Des langages de programmation formels pour le logiciel temps réel

Marc Pouzet
DIENS
Equipe INRIA PARKAS [1]

PSL/ENS
Marc.Pouzet@ens.fr

Exposé de rentrée, ENS Paris-Saclay
29 septembre 2023

[1] https://parkas.di.ens.fr

Un étudiant, un enseignant chercheur, part en mission...


$x = 0$ fby $x + 1$


$x = 0$ fby $x + 1$
Justificatifs!

Justificatifs!

Comment spécifier, programmer, vérifier le logiciel de cette petite imprimante?

Qu'y a-t'il ici de si différent?

Des composants physiques, e.g., des capteurs, des actionneurs, un moteur pas-à-pas (rouleau à papier), etc.

Contrôler en permanence (la température), commander suffisamment vite, de manière régulière (le moteur), etc.

Plus largement...

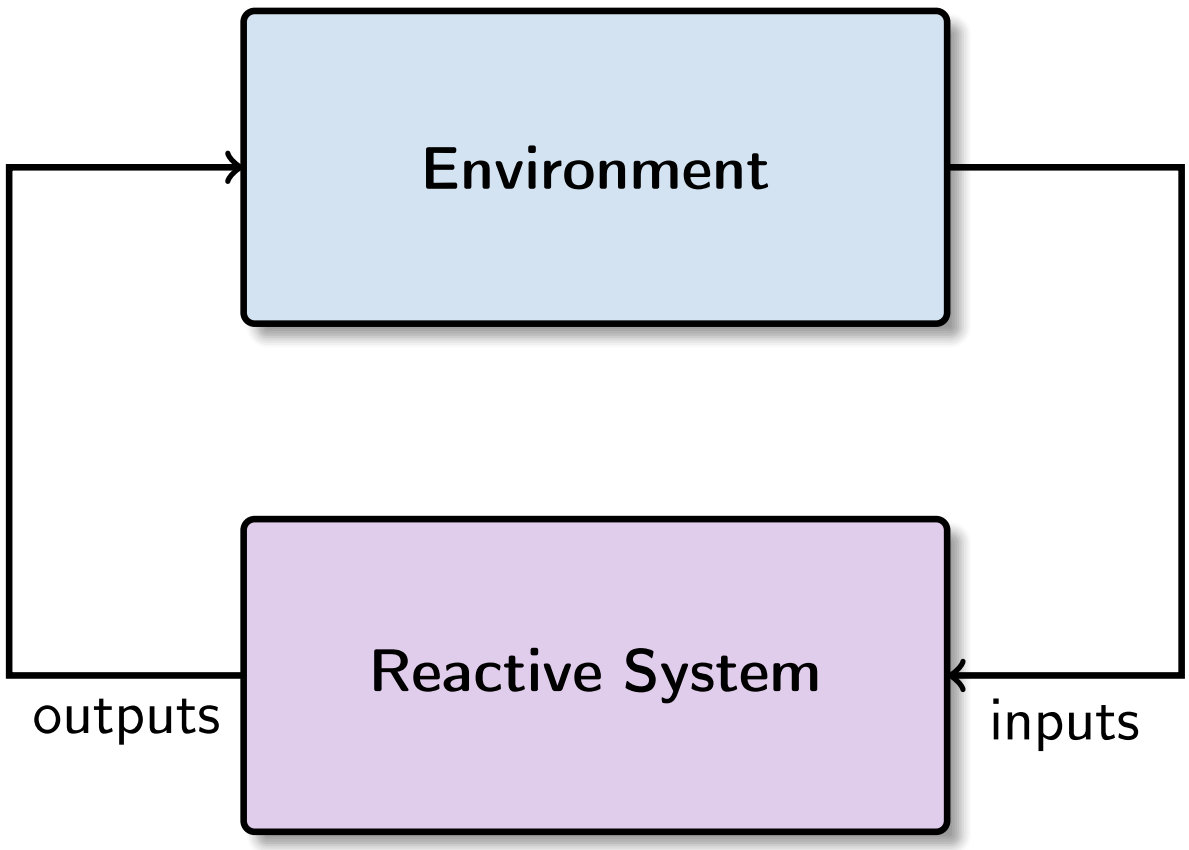
Les commandes de vol d'un avion, les freins, le moteur, etc.

Le contrôle de bord d'un train, sa localisation, la commande des aiguillages, etc.

La gestion électrique d'une voiture, etc.

Ce sont tous des systèmes réactifs temps réel et embarqués.

De quoi s'agit-il?



Ce qui est spécifique

Du parallélisme:

L'environnement et le contrôleur fonctionnent en parallèle et en même temps;

E.g., contrôler (en même temps) roulis et tanguage,
commander (en même temps) les freins d'un train.

Du déterminisme:

Si le système de contrôle/commande est soumis aux mêmes entrées, ses sorties sont elles les mêmes?

Veut-on d'un système non déterministe? Comment s'assurer alors qu'il fonctionne correctement?

Que signifie qu'un calcul doit être effectué "toutes les 10 milisecondes"? Comment s'en assurer?

Que se passe-t'il s'il prend plus de temps que prévu?

Écrire du code Assembleur/C/C++/JavaScript/Python/OCaml/Coq à la main?

Et le comparer a quoi?

Et le comparer a quoi?

Quelle est la spécification?

Et si on a une spécification formelle,

Et si on a une spécification formelle,

dans quel langage de programmation la retranscrire?

Et si on a une spécification formelle,

dans quel langage de programmation la retranscrire?

comment s'assûrer que le code produit est sûr et correct? E.g.,

Et si on a une spécification formelle,

dans quel langage de programmation la retranscrire?

comment s'assûrer que le code produit est sûr et correct? E.g.,

il n'y a pas d'erreur à l'exécution;

Et si on a une spécification formelle,

dans quel langage de programmation la retranscrire?

comment s'assûrer que le code produit est sûr et correct? E.g.,

il n'y a pas d'erreur à l'exécution;

la mémoire utilisée est bornée et connue statiquement;

Et si on a une spécification formelle,

dans quel langage de programmation la retranscrire?

comment s'assûrer que le code produit est sûr et correct? E.g.,

il n'y a pas d'erreur à l'exécution;

la mémoire utilisée est bornée et connue statiquement;

le temps de réponse pire-cas est borné et connu statiquement;

Et si on a une spécification formelle,

dans quel langage de programmation la retranscrire?

comment s'assûrer que le code produit est sûr et correct? E.g.,

il n'y a pas d'erreur à l'exécution;

la mémoire utilisée est bornée et connue statiquement;

le temps de réponse pire-cas est borné et connu statiquement;

le code exécuté implémente fidèlement la spécification.

Une idée radicale

Inventer des langages de programmation dédiés, [2]

[2] Gérard Berry: Real Time Programming: Special Purpose or General Purpose Languages. IFIP Congress 1989

Une idée radicale

Inventer des langages de programmation dédiés, [2]

dont l'expressivité est adaptée, e.g., conciliant parallélisme et déterminisme;

[2] Gérard Berry: Real Time Programming: Special Purpose or General Purpose Languages. IFIP Congress 1989

Une idée radicale

Inventer des langages de programmation dédiés, [2]

dont l'expressivité est adaptée, e.g., conciliant parallélisme et déterminisme;

permettant de parler du temps et de calculer avec, e.g.:

[2] Gérard Berry: Real Time Programming: Special Purpose or General Purpose Languages. IFIP Congress 1989

Une idée radicale

Inventer des langages de programmation dédiés, [2]

dont l'expressivité est adaptée, e.g., conciliant parallélisme et déterminisme;

permettant de parler du temps et de calculer avec, e.g.:

`await 3 second; await 2 second` est-il équivalent à `await 5 second`?

[2] Gérard Berry: Real Time Programming: Special Purpose or General Purpose Languages. IFIP Congress 1989

Une idée radicale

Inventer des langages de programmation dédiés, [2]

dont l'expressivité est adaptée, e.g., conciliant parallélisme et déterminisme;

permettant de parler du temps et de calculer avec, e.g.:

`await 3 second; await 2 second` est-il équivalent à `await 5 second`?

indépendemment de toute implémentation,

[2] Gérard Berry: Real Time Programming: Special Purpose or General Purpose Languages. IFIP Congress 1989

Une idée radicale

Inventer des langages de programmation dédiés, [2]

dont l'expressivité est adaptée, e.g., conciliant parallélisme et déterminisme;

permettant de parler du temps et de calculer avec, e.g.:

`await 3 second; await 2 second` est-il équivalent à `await 5 second`?

indépendemment de toute implémentation,

s'appuyant sur une sémantique statique/dynamique et une compilation formellement définies;

[2] Gérard Berry: Real Time Programming: Special Purpose or General Purpose Languages. IFIP Congress 1989

Une idée radicale

Inventer des langages de programmation dédiés, [2]

dont l'expressivité est adaptée, e.g., conciliant parallélisme et déterminisme;

permettant de parler du temps et de calculer avec, e.g.:

`await 3 second; await 2 second` est-il équivalent à `await 5 second`?

indépendemment de toute implémentation,

s'appuyant sur une sémantique statique/dynamique et une compilation formellement définies;

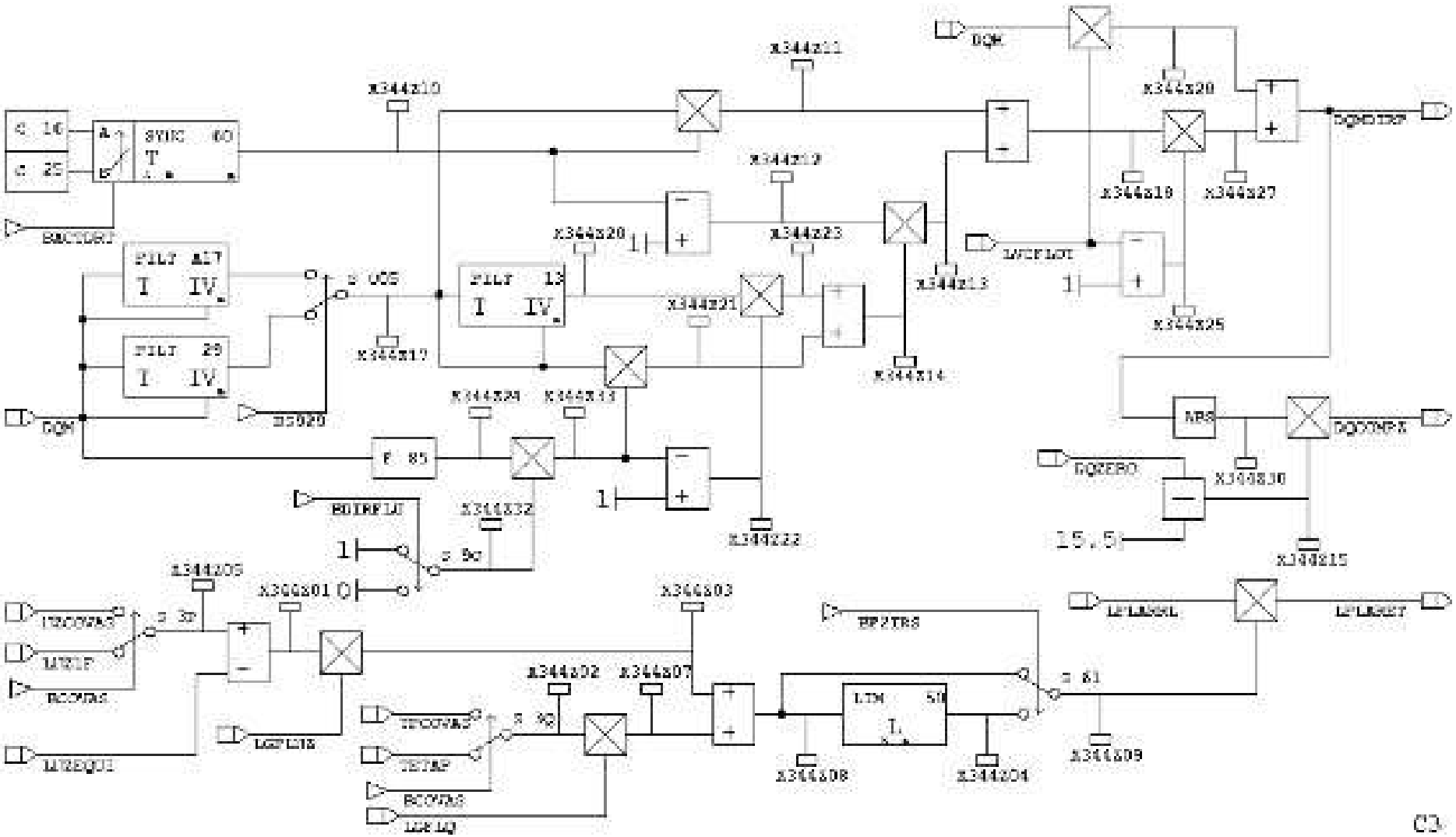
pour que les propriétées vérifiées sur le source le soient sur le code produit.

"what you prove is what you execute" [Berry, 89]

[2] Gérard Berry: Real Time Programming: Special Purpose or General Purpose Languages. IFIP Congress 1989

Un langage permettant d'écrire des spécifications mathématiques exécutables?

SAO (Spécification Assistée par Ordinateur) — Airbus 80's



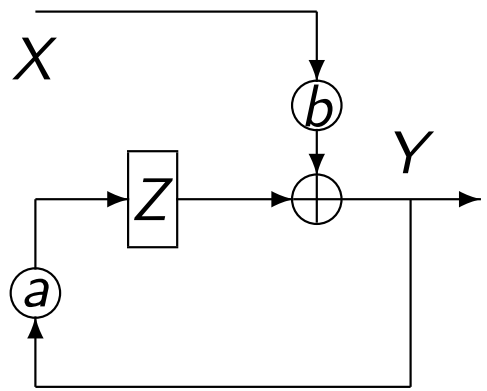
Des dessins précis

Les automaticiens/traiteurs de signaux décrivaient les systèmes de contrôle/commande avec des mathématiques précises avant même l'arrivée de calculateurs.

Systèmes échantillonés, équations de suites, machines à état, etc.

Exemple: un filtre linéaire

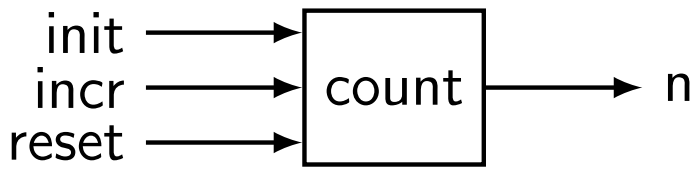
$$Y_0 = bX_0 \ , \ \forall n \ Y_{n+1} = aY_n + bX_{n+1}$$



...mais non exécutables! Écrire du code et se convaincre qu'il est juste.

Comment rendre ces mathématiques exécutables?

A Grenoble... le langage Lustre (Caspi et Halbwachs, 1987) [CHPP87]




```
node COUNT (init, incr: int; reset: bool)
   returns (n: int);
let
   n = init ->
       if reset then init else pre(n) + incr;
tel;
```

Programmer en écrivant des équations de suites

Un système discret: une fonction de suites; les suites sont synchrones.

X	1	2	1	4	5	6	...
Y	2	4	2	1	1	2	...
$X + Y$	3	6	3	5	6	8	...
pre X	*nil*	1	2	1	4	5	...
Y *->*X	2	2	1	4	5	6	...
C	*true*	*false*	*true*	*true*	*false*	*true*	...
X *when* C	1		1	4		6	...
Z			1		2		...
current(Z)	*nil*	*nil*	1	1	2	2	...

L'hypothèse synchrone

- Les opérations importées (e.g., $+$) s'appliquent point-à-point.
- pre est le délai unitaire; -> est l'opération d'initialisation.
- when est l'opérateur d'échantillonnage: il produit une sous-suite; a stream;
- current maintient une valeur (0-holder).
- L'équation $Z = X + Y$ signifie $\forall n. Z_n = X_n + Y_n$.
- Le temps est logical: les entrées X et Y arrivent "en même temps"; la sortie Z est produite "en même temps"
- Restreindre l'expressivité pour garantir une exécution en temps et mémoire bornés.

c'est temps réel?

Raisonner en pire cas: vérifier que le code généré produit la sortie avant l'arrivée de l'entrée suivante.

Démo avec Vélus

`https://velus.inria.fr`

un filtre linéaire,

un intégrateur,

le moteur pas-à-pas (de notre petite imprimante)

L'idée géniale de Lustre

Une interprétation synchrone des réseaux de Kahn et MacQueen [Kah74, KM77] et de Lucid [AW85].

Ecrire directement les modèles échantillonnés de l'automatique.

Les analyser/transformer/simuler/tester/vérifier.

Les traduire automatiquement vers du code exécutable.

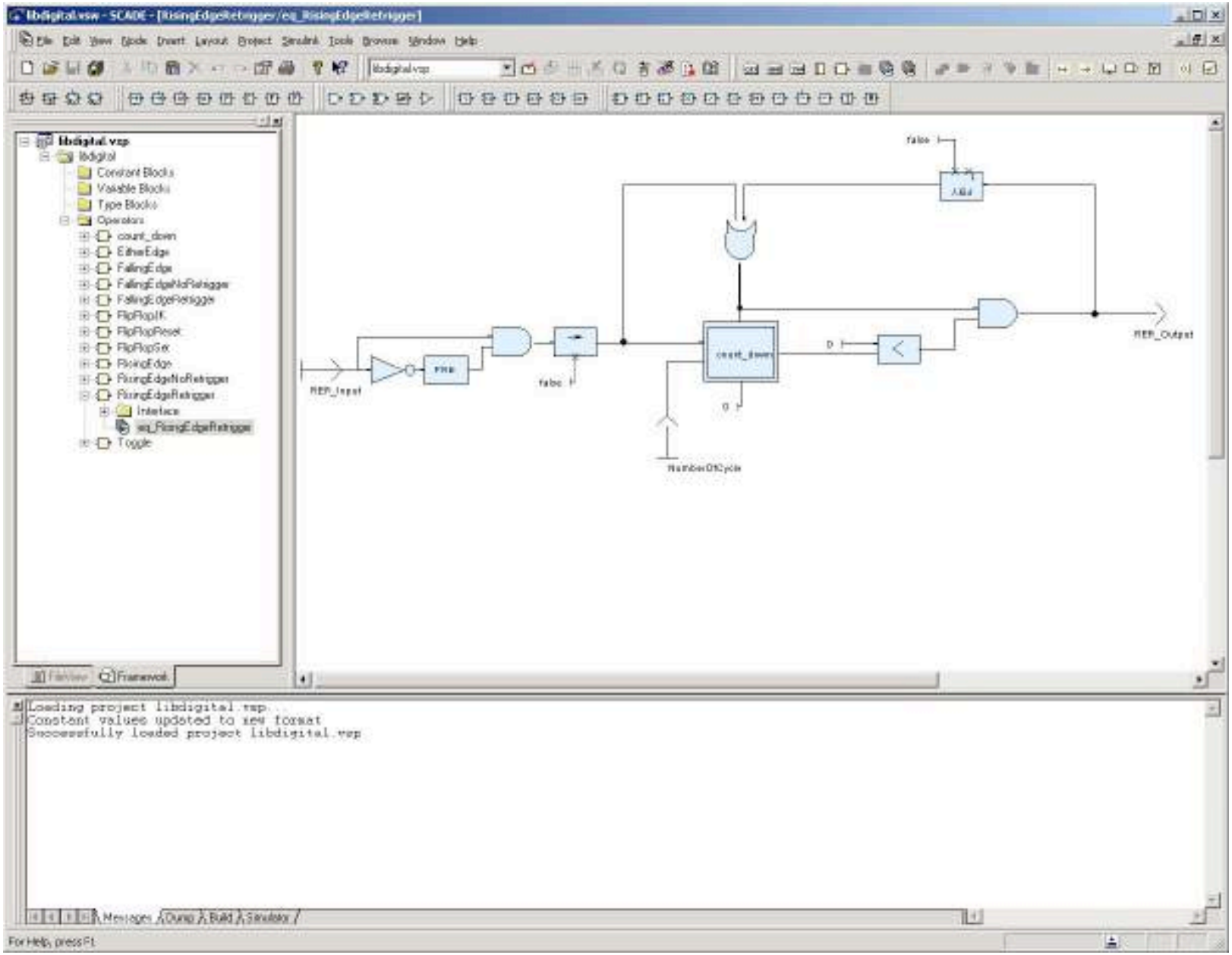
Un programme Lustre est une spécification précise.

Une résonnance immédiate avec la pratique industrielle:

- SAO (Spécification Assistée par Ordinateur) - Airbus.
- Saga - Merlin Gerin

Puis SCADE (1995-) - un environnement fondé sur le langage Lustre.

SCADE: Safety Critical Application Dev. Env. (Verilog, 95)



Au même moment...

A Rennes: le langage Signal (1987) [BLJ91]

Même influence et approche que Lustre mais bien plus expressif.

Un système : une relation entre suites.

Ecrire des spécifications (partielles, non déterministes) d'un système.

Un programme Signal permet de spécifier l'interface d'un composant.

Etudier les relations de raffinement.

Outil industriel Sildex: fondé sur Signal (TNI-Software puis DS).

À Nice: le langage Esterel (1984) [Gon88, BG92]

Systèmes dominés "contrôle" (e.g., machines de Mealy) et leur transcription en circuits séquentiels.

Informatique théorique (calcul de processus, lambda calcul, sémantique).

Plusieurs idées radicales:

- Un style de programmation plus proche de l'informatique: séquence, boucles, interruption, suspension d'une tâche, composition parallèle, hiérarchie, etc. pour gagner en expressivité.
- Plusieurs sémantiques, en forme SOS.
- Comment rendre cela portable et déterministe?

L'idée géniale d'Esterel

Faire comme si la machine calculait infiniment vite!

Concilier parallélisme (pour l'expressivité) et déterminisme (pour la sûreté) [3].

Une nouvelle découverte: compiler Esterel vers des circuits (Lustre).

```
module ABRO
input A, B, R;
output C
loop
   [await A || await B];
   emit C
each R;
end module
```

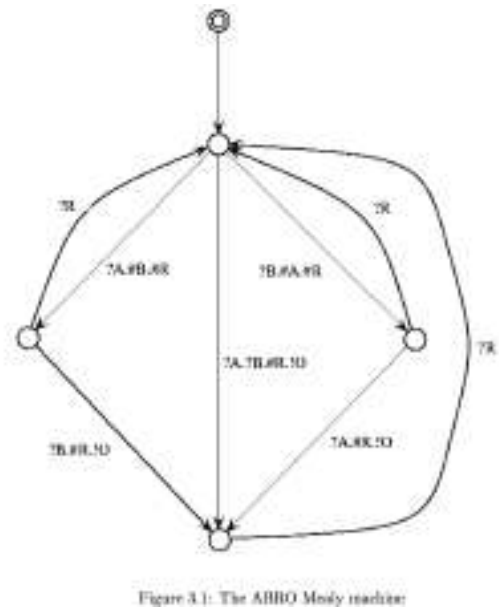


Figure 3.1: The ABRO Mealy machine

Outil industriel: Esterel-Studio puis start-up Esterel-Technologies (1999).

[3] "Write things once" (The Esterel Language Primer Version5.91, G. Berry, 2000).

Une même approche synchrone

Avec des styles de programmation très différents, tous ces langages partagent les mêmes principes

(1) raisonner idéalement;

(2) Compiler le parallélisme et calculer le WCET du code généré.

C'est bien plus simple.

Certains programmes sont des monstres...

comment les rejeter?

Démo

Et si on plongeait Lustre dans Haskell et OCaml pour voir?

Démo

Et si on plongeait Lustre dans Haskell et OCaml pour voir?

On parle de “shallow-embedding”: on décrit la sémantique des constructions d'un langage L1 en les exprimant avec les constructions d'un langage L2.

Différente échelles de temps

Synchroniser des processus lents et rapide?

X	1	2	3	4	5	...
half	*true*	*false*	*true*	*false*	*true*	...
X *when half*	1		3		5	...
$X + (X$ *when half*$)$	2		5		8	...

```
let half = true -> not (pre half);
    o = x + (x when half);
tel
```

Définit la suite: $\forall n \in \mathbb{N}.o_n = x_n + x_{2n}$

- ne peut être implémentée à mémoire (buffer) bornée;
- le rejeter statiquement: on peut le faire par typage [CP96, CP03].

Analyser les dépendances entre signaux

Des programmes ont zéro solutions (*deadlock*) ou trop (*non déterminisme*)

En Lustre/Signal

- `x = y + 1 and y = x + 2`
- `y = x and x = y`

Pour Lustre, un choix simple: "Causalité syntaxique" [CHPP87]:

"toute boucle doit traverser un retard", i.e., ordre partiel entre calculs.

Cette analyse peut être faite de manière modulaire, par typage et indépendemment du calcul d'horloge [CP01, BBC$^+$14]. Et on peut compiler en blocs indépendants [PR09].

Pour Signal, dépendances conditionnelles: "x dépend de y si c".

Détecter les "vrais" cycles. Combine calcul d'horloges et causalité; compilation plus complexe [ABG95].

Analyser les dépendances entre signaux

En Esterel

- `present S else emit S`
- `present S1 then emit S2 || present S2 then emit S1`
- ```
  present I then
    present O2 then emit O1 else present O1 then emit O2
  ```

Deux découvertes [Draft book'02] [4]

Sémantique constructive par calcul d'un point-fixe à chaque instant [Gon88, Ber02].

Reste t'il des signaux indéterminés?

Pour Esterel, la "bonne" notion de causalité est celle de l'électricité.
"Si on cable le programme synchrone, les sorties sont-elles stables?"

Coincide avec ce qui est démontrable en logique constructive [MSB12]

[4] The Constructive Semantics of Pure Esterel Draft Version 3, 2002, G. Berry.
  ```

Programmation synchrone fonctionnelle

Lucid Synchrone et ReactiveML

Une idée de Paul Caspi, en 1994, à Grenoble.

> *"Marc, observe bien, on peut écrire des programmes Lustre récursifs en quelques lignes de LazyML!"*

Très expressif: ordre supérieur, inférence des types, récursivité, etc.

mais les "monstres" sont toujours la.

du typage, du typage, du typage... et adapter la compilation.

Lucid Synchrone (95-06) [CP96, Pou06]

Construire un langage synchrone fonctionnel avec des traits de ML

ReactiveML (05-15) [MP05]

Du parallélisme synchrone dans un langage à la ML (OCaml)

Modèle de Boussinot [Bou91]: réaction retardée à l'absence.

Le temps a passé...

Le temps a passé...

Un foisonnement de nouveaux langages et de compilateurs: Quartz, SCcharts, Heptagon, Scade 6, Lustre V6, Lucid Synchrone, LustreC, Vélus, Zélus, etc.

Le temps a passé...

Un foisonnement de nouveaux langages et de compilateurs: Quartz, SCcharts, Heptagon, Scade 6, Lustre V6, Lucid Synchrone, LustreC, Vélus, Zélus, etc.

des compilateurs optimisant,

Le temps a passé...

Un foisonnement de nouveaux langages et de compilateurs: Quartz, SCcharts, Heptagon, Scade 6, Lustre V6, Lucid Synchrone, LustreC, Vélus, Zélus, etc.

des compilateurs optimisant,

des systèmes de types dédiés: analyse de causalité, calcul d'horloges, etc.

Le temps a passé...

Un foisonnement de nouveaux langages et de compilateurs: Quartz, SCcharts, Heptagon, Scade 6, Lustre V6, Lucid Synchrone, LustreC, Vélus, Zélus, etc.

des compilateurs optimisant,

des systèmes de types dédiés: analyse de causalité, calcul d'horloges, etc.

des constructions nouvelles, des formalisations mécanisées de sémantiques et de compilateurs,

Le temps a passé...

Un foisonnement de nouveaux langages et de compilateurs: Quartz, SCcharts, Heptagon, Scade 6, Lustre V6, Lucid Synchrone, LustreC, Vélus, Zélus, etc.

des compilateurs optimisant,

des systèmes de types dédiés: analyse de causalité, calcul d'horloges, etc.

des constructions nouvelles, des formalisations mécanisées de sémantiques et de compilateurs,

des outils de simulation, de test, de vérification formelle,

Le temps a passé...

Un foisonnement de nouveaux langages et de compilateurs: Quartz, SCcharts, Heptagon, Scade 6, Lustre V6, Lucid Synchrone, LustreC, Vélus, Zélus, etc.

des compilateurs optimisant,

des systèmes de types dédiés: analyse de causalité, calcul d'horloges, etc.

des constructions nouvelles, des formalisations mécanisées de sémantiques et de compilateurs,

des outils de simulation, de test, de vérification formelle,

des extensions pour descrire des modèle hybrides,

Le temps a passé...

Un foisonnement de nouveaux langages et de compilateurs: Quartz, SCcharts, Heptagon, Scade 6, Lustre V6, Lucid Synchrone, LustreC, Vélus, Zélus, etc.

des compilateurs optimisant,

des systèmes de types dédiés: analyse de causalité, calcul d'horloges, etc.

des constructions nouvelles, des formalisations mécanisées de sémantiques et de compilateurs,

des outils de simulation, de test, de vérification formelle,

des extensions pour descrire des modèle hybrides,

et des outils industriels qui ont changé radicalement

Le temps a passé...

Un foisonnement de nouveaux langages et de compilateurs: Quartz, SCcharts, Heptagon, Scade 6, Lustre V6, Lucid Synchrone, LustreC, Vélus, Zélus, etc.

des compilateurs optimisant,

des systèmes de types dédiés: analyse de causalité, calcul d'horloges, etc.

des constructions nouvelles, des formalisations mécanisées de sémantiques et de compilateurs,

des outils de simulation, de test, de vérification formelle,

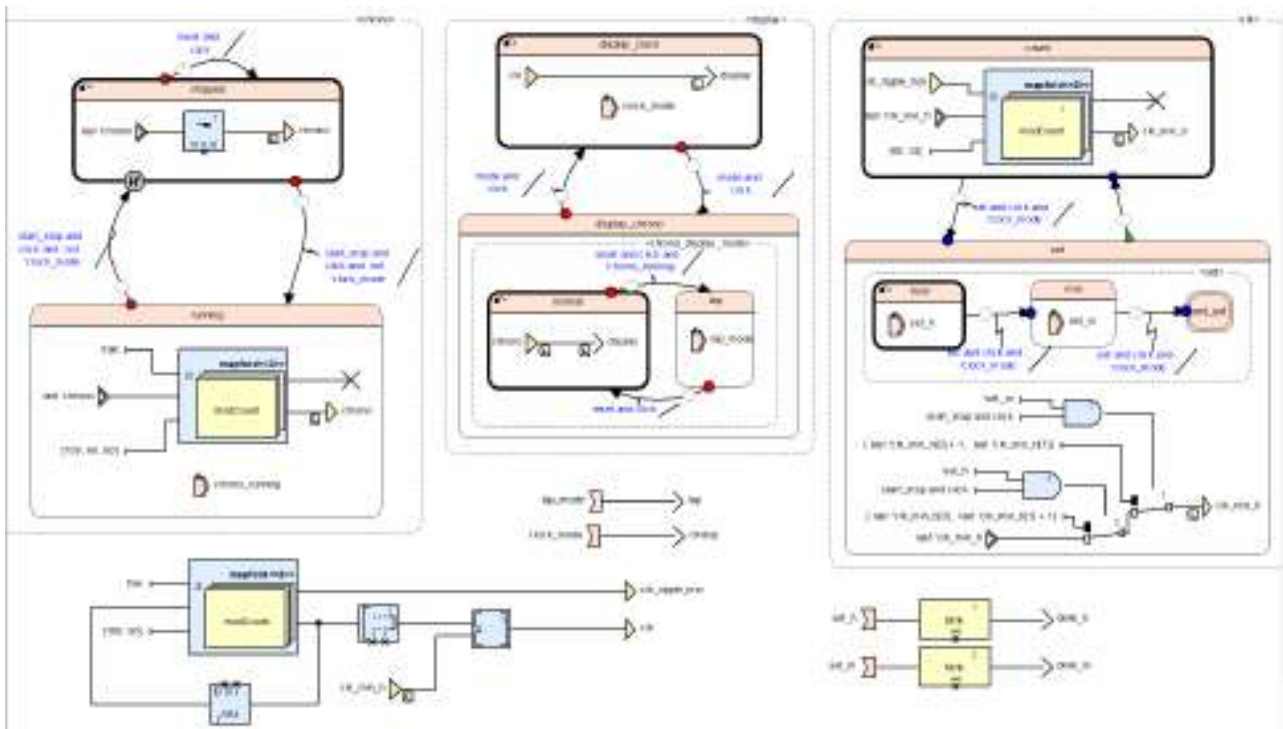
des extensions pour descrire des modèle hybrides,

et des outils industriels qui ont changé radicalement

Le modèle synchrone est aujourd'hui un standard industriel pour le logiciel de contrôle/commande critique.

Scade/KCG 6 *[TASE'17]* [5]

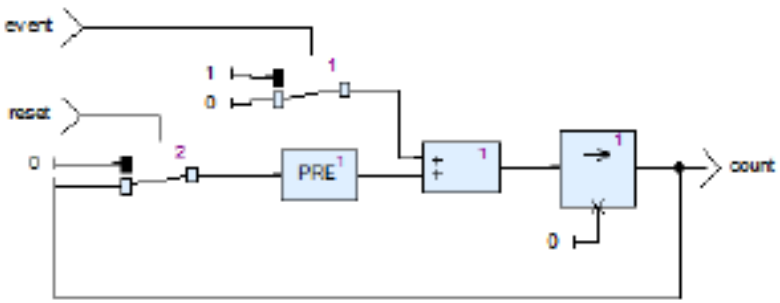
- Un nouveau langage en 2008; combinant de manière originale des traits de Lustre, Lucid Synchrone et Esterel.
- Utilisé dans +100 projets *qualifiés* de logiciel critique (avions, trains).



[5] www.ansys.com/fr-fr/products/embedded-software/ansys-scade-suite

Tes dessins sont justes? Prouve le!

‘What you prove is what you execute’ (Berry ’89)




```
L1 = pre L7;
L2 = (L11) -> (L5);
L3 = event;
L4 = reset;
count = L2;
L5 = L6 + L1;
L6 = if L3 then (L8) else (L9);
L7 = if L4 then (L10) else (L2);
L8 = 1;
L9 = 0;
L10 = 0;
L11 = 0;
```

code gen.

```
void counter_reset(outC_counter *outC)
{
  outC->init = kcg_true;
}

void counter(inC_counter *inC, outC_counter *o
{
  kcg_int tmp;

  if (outC->init) {
    outC->count = 0;
  }
  else {
    if (inC->event) {
      tmp = 1;
    }
    else {
      tmp = 0;
    }
    outC->count = tmp + outC->_L9;
  }
  if (inC->reset) {
    outC->_L9 = 0;
  }
  else {
    outC->_L9 = outC->count;
  }
  outC->init = kcg_false;
}
```

Un compilateur certifié? une idée née en 1992/1993

Scade: un langage issu de Lustre (labo. commun VERILOG/IMAG = VERIMAG).

Réaliser un compilateur certifié pour les normes les plus strictes de l'avionique.

Éviter d'avoir à revérifier que le code généré est correct vis-à-vis du source.

En s'appuyant sur définition simple et précise de Lustre et de sa génération de code.

Les évolutions majeures du langage (Scade 6) et du compilateur ont toutes été guidées par cet objectif.

Une obligation de moyen vs une obligation de résultat (e.g., CompCert).

Un compilateur certifié? une idée née en 1992/1993

Scade: un langage issu de Lustre (labo. commun VERILOG/IMAG = VERIMAG).

Réaliser un compilateur certifié pour les normes les plus strictes de l'avionique.

Éviter d'avoir à revérifier que le code généré est correct vis-à-vis du source.

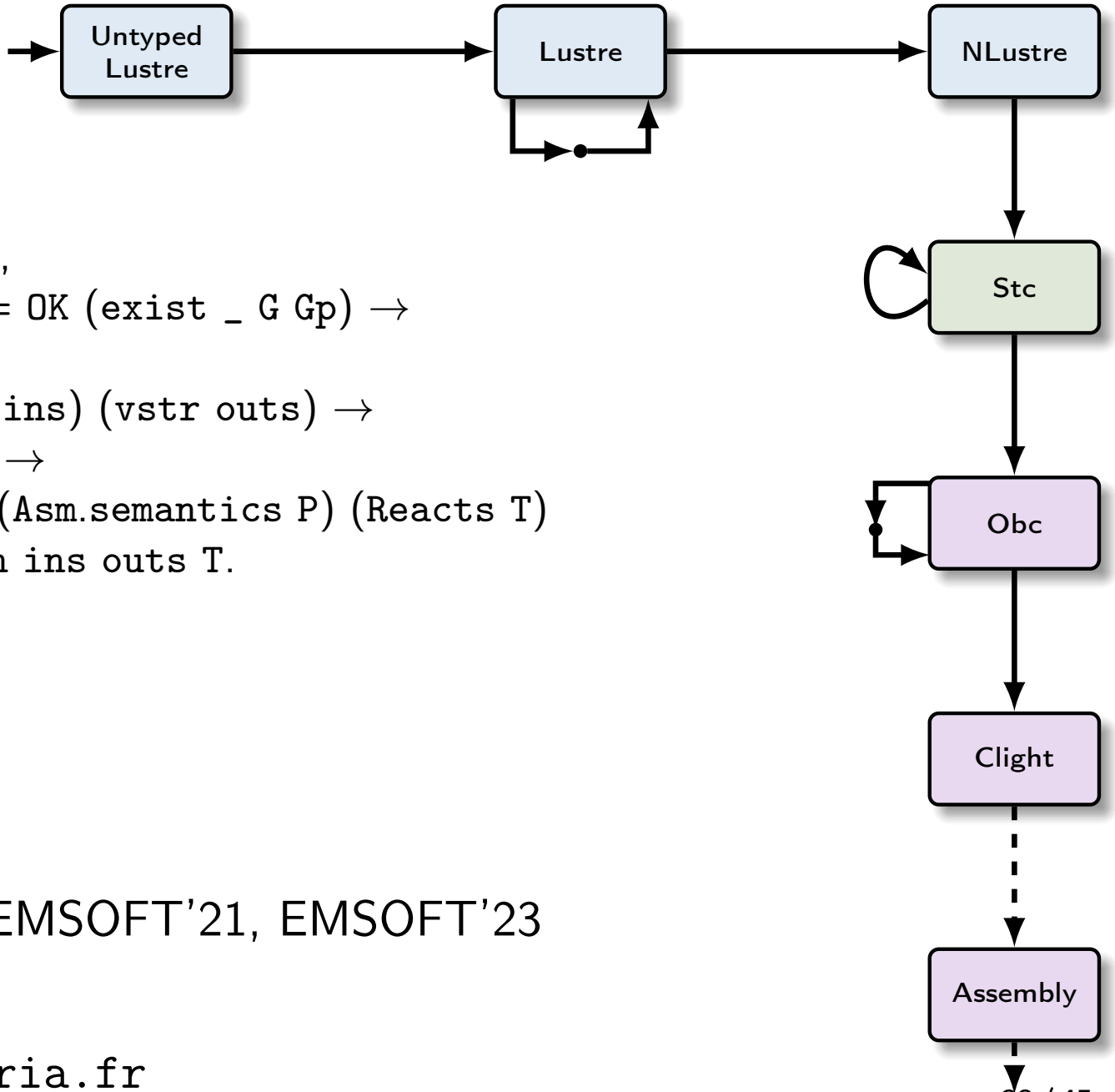
En s'appuyant sur définition simple et précise de Lustre et de sa génération de code.

Les évolutions majeures du langage (Scade 6) et du compilateur ont toutes été guidées par cet objectif.

Une obligation de moyen vs une obligation de résultat (e.g., CompCert).

Peut-on (et comment) développer un compilateur synchrone prouvé?

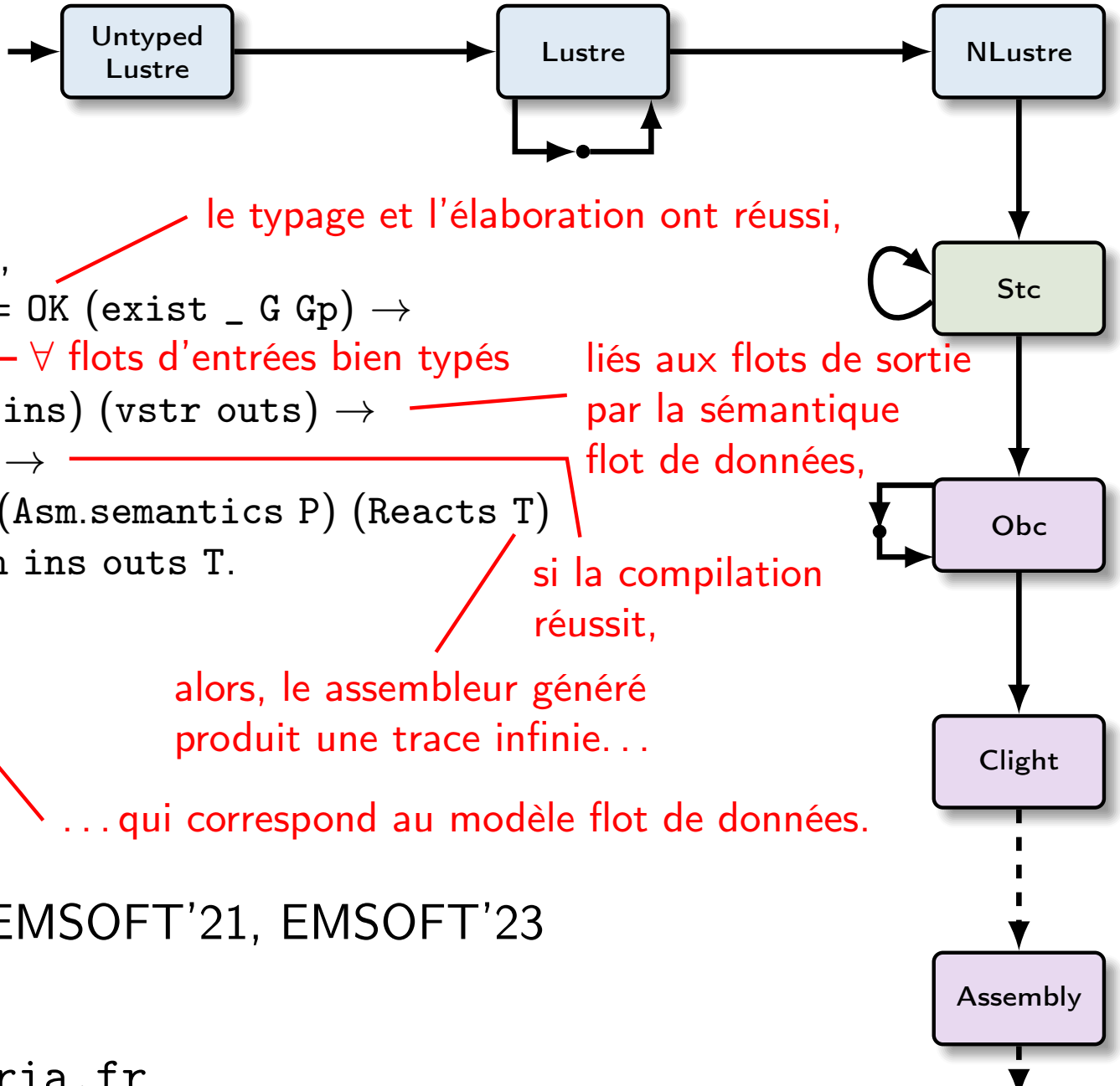
Vélus: un compilateur Lustre prouvé vers CompCert




```
Theorem behavior_asm:
  ∀ D G Gp P main ins outs,
    elab_declarations D = OK (exist _ G Gp) →
    wt_ins G main ins →
    sem_node G main (vstr ins) (vstr outs) →
    compile D main = OK P →
    ∃ T, program_behaves (Asm.semantics P) (Reacts T)
         ∧ bisim_io G main ins outs T.
```

- PLDI'17,POPL'20, EMSOFT'21, EMSOFT'23
- 100kLOC de Coq.
- `https://velus.inria.fr`

Vélus: un compilateur Lustre prouvé vers CompCert




```
Theorem behavior_asm:
  ∀ D G Gp P main ins outs,
    elab_declarations D = OK (exist _ G Gp) →
    wt_ins G main ins →
    sem_node G main (vstr ins) (vstr outs) →
    compile D main = OK P →
    ∃ T, program_behaves (Asm.semantics P) (Reacts T)
         ∧ bisim_io G main ins outs T.
```

le typage et l'élaboration ont réussi,

– ∀ flots d'entrées bien typés

liés aux flots de sortie par la sémantique flot de données,

si la compilation réussit,

alors, le assembleur généré produit une trace infinie. . .

. . . qui correspond au modèle flot de données.

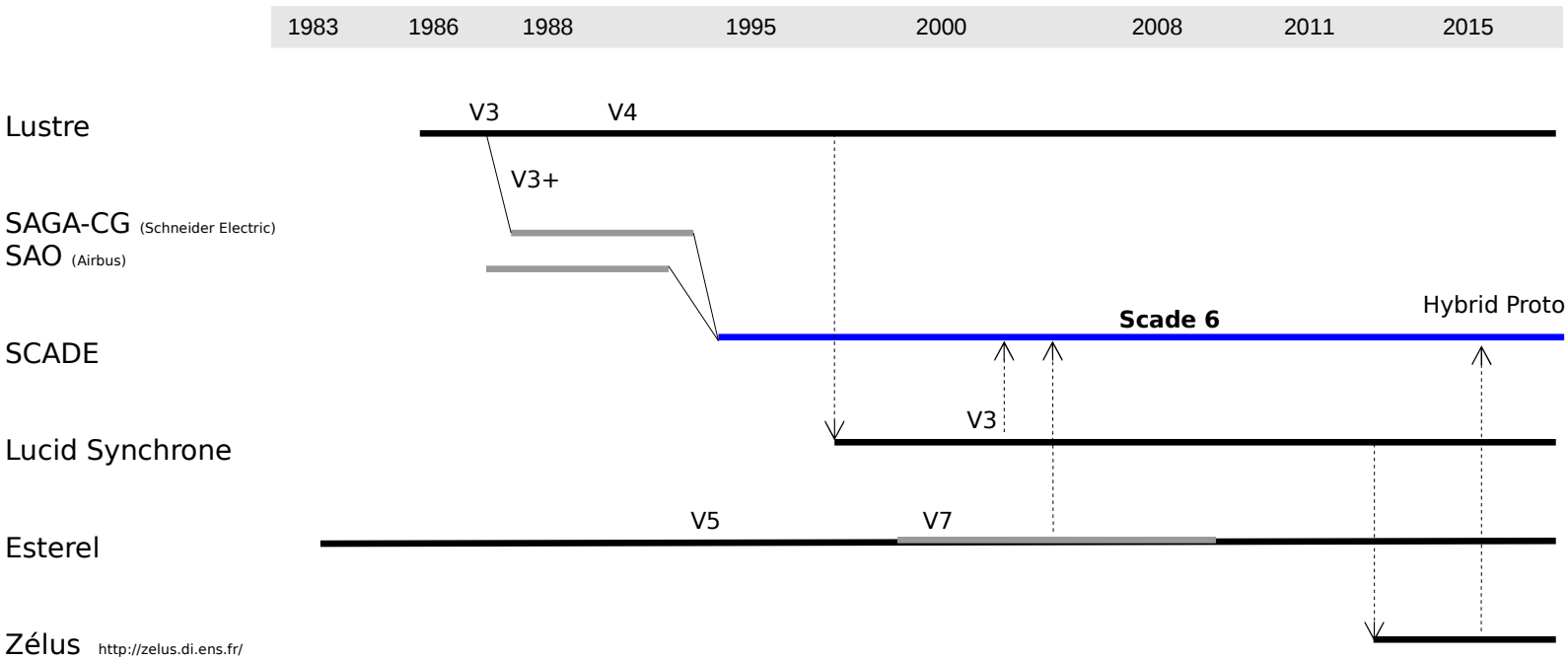
- PLDI'17,POPL'20, EMSOFT'21, EMSOFT'23
- 100kLOC de Coq.
- `https://velus.inria.fr`

Le moteur pas-à-pas ?

`https://vertmo.org/jsofocaml/try-velus/` [6]

[6] Merci à Timothy Bourke et Basile Pesin!

Timeline


1983
1986
1988
1995
2000
2008
2011
2015
Lustre
V3
V4
V3+
SAGA-CG (Schneider Electric)
SAO (Airbus)
SCADE
Scade 6
Hybrid Proto
Lucid Synchrone
V3
Esterel
V5
V7
Zélus http://zelus.di.ens.fr/

Conclusion

- Des langages dédiés, parallèles et déterministes;
- adaptés à la culture mathématique et la pratique des ingénieurs;
- propres et adoptés dès le début.
- Ne pas transiger sur les principes: sémantique statique et dynamique spécifiés en détail ainsi que le compilateur.
- Une évolution continue des langages, des méthodes pour les compiler, les vérifier.
- Des idées utilisées dans d'autres applications: web, trading haute fréquence, musique mixte, ChatBot, etc.
- Un impact direct vers des outils industriels utilisés tous les jours.
- Certains pour réaliser du logiciel; d'autre pour spécifier/vérifier des propriétés de systèmes implémentés autrement.

Et maintenant ?

- Est-ce que les spécifications en Coq de la sémantique statique et dynamique peuvent compléter/remplacer les étapes de vérification (humaines) actuelles?
- Compilation intégrant de la validation de traduction, du test indépendant avec des sémantiques exécutables.
- Relacher certaines contraintes de synchronisme mais contrôler les latences de bout-en-bout (pratique Airbus).
- Exprimer des contraintes de temps réel et les exploiter pour générer du code séquentiel, en tâches ou parallèle.
- Écrire des modèle hybrides.
- Exprimer des algorithmes combinant du contrôle et du calcul intensif.

Pour aller plus loin

Cours de M2 du MPRI: 2.23-1 “Systèmes réactifs synchrones” à l’automne.
`https://wikimpri.dptinfo.ens-cachan.fr`

References I

T. Amagbegnon, L. Besnard, and P. Le Guernic.

Implementation of the data-flow synchronous language signal.
In *Programming Languages Design and Implementation (PLDI)*, pages 163–173. ACM, 1995.

E. A. Ashcroft and W. W. Wadge.

Lucid, the data-flow programming language.
A.P.I.C. Studies in Data Processing, Academic Press, 1985.

Albert Benveniste, Timothy Bourke, Benoit Caillaud, Bruno Pagano, and Marc Pouzet.

A Type-based Analysis of Causality Loops in Hybrid Systems Modelers.
In *International Conference on Hybrid Systems: Computation and Control (HSCC)*, Berlin, Germany, April 15–17 2014. ACM.

Gérard Berry.

The constructive semantics of pure esterel, draft, version 3.
Draft book. Available at:
`http://www-sop.inria.fr/members/Gerard.Berry/Papers/EsterelConstructiveBook.pdf`, 2002.

G. Berry and G. Gonthier.

The Esterel synchronous programming language, design, semantics, implementation.
Science of Computer Programming, 19(2):87–152, 1992.

A. Benveniste, P. LeGuernic, and Ch. Jacquemot.

Synchronous programming with events and relations: the SIGNAL language and its semantics.
Science of Computer Programming, 16:103–149, 1991.

F. Boussinot.

Reactive C: An Extension of C to Program Reactive Systems.
Software Practice and Experience, 21(4):401–428, 1991.

References II

P. Caspi, N. Halbwachs, D. Pilaud, and J. Plaice.

Lustre: a declarative language for programming synchronous systems.
In *14th ACM Symposium on Principles of Programming Languages*. ACM, 1987.

Paul Caspi and Marc Pouzet.

Synchronous Kahn Networks.
In *ACM SIGPLAN International Conference on Functional Programming (ICFP)*, Philadelphia, Pensylvania, May 1996.

Pascal Cuoq and Marc Pouzet.

Modular Causality in a Synchronous Stream Language.
In *European Symposium on Programming (ESOP'01)*, Genova, Italy, April 2001.

Jean-Louis Colaço and Marc Pouzet.

Clocks as First Class Abstract Types.
In *Third International Conference on Embedded Software (EMSOFT'03)*, Philadelphia, Pennsylvania, USA, october 2003.

Georges Gonthier.

Sémantiques et modèles d'exécution des langages réactifs synchrones.
PhD thesis, Université d'Orsay, 1988.

Gilles Kahn.

The semantics of a simple language for parallel programming.
In *IFIP 74 Congress*. North Holland, Amsterdam, 1974.

Gilles Kahn and David B. MacQueen.

Coroutines and networks of parallel processes.
In *IFIP Congress*, pages 993–998, 1977.

References III

Louis Mandel and Marc Pouzet.

ReactiveML, a Reactive Extension to ML.
In *ACM International Conference on Principles and Practice of Declarative Programming (PPDP)*, Lisboa, July 2005.
Recipient of the price for the "most influential PPDP'05 paper" given in July 2015 at PPDP'15.

Michael Mendler, Thomas R. Shiple, and Gérard Berry.

Constructive boolean circuits and the exactness of timed ternary simulation.
Form. Methods Syst. Des., 40(3):283–329, June 2012.

Marc Pouzet.

Lucid Synchrone, version 3. Tutorial and reference manual.
Université Paris-Sud, LRI, April 2006.
Distribution available at: `https://www.di.ens.fr/~pouzet/lucid-synchrone/`.

Marc Pouzet and Pascal Raymond.

Modular Static Scheduling of Synchronous Data-flow Networks: An efficient symbolic representation.
In *ACM International Conference on Embedded Software (EMSOFT'09)*, Grenoble, France, October 2009.