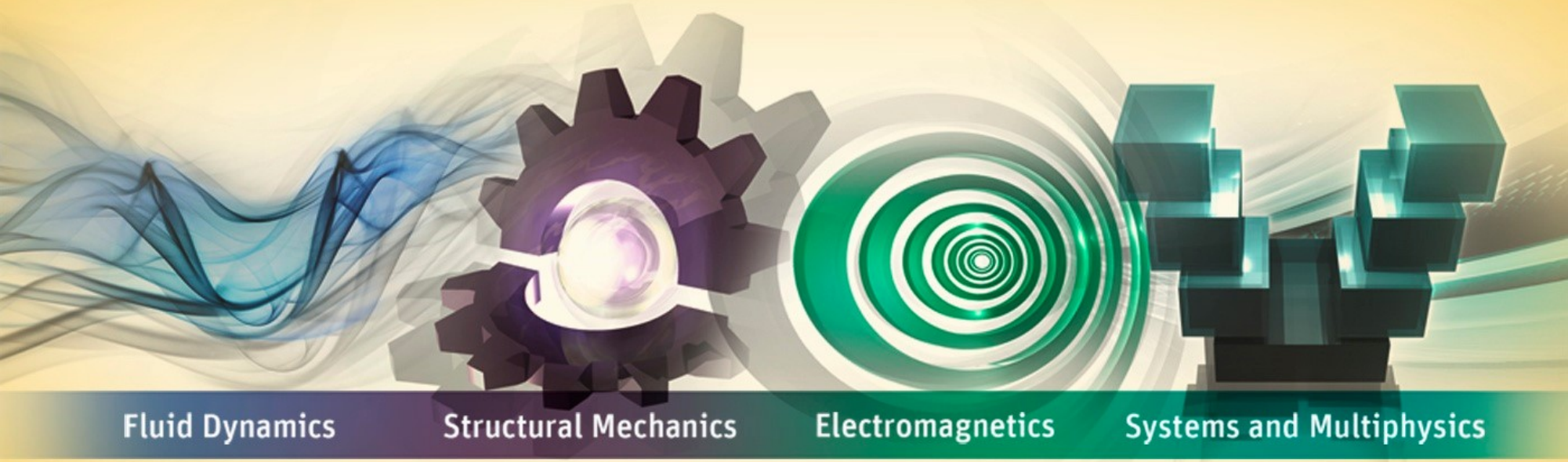
ANSYS

Realize Your Product Promise®

SCADE: un langage formel pour l'embarqué critique



Jean-Louis Colaço

28 avril 2014

ANSYS ESTEREL Technologies

Application embarquée critique

Applications pour lesquelles un "bug" peut entraîner:

- une importante perte financière (mission critical) ex. spatial
- un risque pour la vie des usagers (safety critical) ex. transports

Chaque domaine d'application est soumis à des standards exigeant des obligations de moyens dans le processus de développement. Des autorités de certification indépendantes auditent les projets par rapports au standard avant de décider d'une mise en service.

Méthodes Formelles: ce qu'en disent les standards

Formal Methods Supplement to DO-178C and DO-278A:

"Establishing a formal model of the software artifact of interest is fundamental to all formal methods. In general a model is an abstract representation of a given set of aspects of the software that is used for analysis, simulation, and/or code generation. In the context of this document, to be formal, a model should have an unambiguous, mathematically defined syntax and semantics. This makes it possible to use automated means to obtain guarantees that the model has certain specified properties."

SCADE

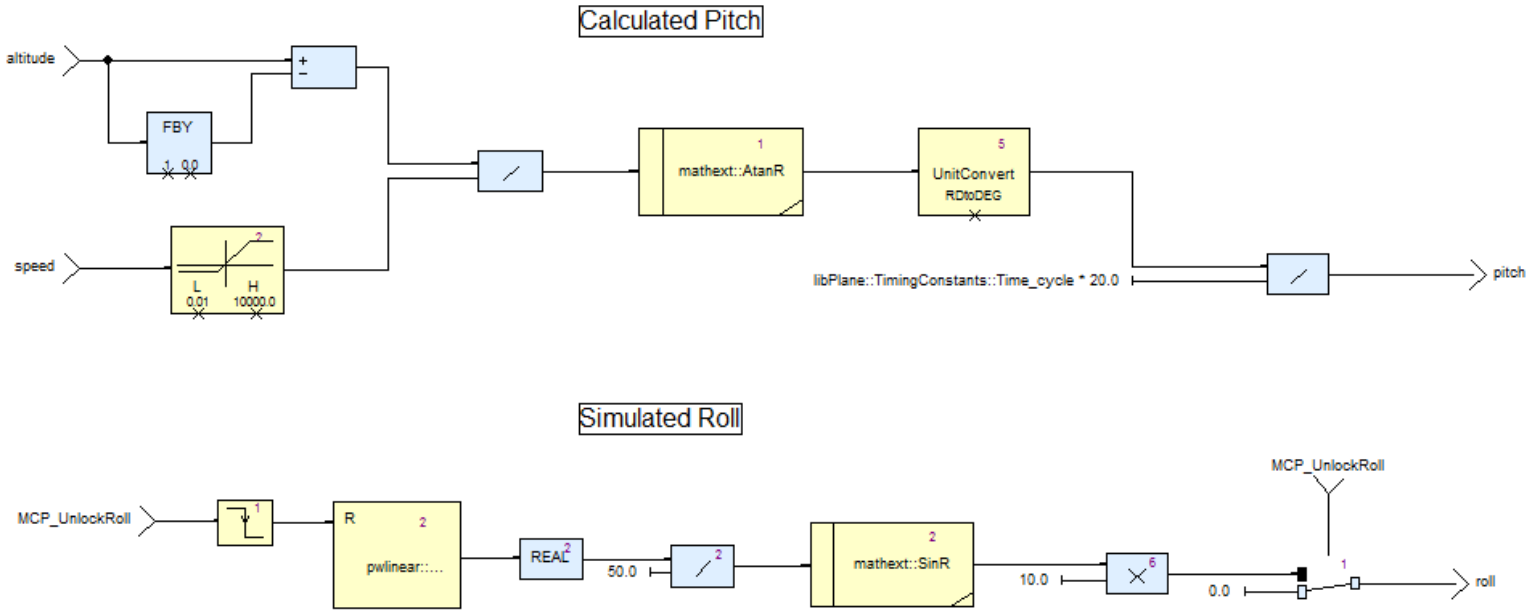
- **S**afety **C**ritical **A**pplication **D**evelopment **E**nvironment
- SCADE est le langage de modélisation de SCADE Suite
- appartient à la famille des **langages synchrones**
- dialecte du language LUSTRE (flot de données) avec des extensions majeures introduites dans la version 6.
- DSL dedié au développement de **systèmes reactifs**

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.

Les reponses qu'apporte SCADE

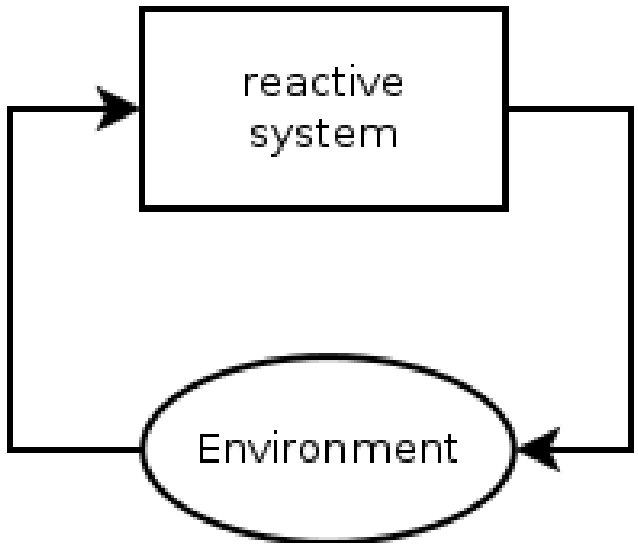
- un langage formel,
- une notation graphique,
- un générateur de code qualifié,
- la possibilité de vérifier formellement des propriétés.

SCADE 5 example



Système Réactif

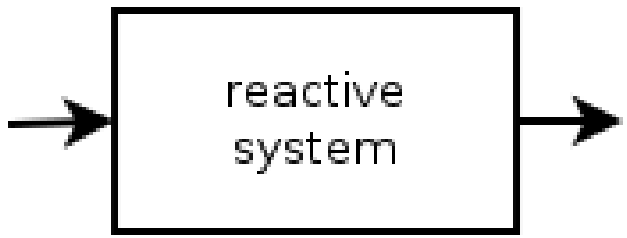
Système en interaction permanente avec son environnement (utilisateurs, physique, ...)



Point de vue Mathématique

Un système réactif peut être vu comme une fonction des suites dans les suites.

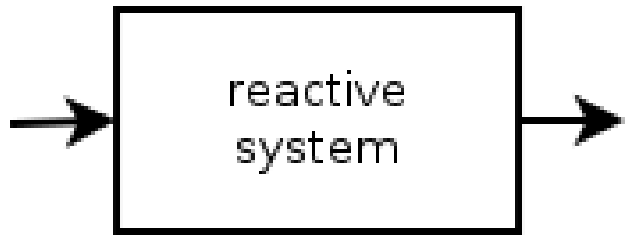
$i_0, i_1, \ldots, i_n, \ldots$ $o_0, o_1, \ldots, o_n, \ldots$



SCADE est un langage de définition de suites récursives.

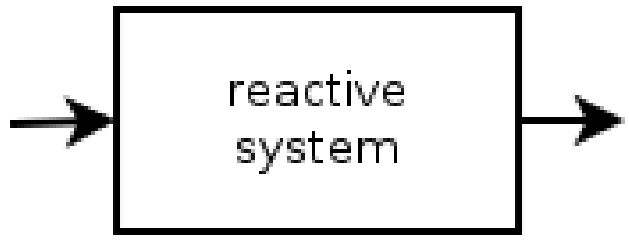
Point de vue opérationnel

$i_0, i_1, \ldots, i_n, \ldots$ S_0 $o_0, o_1, \ldots, o_n, \ldots$



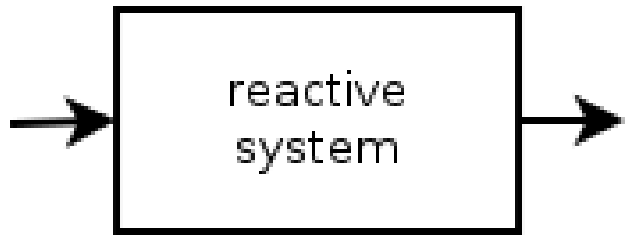
Point de vue opérationnel

$i_0, i_1, \ldots, i_n, \ldots$ S_1 $o_0, o_1, \ldots, o_n, \ldots$

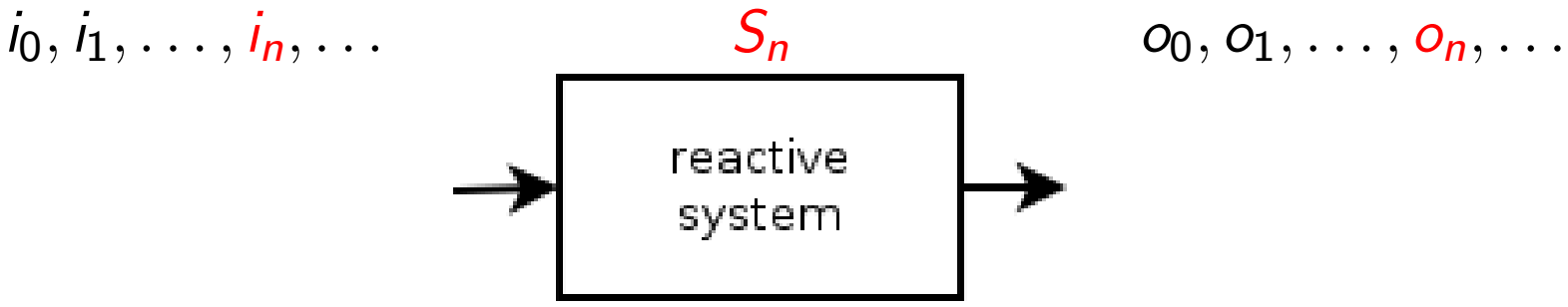


Point de vue opérationnel

$i_0, i_1, \ldots, i_n, \ldots$ S_n $o_0, o_1, \ldots, o_n, \ldots$



Point de vue opérationnel



A chaque instant, les valeurs en entrée sont lues, les sorties sont calculées et l'état interne est mis à jour. Soit f la fonction de transition qui calcule une réaction:

$$o_n, S_{n+1} = f(i_n, S_n)$$

Le générateur de code de SCADE produit un code séquentiel qui calcule f et la structure de donnée pour représenter l'état S.

Le noyau flot de données: operateurs combinatoires

Extension point à point des opérations combinatoires:

x	x_0	x_1	$\ldots$	x_n	$\ldots$
y	y_0	y_1	$\ldots$	y_n	$\ldots$
x + y	$x_0 + y_0$	$x_1 + y_1$	$\ldots$	$x_n + y_n$	$\ldots$

x+y represente la suite $(x_n + y_n)_{n \in \mathbb{N}}$

Le noyau flot de données: operateurs combinatoires

Extension point à point des opérations combinatoires:

`x`	x_0	x_1	$\ldots$	x_n	$\ldots$
`y`	y_0	y_1	$\ldots$	y_n	$\ldots$
`x + y`	$x_0 + y_0$	$x_1 + y_1$	$\ldots$	$x_n + y_n$	$\ldots$

`x+y` represente la suite $(x_n + y_n)_{n \in \mathbb{N}}$

Idem pour: `not`, `and`, `or`, `-`, `*`, ...

Le noyau flot de données: operateurs combinatoires

Extension point à point des opérations combinatoires:

`x`	x_0	x_1	$\ldots$	x_n	$\ldots$
`y`	y_0	y_1	$\ldots$	y_n	$\ldots$
`x + y`	$x_0 + y_0$	$x_1 + y_1$	$\ldots$	$x_n + y_n$	$\ldots$

`x+y` represente la suite $(x_n + y_n)_{n \in \mathbb{N}}$
Idem pour: `not`, `and`, `or`, `-`, `*`, ...
Constantes et litéraux représentent des suites constantes

`2`	2	2	$\ldots$	2	$\ldots$
`x`	x_0	x_1	$\ldots$	x_n	$\ldots$
`2 * x`	$2 * x_0$	$2 * x_1$	$\ldots$	$2 * x_n$	$\ldots$

Le noyau flot de données: retard

`x`	x_0	x_1	...	x_n	...
`pre x`	*nil*	x_0	...	x_{n-1}	...

Le noyau flot de données: retard

`x`	x_0	x_1	$\ldots$	x_n	$\ldots$
`pre x`	*nil*	x_0	$\ldots$	x_{n-1}	$\ldots$

soit `x` la suite $(x_n)_{n\in\mathbb{N}}$, `pre x` représente la suite $(p_n)_{n\in\mathbb{N}}$ définie par:

$$p_0 = \mathit{nil} \text{ et } \forall n \in \mathbb{N}, p_{n+1} = x_n$$

où *nil* est une valeur indéterminée (mais dans le type).

Le noyau flot de données: initialisation

	x	x_0	x_1	...	x_n	...
	y	y_0	y_1	...	y_n	...
x ->	y	x_0	y_1	...	y_n	...

Le noyau flot de données: initialisation

`x`	x_0	x_1	...	x_n	...
`y`	y_0	y_1	...	y_n	...
`x -> y`	x_0	y_1	...	y_n	...

combiné avec pre, on peut construit un flot retardé sans *nil*:

`x`	x_0	x_1	...	x_n	...
`pre y`	*nil*	y_0	...	y_{n-1}	...
`x -> pre y`	x_0	y_0	...	y_{n-1}	...

Le noyau flot de données: Horloges

Filtrage (sous-suite):

h	true	false	true	true	false	...
x	x_0	x_1	x_2	x_3	x_4	...
x when h	x_0	–	x_2	x_3	–	...

Le noyau flot de données: Horloges

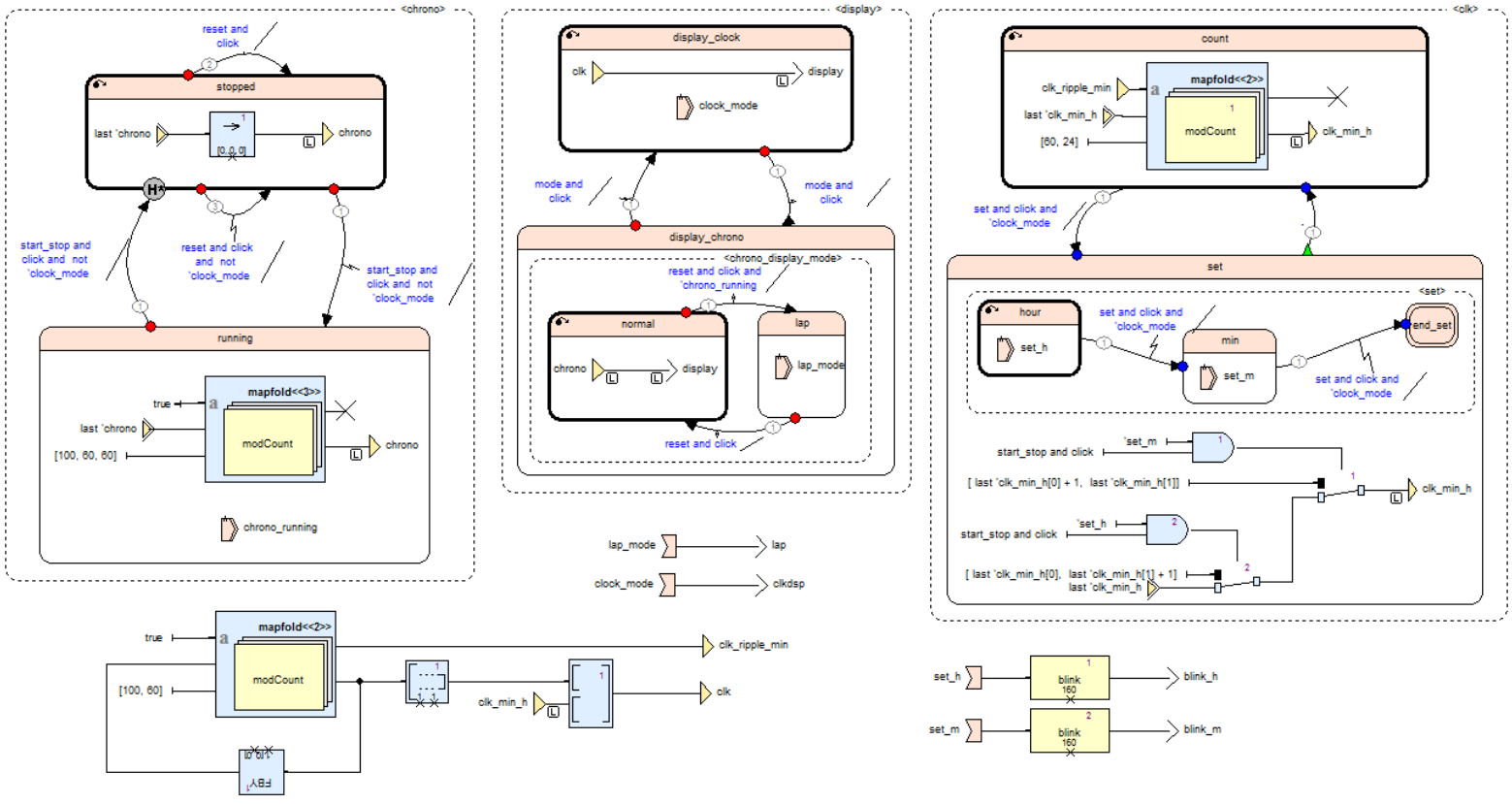
Filtrage (sous-suite):

h	true	false	true	true	false	...
x	x_0	x_1	x_2	x_3	x_4	...
x when h	x_0	–	x_2	x_3	–	...

Entrelacement de flots complémentaires:

h	true	false	true	true	false	...
a	a_0	–	a_1	a_2	–	...
b	–	b_0	–	–	b_1	...
merge (h; a; b)	a_0	b_0	a_1	a_2	b_1	...

SCADE 6 example



Sémantique:

- par traduction dans le noyau flot de données;
- permet de s'assurer que l'extension est conservative (n'altère pas le noyau flot de données);
- la compilation suit ce schéma.

SCADE 6

Sémantique:

- par traduction dans le noyau flot de données;
- permet de s'assurer que l'extension est conservative (n'altère pas le noyau flot de données);
- la compilation suit ce schéma.

Inspiration:

- Lucid synchrone (http://www.di.ens.fr/ pouzet/lucid-synchrone/, Thèse de Grégoire Hamon);
- Esterel (G. Berry et G. Gontier) et Syncharts (C. André).

Jean-Louis Colaço and Bruno Pagano and Marc Pouzet.
A Conservative Extension of Synchronous Data-flow with State Machines.
In *ACM International Conference on Embedded Software (EMSOFT'05)*

Typage

$$\frac{}{H;S_z;C \overset{0}{\vdash} \textbf{true} : \textbf{bool}}\ [\text{LRM-149}]\ (\textsc{True}) \qquad \frac{}{H;S_z;C \overset{0}{\vdash} \textbf{false} : \textbf{bool}}\ [\text{LRM-150}]\ (\textsc{False})$$

The type of the boolean values **true** and **false** is **bool**.

$$\frac{}{H;S_z;C \overset{0}{\vdash} \textsc{char} : \textbf{char}}\ [\text{LRM-151}]\ (\textsc{Char}) \qquad \frac{}{H;S_z;C \overset{0}{\vdash} \textsc{integer} : \textbf{int}}\ [\text{LRM-152}]\ (\textsc{Integer}) \qquad \frac{}{H;S_z;C \overset{0}{\vdash} \textsc{float} : \textbf{real}}\ [\text{LRM-153}]\ (\textsc{Real})$$

These three rules give the type associated to the three value kinds: CHAR, INTEGER and FLOAT.

$$\frac{}{H;S_z;C \overset{0}{\vdash} (\textsc{integer} : 't) : 't \ \text{ where } 't \text{ is } \textbf{num}}\ [\text{LRM-643}]\ (\textsc{Polymorphic literal})$$

An integer value can be used as a polymorphic literal. Its type must be a numerical type.

$$\frac{H;S_z;C \overset{k}{\vdash} f : \tau_1 \xrightarrow{k_1} \tau_2, k_1 \leq k \quad H;S_z;C \overset{k}{\vdash} e : \tau_1' \quad C \vdash \tau_1' \sqsubseteq \tau_1}{H;S_z;C \overset{k}{\vdash} f(e) : \tau_2}\ [\text{LRM-154}]\ (\textsc{Instance})$$

An operator f can be instantiated with an expression e if the type of e matches the types of the arguments of f; the type of the instance is the output type of f. The expression $f(e)$ must be typable in a context that has at least the memory of f ($k_1 \leq k$).

les valeurs à l'entrée d'un opérateur sont du type attendu.

Contrôle des horloges

$$\text{[LRM-252]}\ (\textsc{Clk Operator Specialisation})\quad \frac{H \overset{clf}{\vdash} f : \forall\alpha.\forall X_1,\ldots,X_n.cl_1 \longrightarrow cl_2}{H \overset{clf}{\vdash} f : cl_1 \longrightarrow cl_2[clf'/\alpha][m_i/X_i]_{i\in[1..n]}}$$

A polymorphic operator signature can be specialized by substituting the quantified clock variable α by a clock type clf' and the carrier variables X_i by carrier names m_i.

$$\text{[LRM-253]}\ (\textsc{Clk Instance})\quad \frac{H \overset{clf}{\vdash} f : cl_1 \longrightarrow cl_2 \quad H \overset{clf}{\vdash} e : cl_1}{H \overset{clf}{\vdash} f(e) : cl_2}$$

An operator f with a clock signature $cl_1 \longrightarrow cl_2$ can be instantiated with parameters e of clock type cl_1.

Les flots combinés sont de la "longueur" attendue
Corollaire: pas de buffers $\Rightarrow$ mémoire finie.

Jean-Louis Colaço and Marc Pouzet.

Clocks as first class abstract types.

In *Third International Conference on Embedded Software (EMSOFT'03)*

Analyse de Causalité

$$\text{[LRM-330]}\ (\text{Dep Operator Specialisation})\quad \frac{H; H_{last}; W; C \vdash f : \forall \gamma_1, \ldots, \gamma_n . \gamma_1 \times \cdots \times \gamma_n \longrightarrow ct \quad \forall i \in [1..n], \gamma'_i \notin FCV(H)}{H; H_{last}; W; C \vdash f : \gamma'_1 \times \cdots \times \gamma'_n \longrightarrow ct[\gamma'_i/\gamma_i]_{i \in [1..n]}}$$

An operator signature with quantified type variables can be replaced by a signature without universal quantification by replacing the quantified variables by fresh variables that are free in the typing environment.

$$\text{[LRM-331]}\ (\text{Dep Instance})\quad \frac{H; H_{last}; W; C \vdash f : ctf_1 \times \cdots \times ctf_n \longrightarrow ct \quad H; H_{last}; W; C \vdash e : ctf'_1 \times \cdots \times ctf'_n}{H; H_{last}; W; (C \cup \{ctf_i \supseteq ctf'_i \ / \ i \in [1..n]\}) \vdash f(e) : ct}$$

The causality type of an operator instantiation is the causality type of the outputs if the inputs satisfy the constraint of being bigger than the type of the argument i.e. if the inputs of this instance depend on the flows represented by e.

Pas de cycle instantané ($x_n = f(x_n)$)
Corollaire: code ordonnançable statiquement

Inspiré des travaux de Pascal Cuoq et Marc Pouzet sur la causalité modulaire.

Analyse d'initialisation

$$\text{[LRM-431]}\ (\textsc{Init Arrow})\quad \frac{H; H_{Last} \vdash e_1 : df_1^1 \times \cdots \times df_n^1 \quad H; H_{Last} \vdash e_2 : df_1^2 \times \cdots \times df_1^2}{H; H_{Last} \vdash e_1 -> e_2 : df_1^1 \times \cdots \times df_n^1}$$

An *init* expression $(e_1 -> e_2)$ is well initialized if e_1 and e_2 are; the initialization type of the expression is the one of its first parameter.

$$\text{[LRM-432]}\ (\textsc{Init Pre})\quad \frac{H; H_{Last} \vdash e : \underbrace{\mathbf{0} \times \cdots \times \mathbf{0}}_{n}}{H; H_{Last} \vdash \mathbf{pre}\ e : \underbrace{\mathbf{1} \times \cdots \times \mathbf{1}}_{n}}$$

Les valeurs de sortie sont toujours définies
Corollaire: déterminisme.

Jean-Louis Colaço and Marc Pouzet.
Type-based Initialization Analysis of a Synchronous Data-flow Language.
International Journal on Software Tools for Technology Transfer (STTT), Vol.6, Num.3, November 2004.

Implémentation du compilateur qualifié

- développé en OCaml
- développements spécifiques pour la qualification: outil de couverture de code qualifié, GC simplifié (Stop&Copy).

Pagano (B.), Andrieu (O.), Canou (B.), Chailloux (E.), Colaço (J.-L.), Moniot (T.) and Wang (P.).
Certified development tools implementation in objective caml.
In *International Symposium on Practical Aspects of Declarative Languages PADL 08.* Lecture Notes in Computer Science. Springer-Verlag, January 2008.

Pagano (B.), Andrieu (O.), Canou (B.), Chailloux (E.), Colaço (J.-L.), Moniot (T.), Wang (P.) and Manoury (P.).
Experience Report: Using Objective Caml to Develop Safety-Critical Embedded Tools in a Certification Framework
In *International Conference on Functional Programming* Proceeding of the 14th ACM SIGPLAN international conference on Functional programming, ICFP 2009, Edinburgh, Scotland, UK, August 31 - September 2, 2009

Vérification formelle de programmes SCADE

Une application SCADE a une mémoire finie
$\Rightarrow$ vérification de propriétés de sûreté par *model checking* possible.

Vérification formelle de programmes SCADE

Une application SCADE a une mémoire finie
$\Rightarrow$ vérification de propriétés de sûreté par *model checking* possible.

Sureté: *une situation redoutée ne se produit jamais*
e.g. "le train roule toujours avec les portes fermées"

Vérification formelle de programmes SCADE

Une application SCADE a une mémoire finie
$\Rightarrow$ vérification de propriétés de sûreté par *model checking* possible.

Sureté: *une situation redoutée ne se produit jamais*
e.g. "le train roule toujours avec les portes fermées"

Vivacité: *une situation espérée finira par arriver*
e.g. "le train partira un jour"

Vérification formelle de programmes SCADE

Une application SCADE a une mémoire finie
$\Rightarrow$ vérification de propriétés de sûreté par *model checking* possible.

Sureté: *une situation redoutée ne se produit jamais*
e.g. "le train roule toujours avec les portes fermées"

Vivacité: *une situation espérée finira par arriver*
e.g. "le train partira un jour"

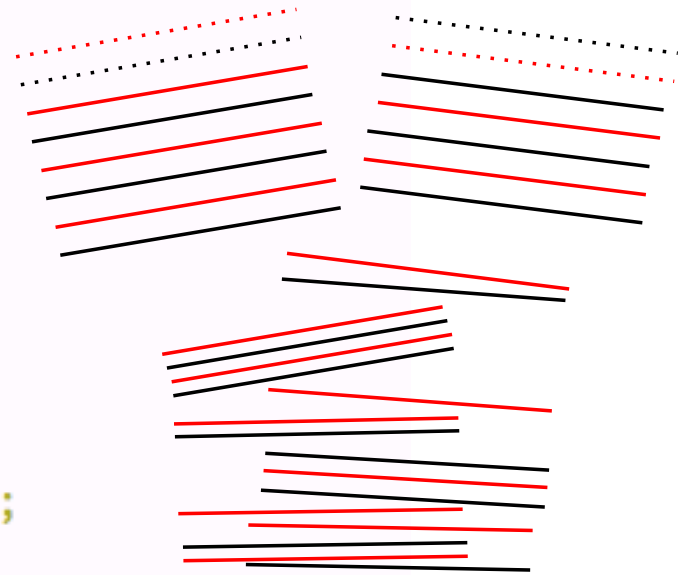
Une propriété de sûreté s'exprime en SCADE par un flot Booléen dont on cherche à montrer qu'il est vrai à chaque instant (*toujours* ...).

Exemple: The Gilbreath trick

```
node Gilbreath_stream (clock c:bool) returns (o, property: bool)
var
  s1 : bool when c;
  s2 : bool when not c;
  half : bool;
let
  s1 = (false when c) -> not (pre s1);
  s2 = (true  when not c) -> not (pre s2);
  o = merge (c; s1; s2);

  half = false -> (not pre half);

  property = true -> not (half and (o = pre o));
tel
```



G. Huet.

The Gilbreath trick: A case study in axiomatisation and proof development in the Coq Proof Assistant.
In *Proceedings, Second Workshop on Logical Frameworks, Edinburgh, May 1991.*

ANSYS ESTEREL Technologies

Perspectives

- Intégrer le temps continu (hybride);
- Poursuivre les efforts de formalisation de la compilation SCADE;
- Prouver la préservation de la sémantique (cf. projet CompCert, X. Leroy);
- Travailler sur l'Interprétation Abstraite au niveau SCADE.
- ...