avec Réinitialisation

Prix de thèse du GDR GPL 2020

Lélio Brun[1]

15 juin 2021

[1]ISAE-SUPAERO – DISC – IpSC

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

Conception de Systèmes Embarqués

Systèmes embarqués

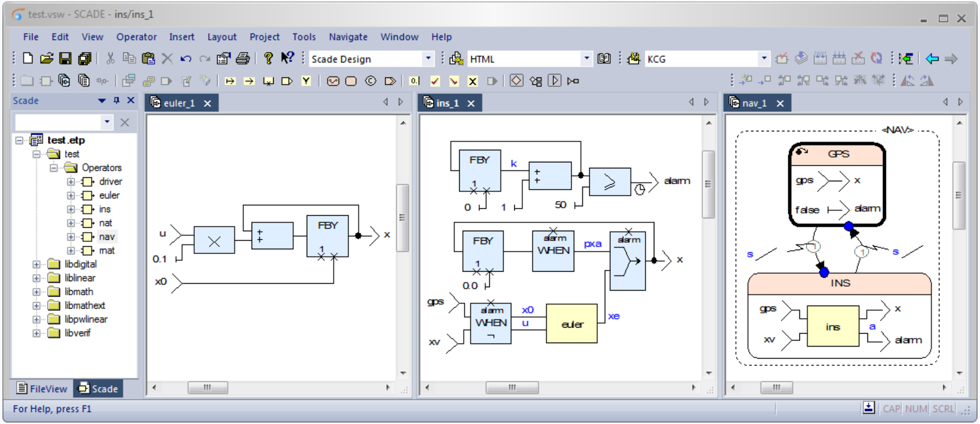
- systèmes informatiques au sein de systèmes physiques interagissant avec le monde réel, souvent sous des contraintes temps-réel
- logiciels habituellement développés avec des langages bas niveau : C, Ada, Assembleur

Model-Based Design

Spécifications abstraites de haut niveau exécutables

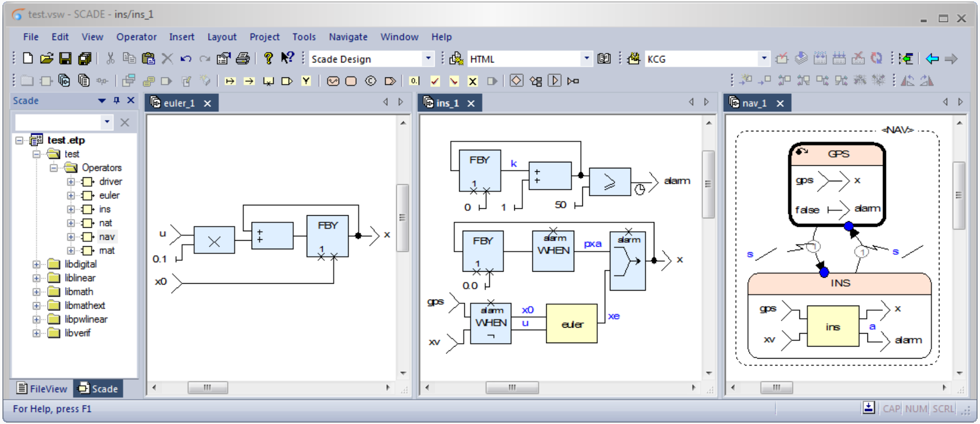
Model-Based Design dans SCADE Suite

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



Model-Based Design dans SCADE Suite

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

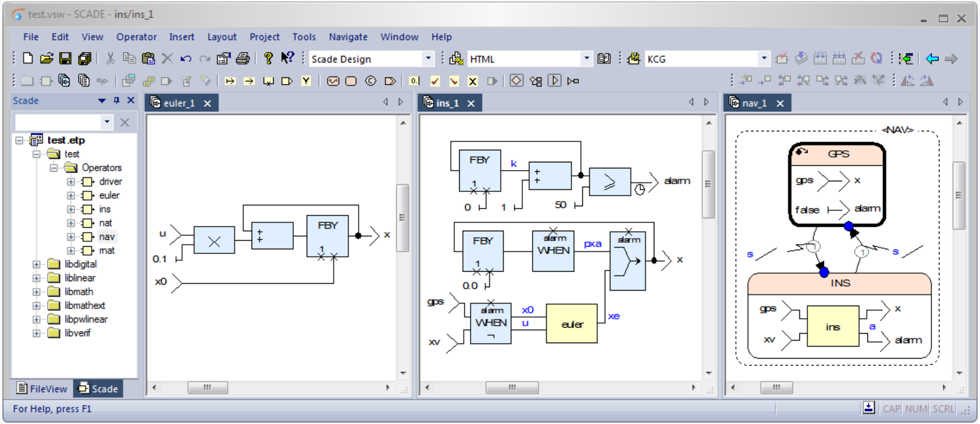


bloc / nœud = système

ligne = signal

Model-Based Design dans SCADE Suite

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

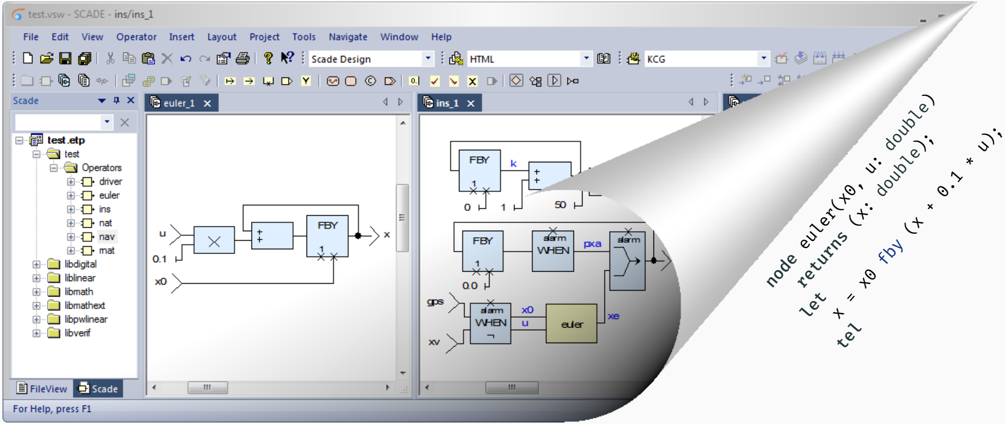


bloc / nœud = système = fonction de flots

ligne = signal = flot de valeurs

Model-Based Design dans SCADE Suite

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

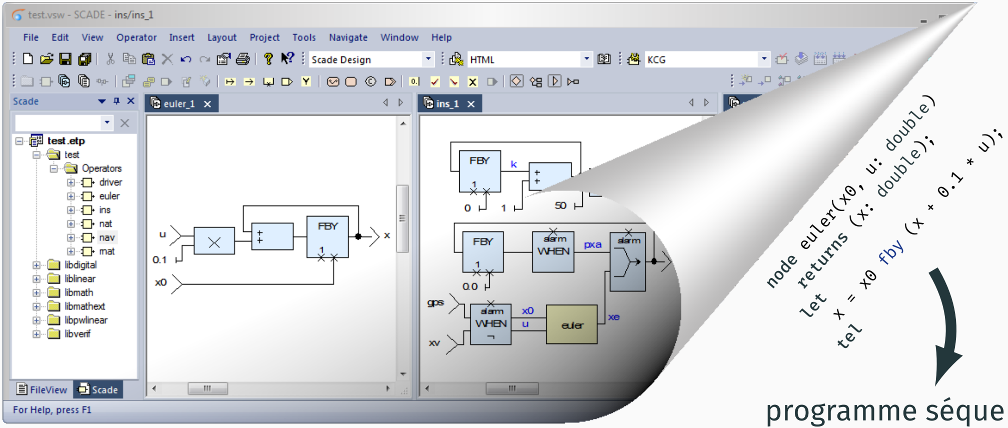


bloc / nœud = système = fonction de flots

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

Criticité

Systèmes qui ne doivent pas échouer

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

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, ...

Formalisation Mécanisée

Klein et al. 2009
Kumar, Myreen, Norrish et Owens 2014
Leroy 2009

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.

Caspi, Pilaud, Halbwachs et Plaice 1987
Halbwachs, Caspi, Raymond et Pilaud 1991
Colaço, Pagano et Pouzet 2017

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

Caspi, Pilaud, Halbwachs et Plaice 1987
Halbwachs, Caspi, Raymond et Pilaud 1991
Colaço, Pagano et Pouzet 2017

Langages pour le Model-Based Design
Scade 6, Lustre

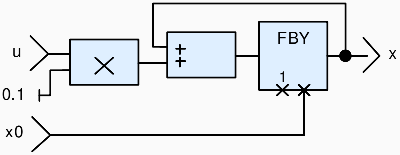
+

Assistants de Preuve
Coq

Défis

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

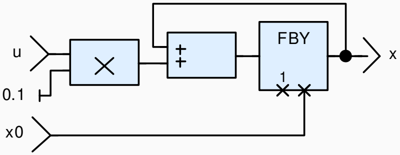
Focus : réinitialisation modulaire (*modular reset*)




```
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	$\cdots$

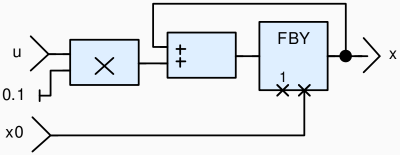
Exemple




```
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	$\cdots$

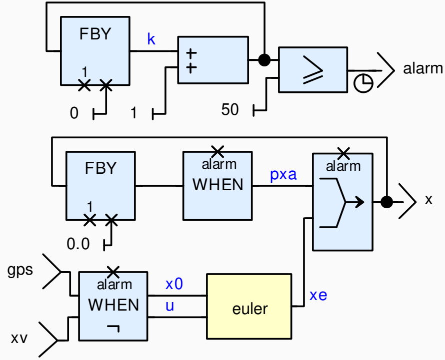
Exemple




```
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	$\cdots$

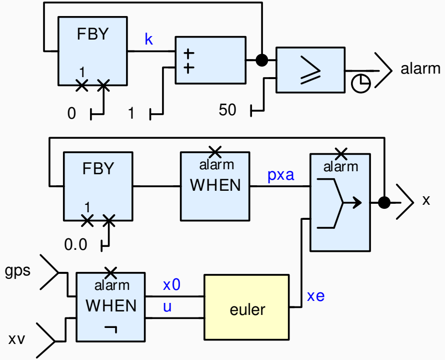
Exemple




```
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	···

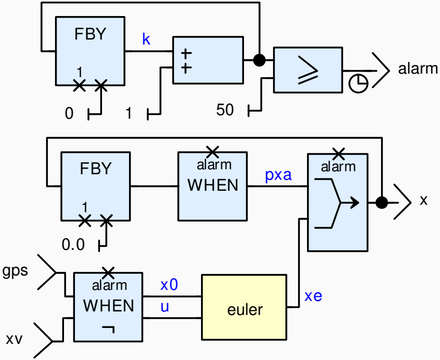
Exemple




```
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	···

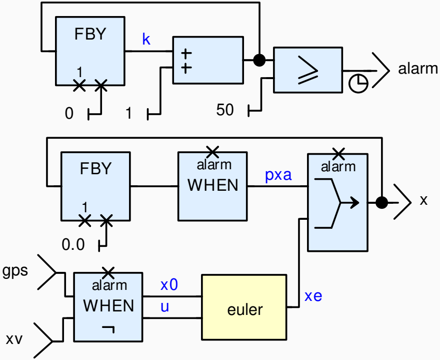
Exemple




```
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	···

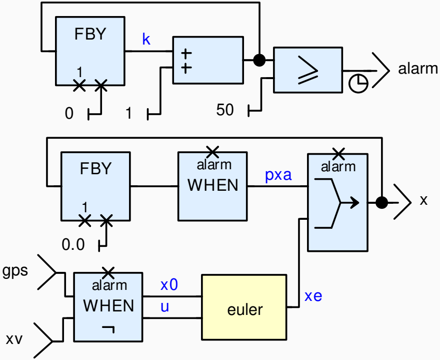
Exemple




```
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	···

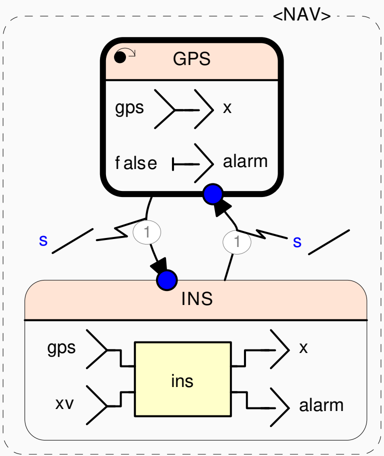
Exemple




```
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
```

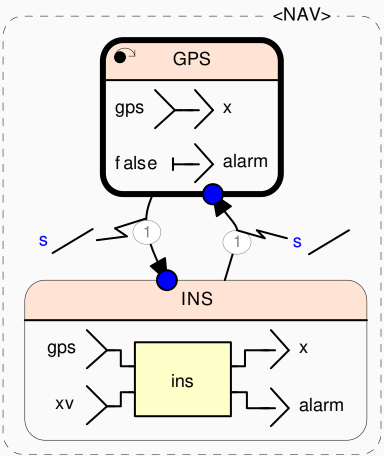
gps	0.00	1.55	3.62	5.46	···	86.52	88.40	90.91	···
xv	15.00	20.00	17.00	12.00	···	18.00	23.00	20.00	···
k	0	1	2	3	···	49	50	51	···
alarm	F	F	F	F	···	F	T	T	···
xe	0.00	1.50	3.50	5.20	···	77.35			···
pxa					···		77.35	77.35	···
x	0.00	1.50	3.50	5.20	···	77.35	77.35	77.35	···

Exemple




```
node nav(gps: double, xv: double, s: bool)
  returns (x: double, alarm: bool)
  var r: bool, 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

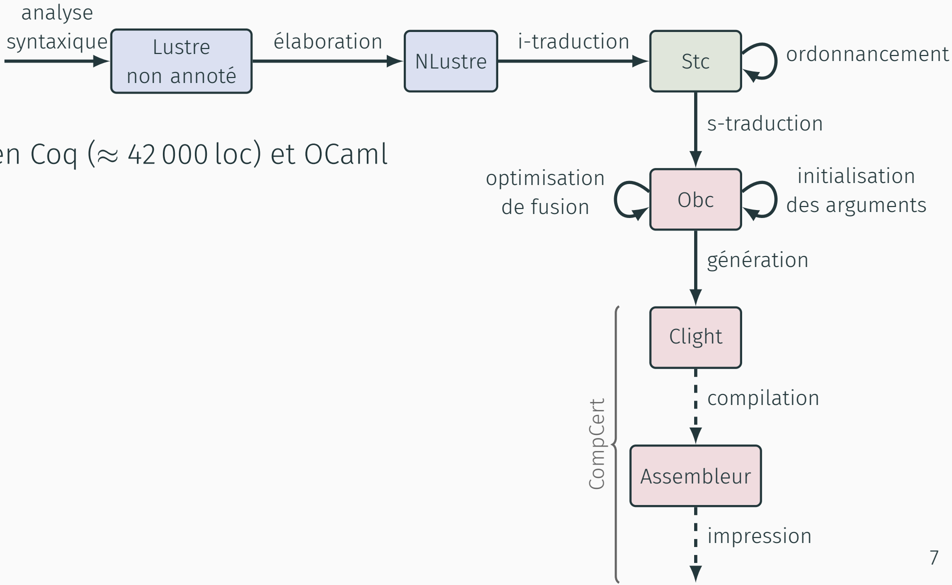



```
node nav(gps: double, xv: double, s: bool)
  returns (x: double, alarm: bool)
  var r: bool, c: bool;
let
  (x, alarm) = merge c
                 (gps when c, false)
                 ((restart ins every r)
                     ((gps, xv) whenot c));
  c = true fby (merge c (not s when c)
       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

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

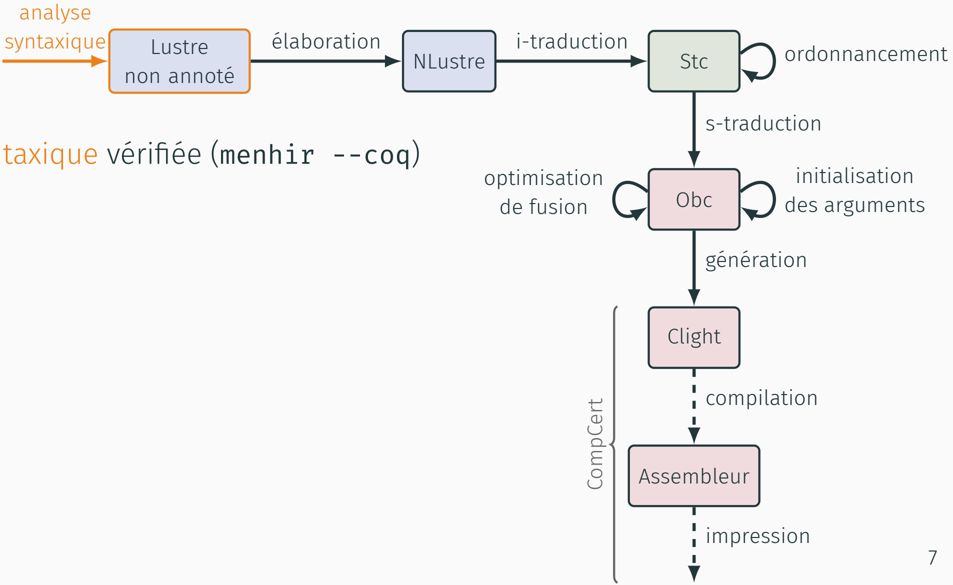
Vélus : un Compilateur Lustre Vérifié

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



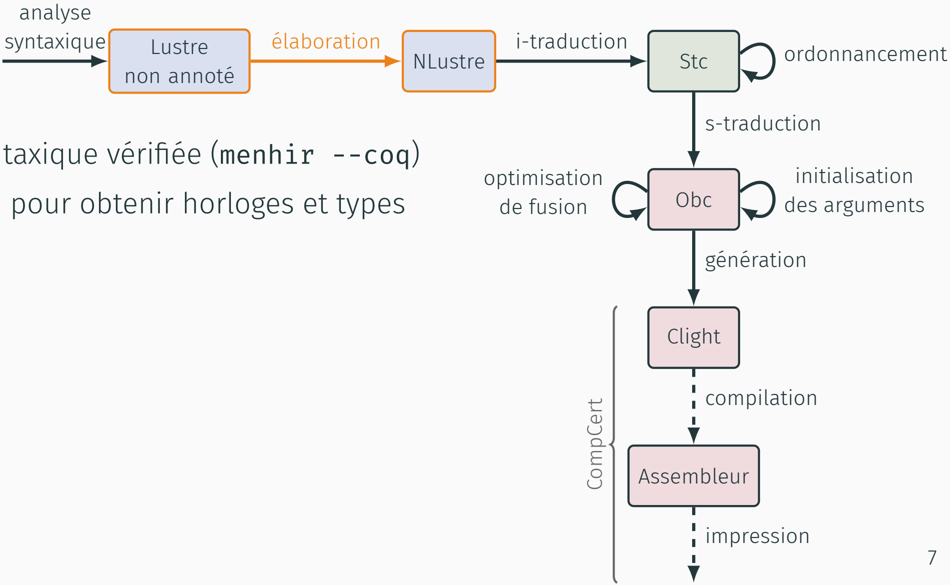
Implémenté en Coq (≈ 42 000 loc) et OCaml

Vélus : un Compilateur Lustre Vérifié



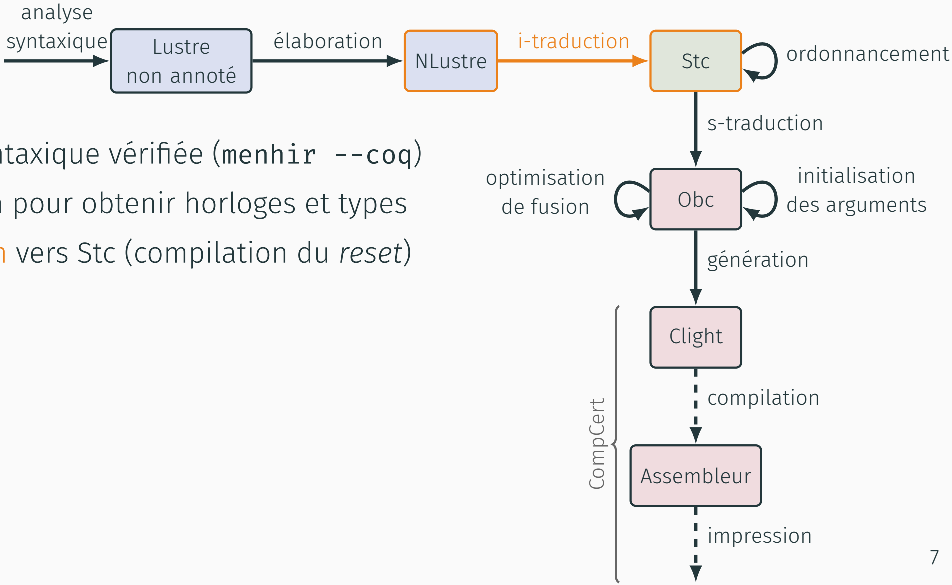
- analyse syntaxique vérifiée (`menhir --coq`)

Vélus : un Compilateur Lustre Vérifié



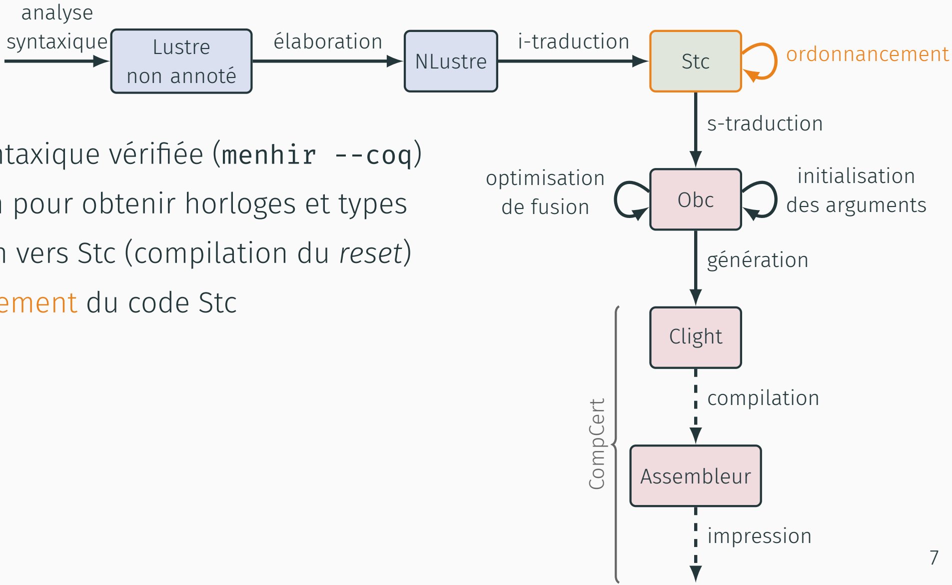
- analyse syntaxique vérifiée (`menhir --coq`)
- élaboration pour obtenir horloges et types

Vélus : un compilateur Lustre vérifié



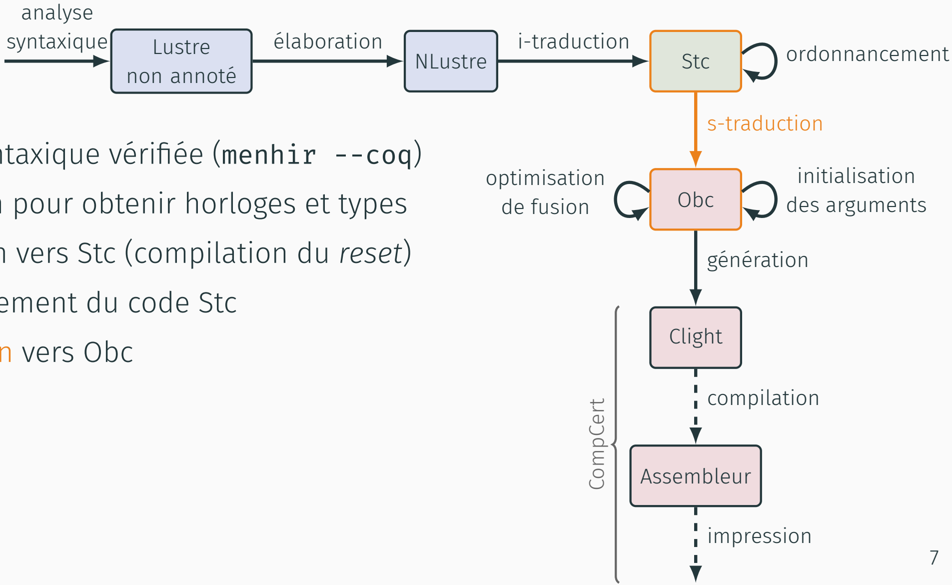
- analyse syntaxique vérifiée (`menhir --coq`)
- élaboration pour obtenir horloges et types
- i-traduction vers Stc (compilation du *reset*)

Vélus : un Compilateur Lustre Vérifié



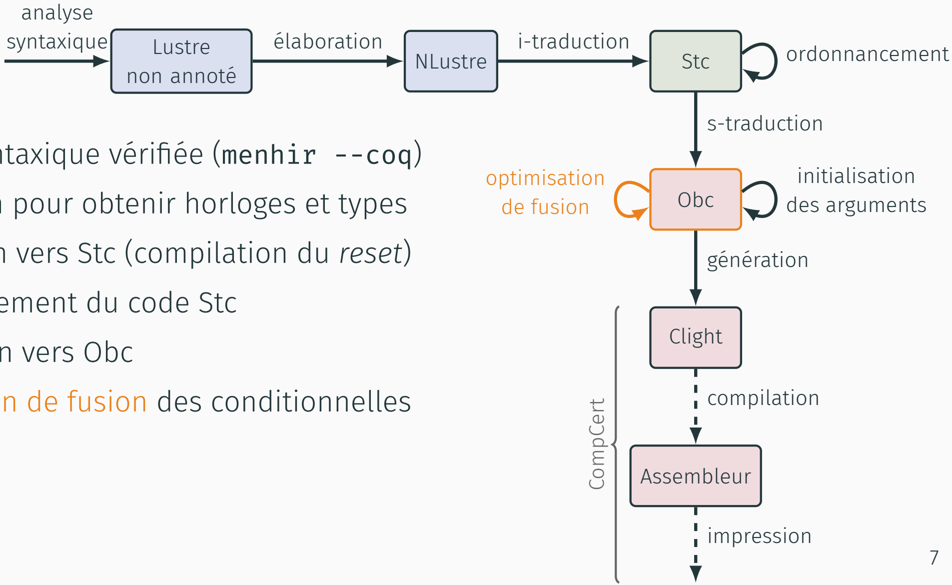
- analyse syntaxique vérifiée (`menhir --coq`)
- élaboration pour obtenir horloges et types
- i-traduction vers Stc (compilation du *reset*)
- ordonnancement du code Stc

Vélus : un compilateur Lustre vérifié



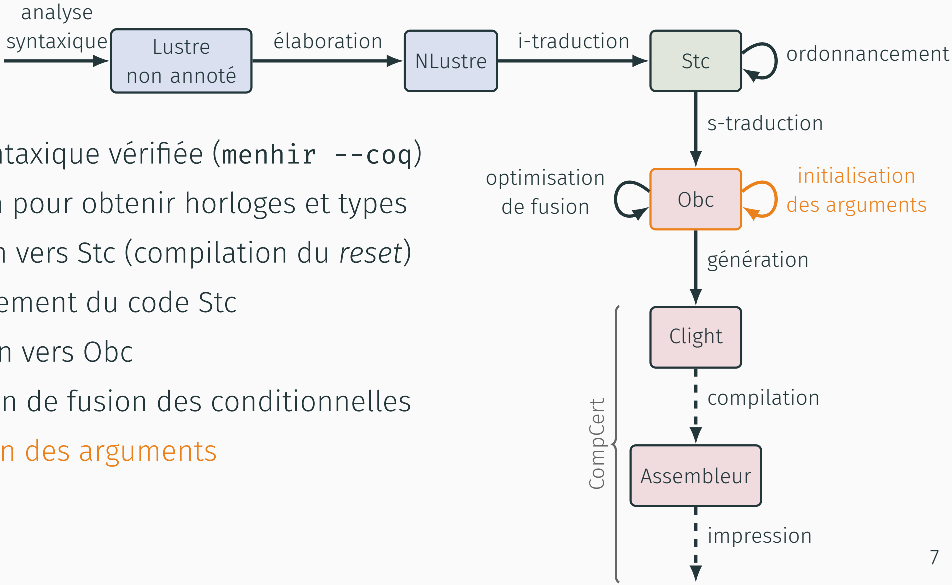
- analyse syntaxique vérifiée (`menhir --coq`)
- élaboration pour obtenir horloges et types
- i-traduction vers Stc (compilation du *reset*)
- ordonnancement du code Stc
- s-traduction vers Obc

Vélus : un Compilateur Lustre Vérifié



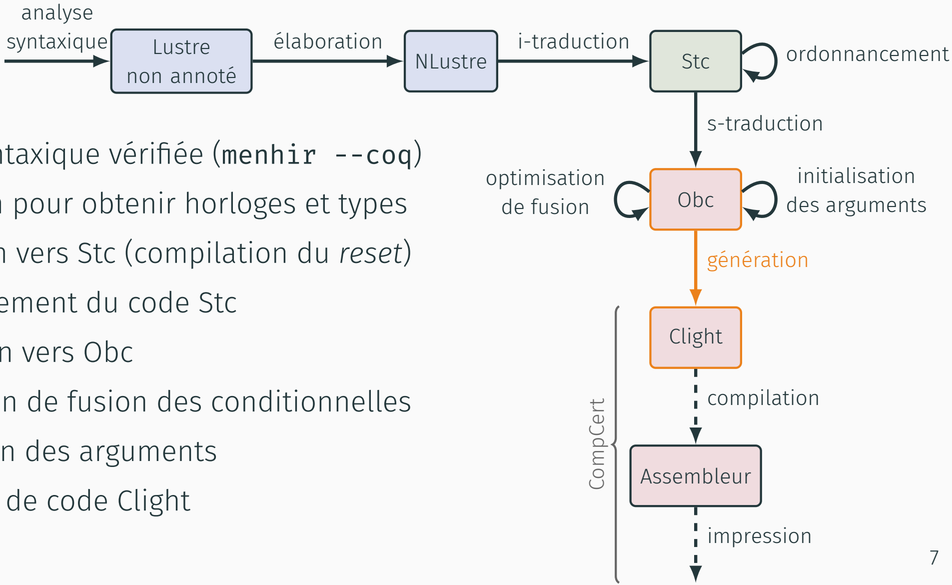
- analyse syntaxique vérifiée (`menhir --coq`)
- élaboration pour obtenir horloges et types
- i-traduction vers Stc (compilation du *reset*)
- ordonnancement du code Stc
- s-traduction vers Obc
- optimisation de fusion des conditionnelles

Vélus : un Compilateur Lustre Vérifié



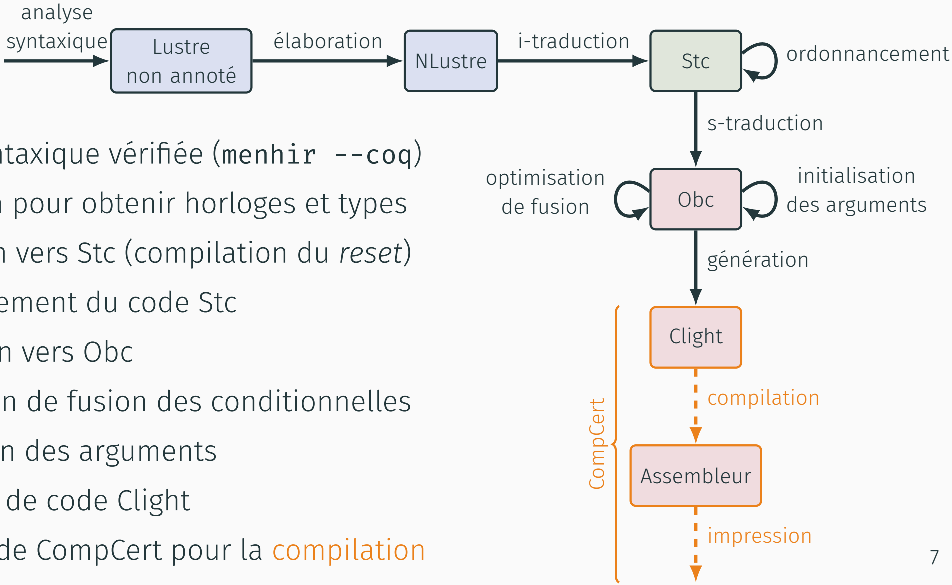
- analyse syntaxique vérifiée (`menhir --coq`)
- élaboration pour obtenir horloges et types
- i-traduction vers Stc (compilation du *reset*)
- ordonnancement du code Stc
- s-traduction vers Obc
- optimisation de fusion des conditionnelles
- initialisation des arguments

Vélus : un Compilateur Lustre Vérifié



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

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: double, xv: double, s: bool)
  returns (x: double, alarm: bool)
  var r: bool, 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: double, xv: double, s: bool)
  returns (x: double, alarm: bool)
  var r: bool, 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 + 0.10000000000000006 * 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;
}

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: double, xv: double, s: bool)
  returns (x: double, alarm: bool)
  var r: bool, 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);
  }
}
```

code traduit

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: double, xv: double, s: bool)
  returns (x: double, alarm: bool)
  var r: bool, 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 + 0.10000000000000006 * 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;
}

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

```
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: double, xv: double, s: bool)
  returns (x: double, alarm: bool)
  var r: bool, 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
```

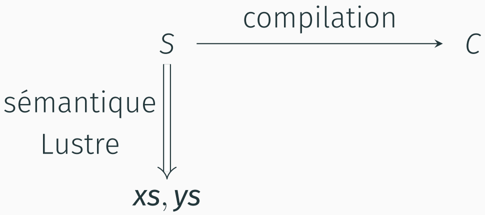
Assembleur

```
        .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 # 0.10000000000000000006
        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
```

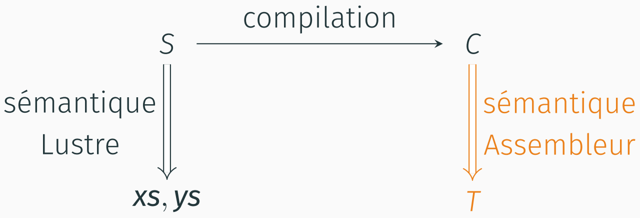
Correction ?

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

Correction ?



Correction ?



compilation
S
C
sémantique
Lustre
xs, ys
sémantique
Assembleur
T

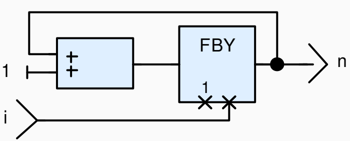
Correction ?

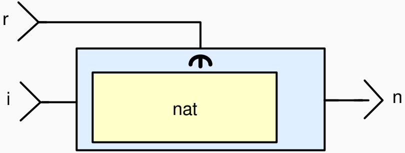

compilation
S
C
sémantique
Lustre
sémantique
Assembleur
xs, ys
T
« équivalence »

Correction ?

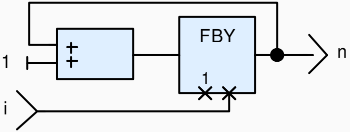
Remarque : on veut en réalité la direction opposée, appelée *raffinement*, c'est-à-dire les comportements observables de C sont aussi des comportements observables de S.

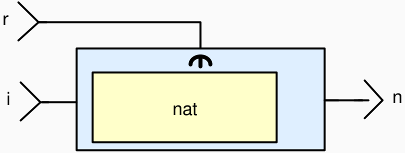
```
node nat(i: int)
  returns (n: int)
let
  n = i fby (n + 1);
tel
```





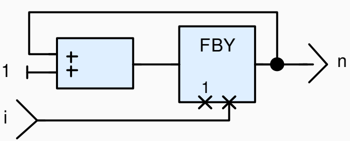
```
node nat(i: int)
  returns (n: int)
let
  n = i fby (n + 1);
tel
```

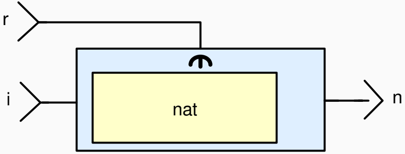




r	F
i	0
$nat(i)$	0
$(\texttt{restart}\ nat\ \texttt{every}\ r)(i)$	0

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

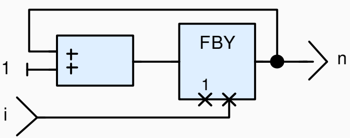


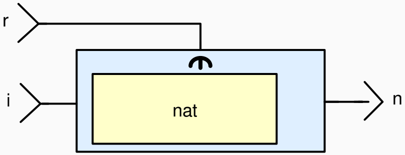


r	F	F
i	0	5
$nat(i)$	0	1
$(\texttt{restart}\ nat\ \texttt{every}\ r)(i)$	0	1

Exemple Plus Simple : Sémantique Intuitive du Reset Modulaire

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

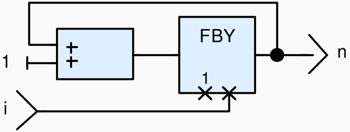


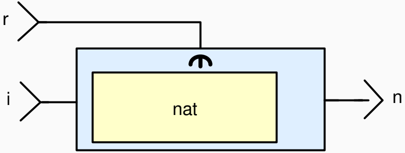


r	F	F	T
i	0	5	10
$nat(i)$	0	1	2
(**restart** nat **every** r)(i)	0	1	10

Exemple Plus Simple : Sémantique Intuitive du Reset Modulaire

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

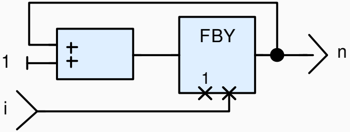


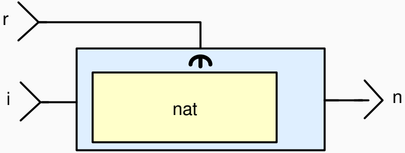


r	F	F	T	F
i	0	5	10	15
$nat(i)$	0	1	2	3
$(\texttt{restart}\ nat\ \texttt{every}\ r)(i)$	0	1	10	11

Exemple Plus Simple : Sémantique Intuitive du Reset Modulaire

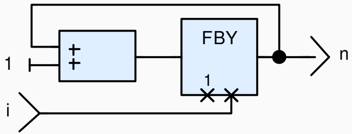
```
node nat(i: int)
  returns (n: int)
let
  n = i fby (n + 1);
tel
```

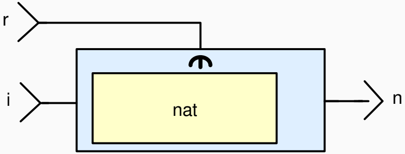




r	F	F	T	F	F
i	0	5	10	15	20
$nat(i)$	0	1	2	3	4
$(\texttt{restart}\ nat\ \texttt{every}\ r)(i)$	0	1	10	11	12

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

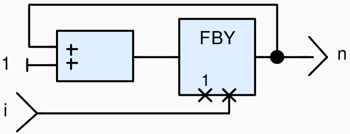


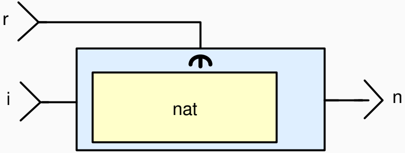


r	F	F	T	F	F	T
i	0	5	10	15	20	25
$nat(i)$	0	1	2	3	4	5
(**restart** *nat* **every** r)(i)	0	1	10	11	12	25

Exemple Plus Simple : Sémantique Intuitive du Reset Modulaire

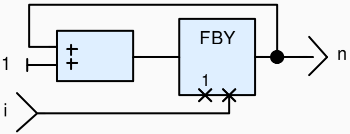
```
node nat(i: int)
  returns (n: int)
let
  n = i fby (n + 1);
tel
```

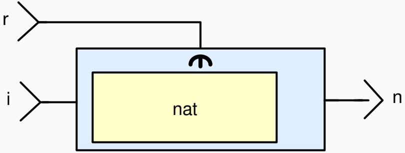




r	F	F	T	F	F	T	F
i	0	5	10	15	20	25	30
$nat(i)$	0	1	2	3	4	5	6
(**restart** *nat* **every** r)(i)	0	1	10	11	12	25	26

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

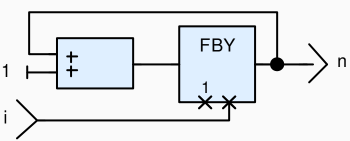


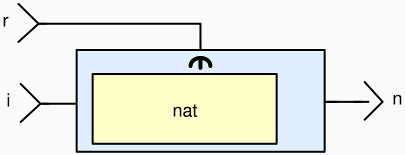


r	F	F	T	F	F	T	F	$\cdots$
i	0	5	10	15	20	25	30	$\cdots$
$nat(i)$	0	1	2	3	4	5	6	$\cdots$
(**restart** *nat* **every** *r*)(*i*)	0	1	10	11	12	25	26	$\cdots$

Exemple Plus Simple : Sémantique Intuitive du Reset Modulaire

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



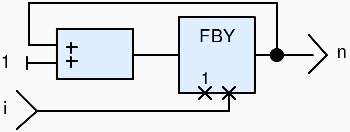


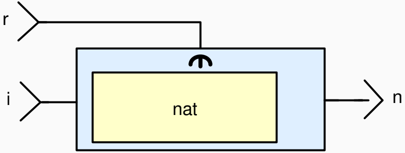
r	F	F	T	F	F	T	F	···
i	0	5	10	15	20	25	30	···
$nat(i)$	0	1	2	3	4	5	6	···
(**restart** *nat* **every** *r*)(*i*)	0	1	10	11	12	25	26	···

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
```



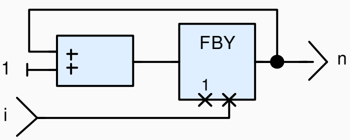


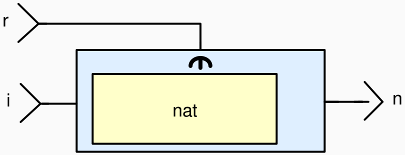
r	F	F	T	F	F	T	F	...
i	0	5	10	15	20	25	30	...
$nat(i)$	0	1	2	3	4	5	6	...
(**restart** *nat* **every** *r*)(*i*)	0	1	10	11	12	25	26	...

Peut être implémenté dans un langage récursif d'ordre supérieur

Exemple Plus Simple : Sémantique Intuitive du Reset Modulaire

```
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	...
$(\textbf{restart}\ nat\ \textbf{every}\ r)(i)$	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			$\cdots$
(`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	$\cdots$
(`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	$\cdots$
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	$\cdots$
$\vdots$								
$(\texttt{restart}\ nat\ \texttt{every}\ r)(i)$	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

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

Instanciation de nœud

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

Instanciation de nœud

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

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

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

Instanciation de nœud

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

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

STC : SYNCHRONOUS TRANSITION CODE

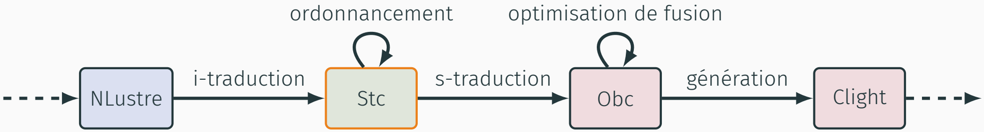
Proposer un nouveau langage intermédiaire

- 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



Système de transitions

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

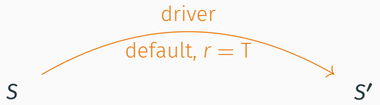
Sémantique Intuitive de Stc

Système de transitions

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

```
system driver {
  sub x: ins;

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

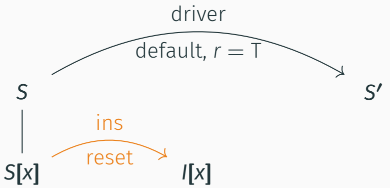


Système de transitions

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

```
system driver {
  sub x: ins;

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

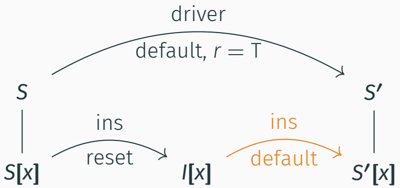


Système de transitions

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

```
system driver {
  sub x: ins;

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

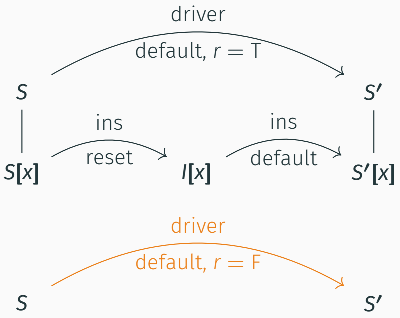


Système de transitions

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

```
system driver {
  sub x: ins;

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

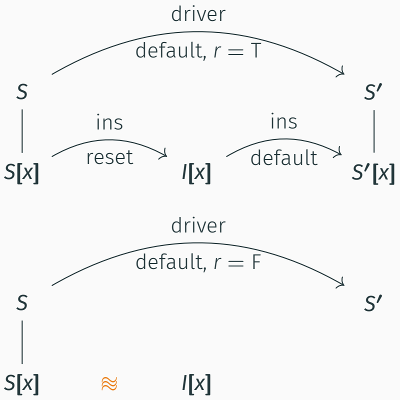


Système de transitions

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

```
system driver {
  sub x: ins;

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

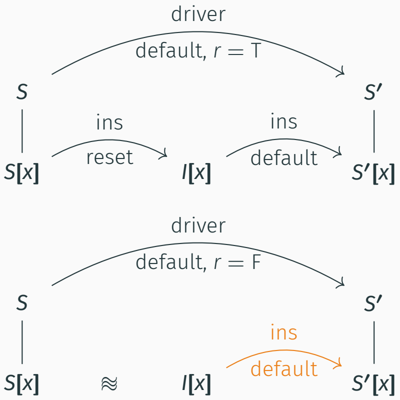


Système de transitions

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

```
system driver {
  sub x: ins;

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



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, ...)

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

E	identifiants vers adresses
L	identifiants vers valeurs
M	adresses vers octets

Conséquences du modèle mémoire de CompCert

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

Manipulation de structures et de pointeurs

Correction : Correspondance Entre Obc et Clight

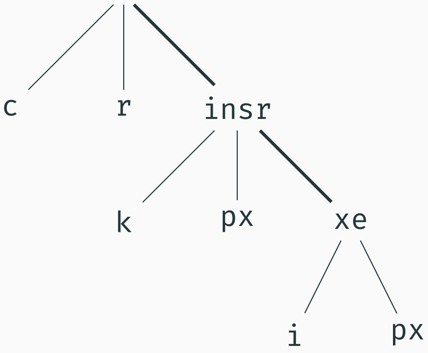
Conséquences du modèle mémoire de CompCert

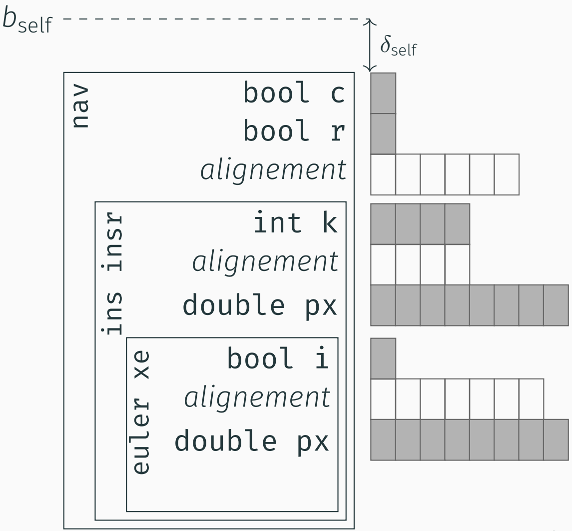
- *aliasing*
- alignement
- permissions
- tailles de types

Manipulation de structures et de pointeurs

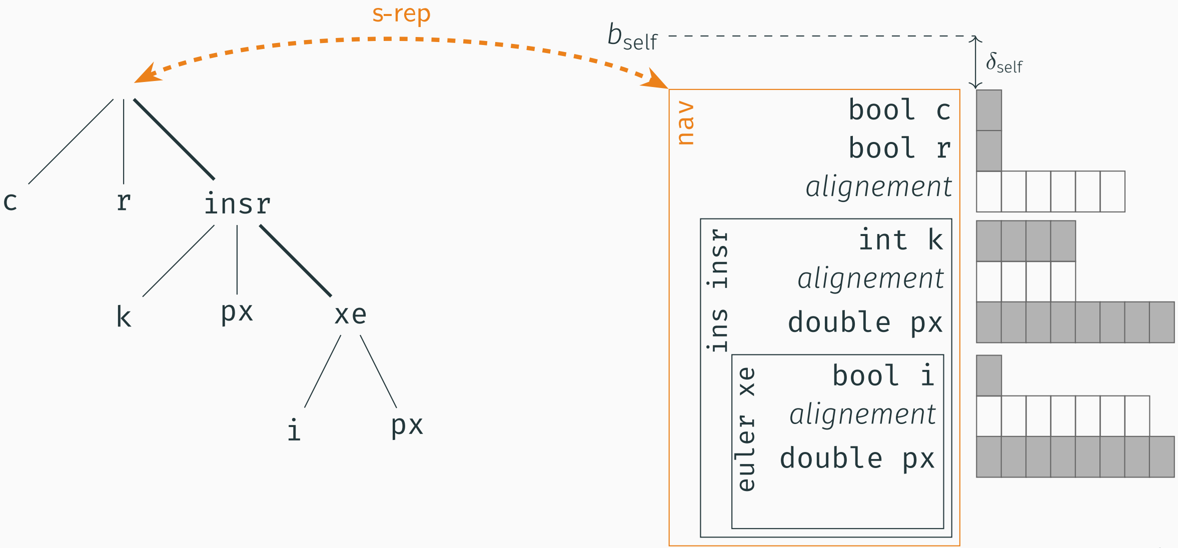
Solution : utiliser des assertions de Logique de Séparation

$$M \models P * Q$$

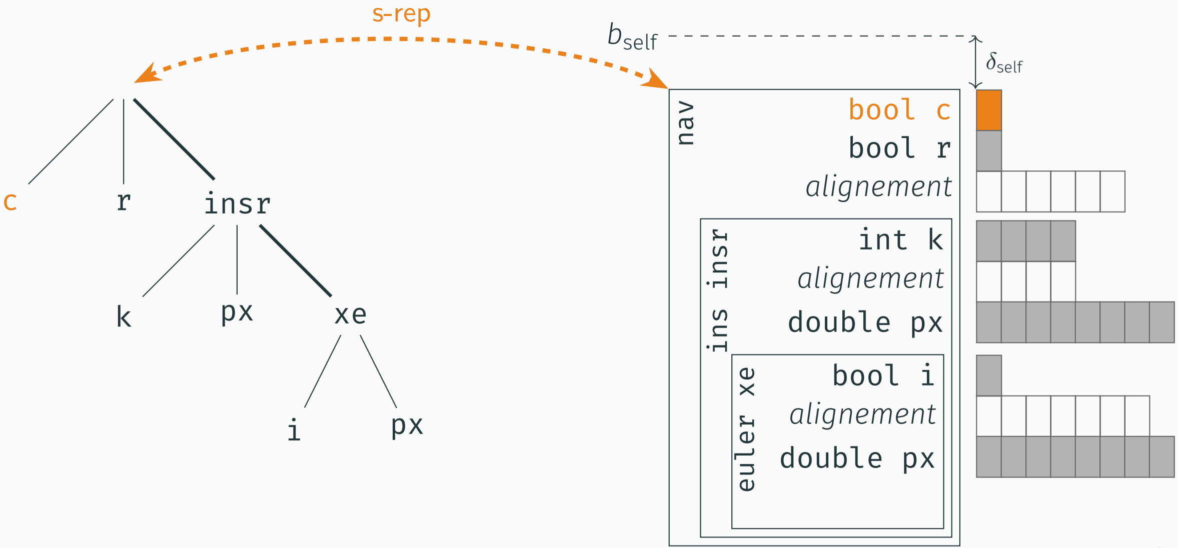

c
r
insr
k
px
xe
i
px


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

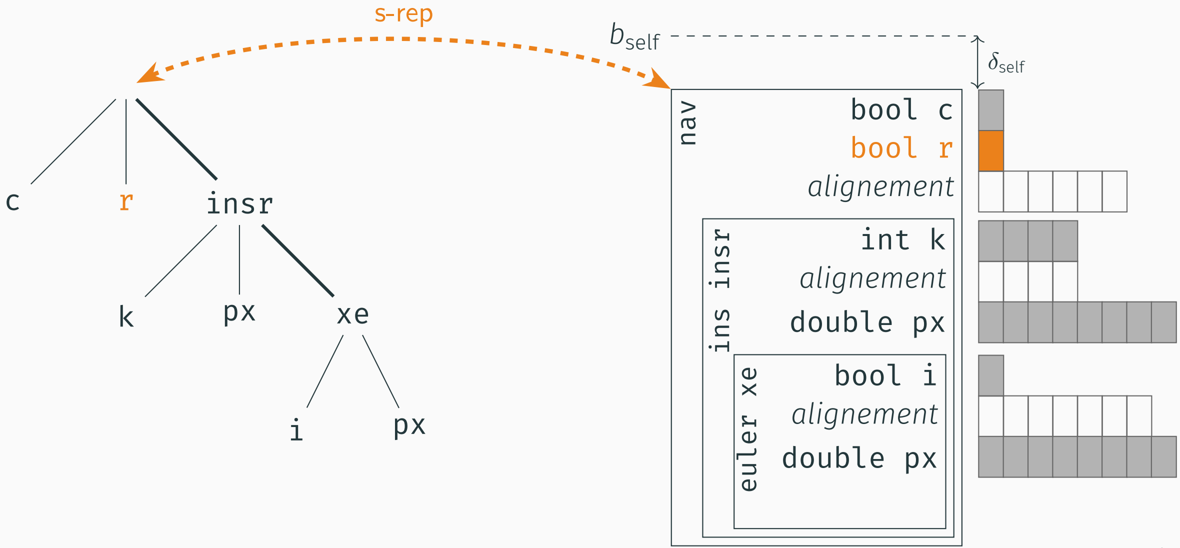
Prédicat de Correspondance d'État



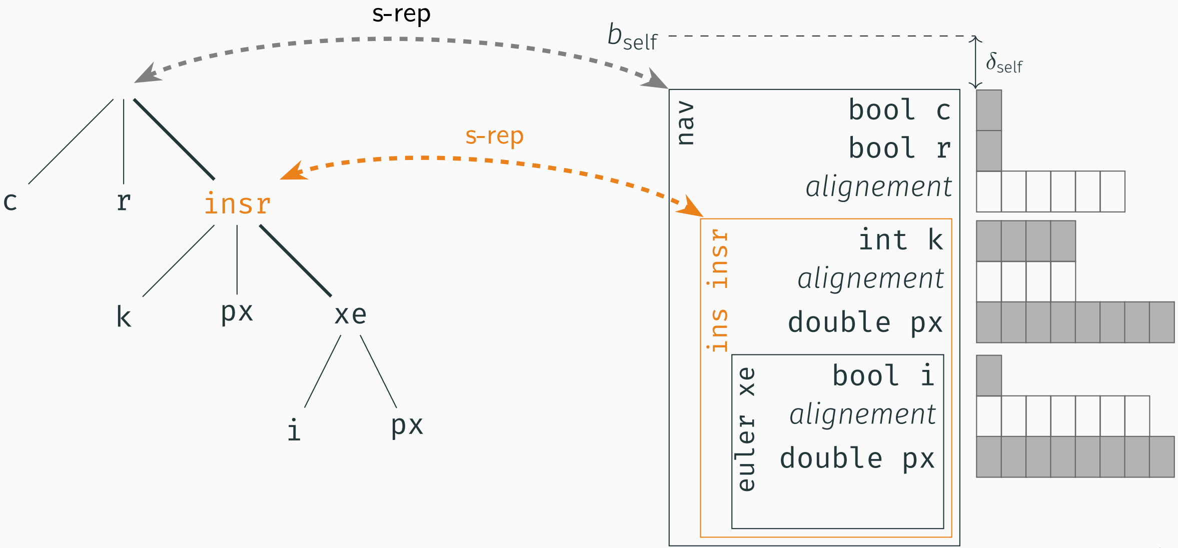
Prédicat de Correspondance d'État

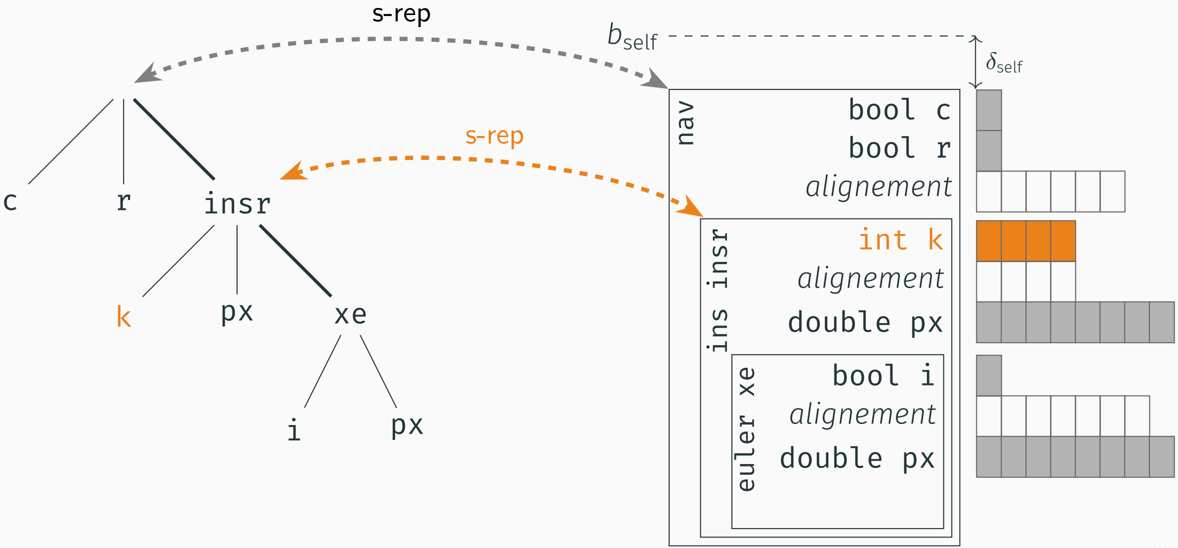


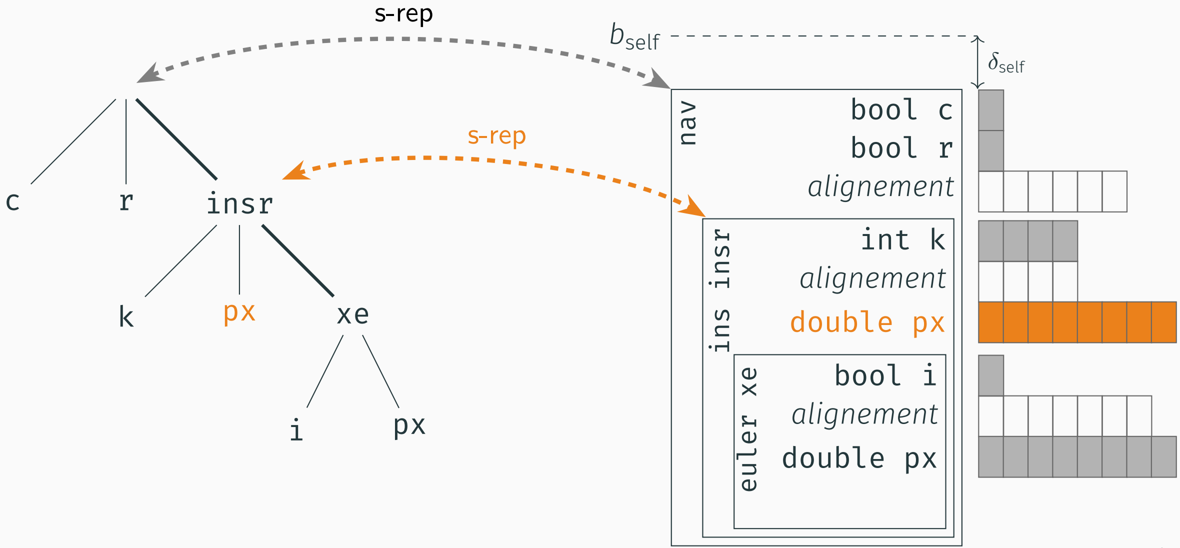
Prédicat de Correspondance d'État



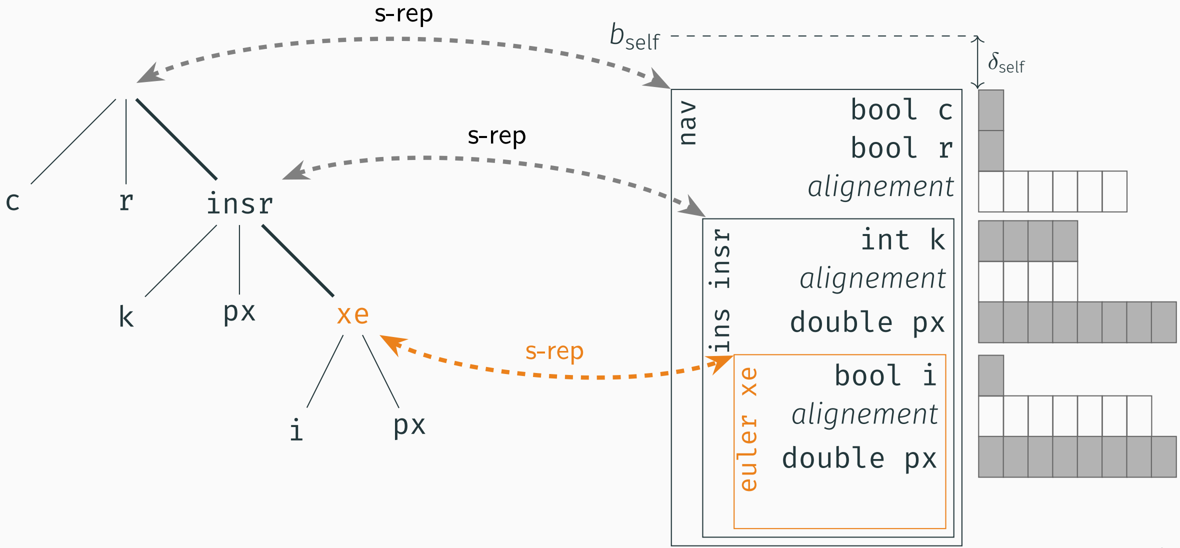
Prédicat de Correspondance d'État



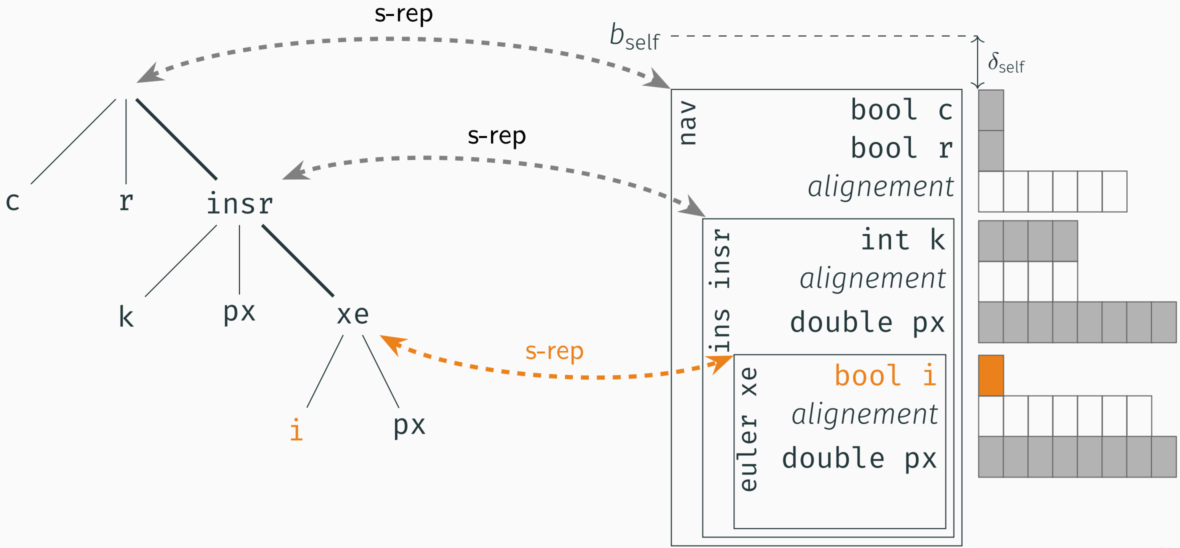

s-rep
b_{self}
δ_{self}
s-rep
c
r
insr
k
px
xe
i
px
nav
bool c
bool r
alignement
ins insr
int k
alignement
double px
euler xe
bool i
alignement
double px

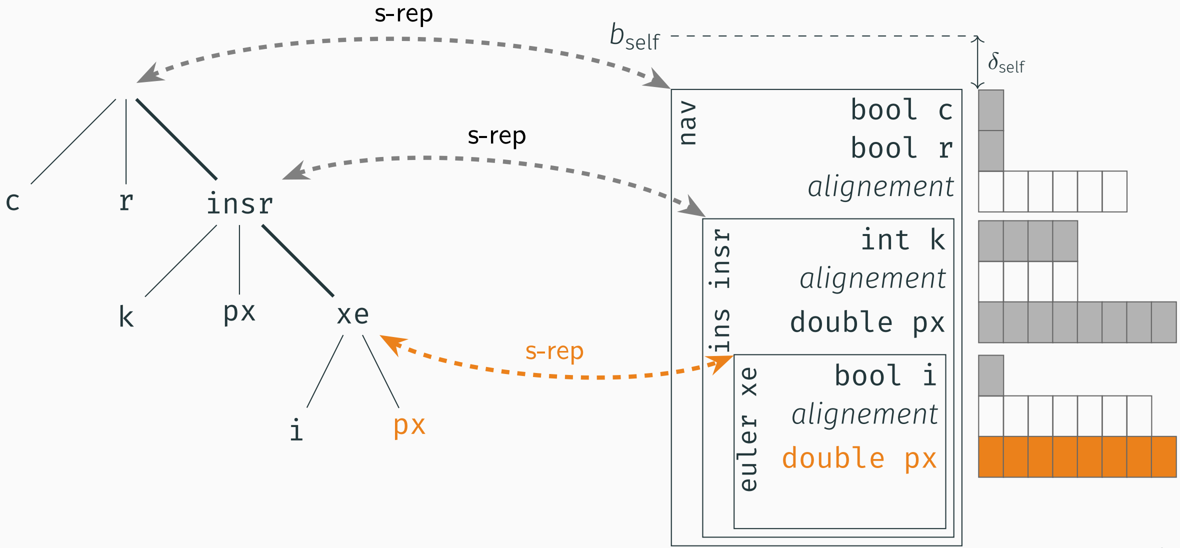

s-rep
b_{self}
δ_{self}
s-rep
c
r
insr
k
px
xe
i
px
nav
bool c
bool r
alignement
ins insr
int k
alignement
double px
euler xe
bool i
alignement
double px

Prédicat de Correspondance d'État



Prédicat de Correspondance d'État




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

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* $\mathsf{compile}\ D\ f = \mathsf{OK}\ (G, P)$ *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 **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*

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

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)

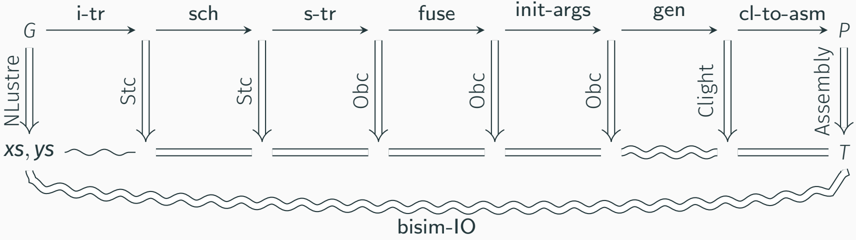
Soient une liste de déclarations D, un nom f, deux listes de flots de valeurs $\mathbf{xs}$ *et* $\mathbf{ys}$*, un programme NLustre G et un programme Assembleur P tels que* $\text{compile}\, D\ f = \text{OK}\,(G, P)$ *et* $G \vdash f\,(\mathbf{xs}) \Downarrow \mathbf{ys}$*, alors, il existe une trace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

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

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*

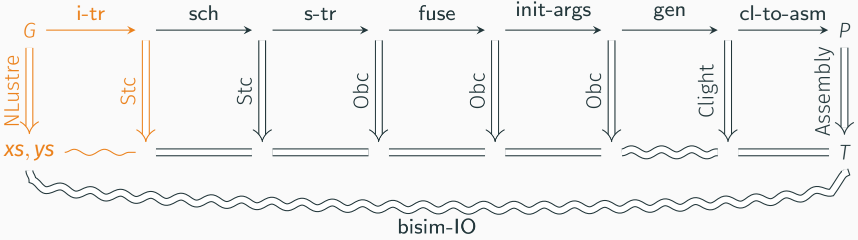
$$P \Downarrow_{ASM} \mathrm{Reacts}(T) \quad et \quad \text{bisim-IO}^{G}\ f\ \mathbf{xs}\ \mathbf{ys}\ T$$



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* $\text{compile}\,D\,f = \text{OK}\,(G, P)$ *et* $G \vdash f\,(\mathbf{xs}) \Downarrow \mathbf{ys}$*, alors, il existe une trace infinie d'événements T telle que*

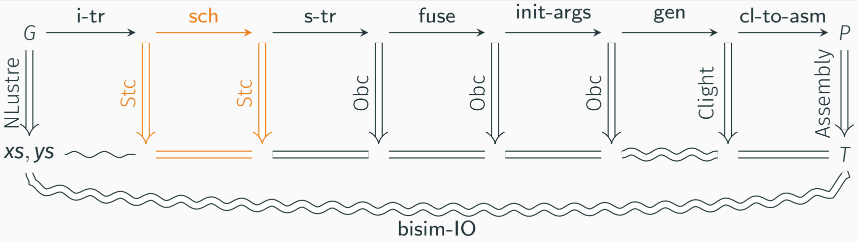
$$P \Downarrow_{ASM} \text{Reacts}(T) \quad \textit{et} \quad \text{bisim-IO}^{G}\, f\, \mathbf{xs}\, \mathbf{ys}\, T$$



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*

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

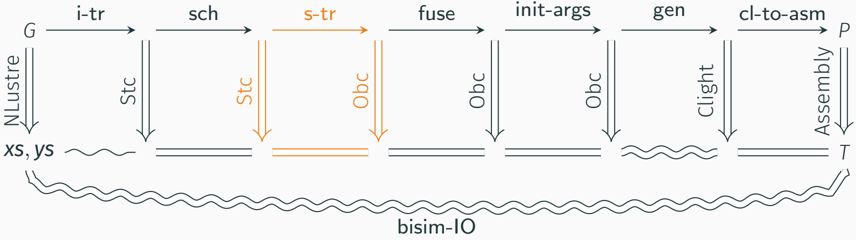


Théorème Final

Théorème (correction de Vélus)

Soient une liste de déclarations D, un nom f, deux listes de flots de valeurs **xs** *et* **ys***, un programme NLustre G et un programme Assembleur P tels que* 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*

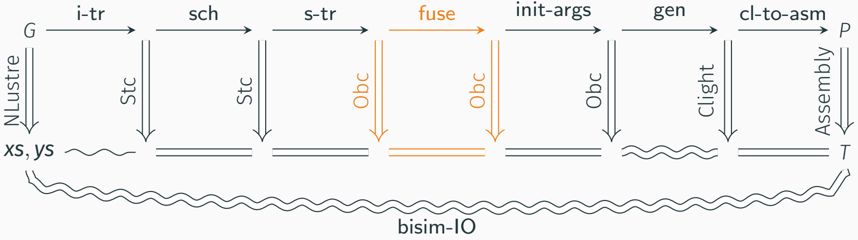
$$P \Downarrow_{ASM} \mathsf{Reacts}(T) \quad \textit{et} \quad \mathsf{bisim\text{-}IO}^{G}\, f\, \mathbf{xs}\, \mathbf{ys}\, T$$



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*

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

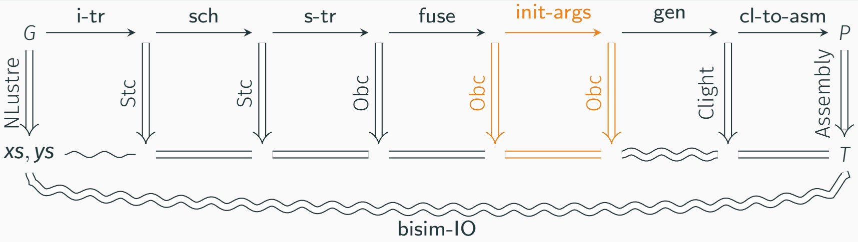


Théorème Final

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*

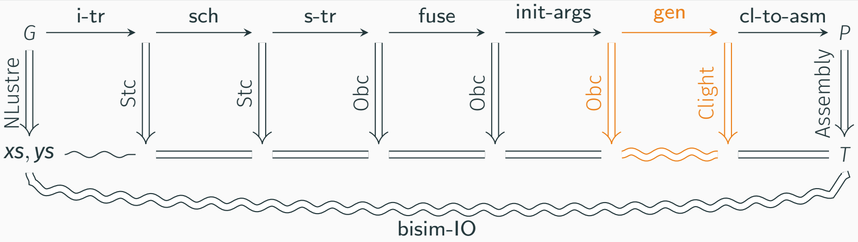
$$P \Downarrow_{ASM} \mathsf{Reacts}(T) \quad \textit{et} \quad \mathsf{bisim\text{-}IO}^G\, f\ \mathbf{xs}\ \mathbf{ys}\ T$$



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* 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*

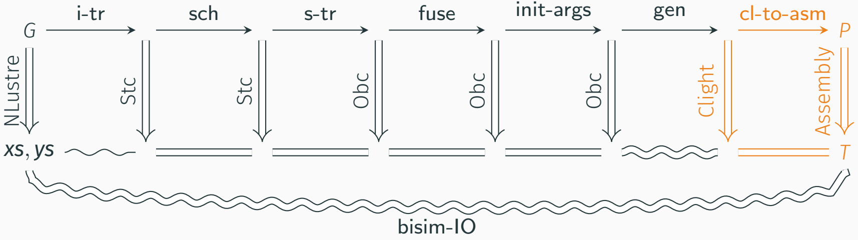
$$P \Downarrow_{ASM} \mathrm{Reacts}(T) \quad \textit{et} \quad \text{bisim-IO}^G\, f\, \mathbf{xs}\, \mathbf{ys}\, T$$



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*

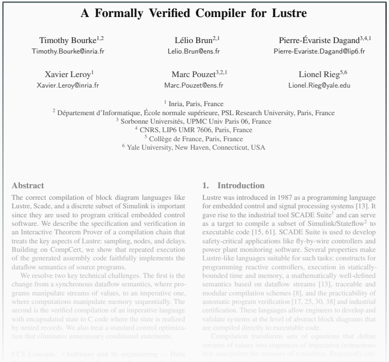
$$P \Downarrow_{ASM} \mathrm{Reacts}(T) \quad \textit{et} \quad \text{bisim-IO}^G\, f\, \mathbf{xs}\, \mathbf{ys}\, T$$



CONTRIBUTIONS

PLDI'17	SCOPES'18	POPL'20

PLDI'17



A Formally Verified Compiler for Lustre

Timothy Bourke[1,2] Timothy.Bourke@inria.fr — Lélio Brun[2,1] Lelio.Brun@ens.fr — Pierre-Évariste Dagand[3,4,1] Pierre-Evariste.Dagand@lip6.fr
Xavier Leroy[1] Xavier.Leroy@inria.fr — Marc Pouzet[3,2,1] Marc.Pouzet@ens.fr — Lionel Rieg[5,6] Lionel.Rieg@yale.edu

[1] Inria, Paris, France
[2] Département d'Informatique, École normale supérieure, PSL Research University, Paris, France
[3] Sorbonne Universités, UPMC Univ Paris 06, France
[4] CNRS, LIP6 UMR 7606, Paris, France
[5] Collège de France, Paris, France
[6] Yale University, New Haven, Connecticut, USA

Abstract

The correct compilation of block diagram languages like Lustre, Scade, and a discrete subset of Simulink is important since they are used to program critical embedded control software. We describe the specification and verification in an Interactive Theorem Prover of a compilation chain that treats the key aspects of Lustre: sampling, nodes, and delays. Building on CompCert, we show that repeated execution of the generated assembly code faithfully implements the dataflow semantics of source programs.

We resolve two key technical challenges. The first is the change from a synchronous dataflow semantics, where programs manipulate streams of values, to an imperative one, where computations manipulate memory sequentially. The second is the verified compilation of an imperative language with encapsulated state to C code where the state is realized by nested records. We also treat a standard control optimization that eliminates unnecessary conditional statements.

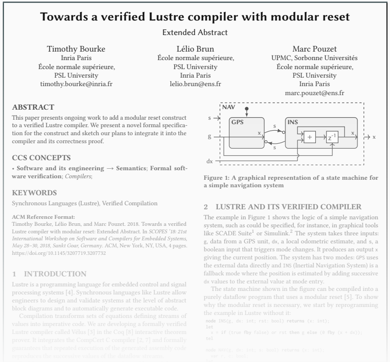
1. Introduction

Lustre was introduced in 1987 as a programming language for embedded control and signal processing systems [13]. It gave rise to the industrial tool SCADE Suite and can serve as a target to compile a subset of Simulink/Stateflow to executable code [15, 61]. SCADE Suite is used to develop safety-critical applications like fly-by-wire controllers and power plant monitoring software. Several properties make Lustre-like languages suitable for such tasks: constructs for programming reactive controllers, execution in statically-bounded time and memory, a mathematically well-defined semantics based on dataflow streams [13], traceable and modular compilation schemes [8], and the practicability of automatic program verification [17, 25, 30, 38] and industrial certification. These languages allow engineers to develop and validate systems at the level of abstract block diagrams that are compiled directly to executable code.

1[re] version de Vélus

passe Obc vers Clight

SCOPES'18



Towards a verified Lustre compiler with modular reset

Extended Abstract

Timothy Bourke, Inria Paris, École normale supérieure, PSL University, timothy.bourke@inria.fr — Lélio Brun, École normale supérieure, PSL University, Inria Paris, lelio.brun@ens.fr — Marc Pouzet, UPMC, Sorbonne Universités, École normale supérieure, PSL University, Inria Paris, marc.pouzet@ens.fr

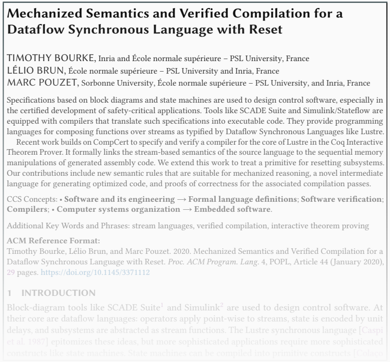
ABSTRACT

This paper presents ongoing work to add a modular reset construct to a verified Lustre compiler. We present a novel formal specification for the construct and sketch our plans to integrate it into the compiler and its correctness proof.

Figure 1: A graphical representation of a state machine for a simple navigation system

Sémantique formelle du *reset*

POPL'20



Mechanized Semantics and Verified Compilation for a Dataflow Synchronous Language with Reset

TIMOTHY BOURKE, Inria and École normale supérieure – PSL University, France
LÉLIO BRUN, École normale supérieure – PSL University and Inria, France
MARC POUZET, Sorbonne University, École normale supérieure – PSL University, and Inria, France

Specifications based on block diagrams and state machines are used to design control software, especially in the certified development of safety-critical applications. Tools like SCADE Suite and Simulink/Stateflow are equipped with compilers that translate such specifications into executable code. They provide programming languages for composing functions over streams as typified by Dataflow Synchronous Languages like Lustre.

Recent work builds on CompCert to specify and verify a compiler for the core of Lustre in the Coq Interactive Theorem Prover. It formally links the stream-based semantics of the source language to the sequential memory manipulations of generated assembly code. We extend this work to treat a primitive for resetting subsystems. Our contributions include new semantic rules that are suitable for mechanized reasoning, a novel intermediate language for generating optimized code, and proofs of correctness for the associated compilation passes.

2[e] version de Vélus

Stc et compilation du *reset*

Conclusion

Résumé

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

VÉLUS

`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

- 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.

Références III

- 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.
- 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.

Références IV

- 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.