Le langage synchrone Lustre

Alain GIRAULT

Inria Rhône-Alpes, projet Bip

Plan de l'exposé

1. Systèmes de contrôle/commande
2. Abstraction synchrone
3. Aspects fondamentaux de Lustre
4. Sémantique
5. Calcul d'horloges
6. Compilation
7. Autres aspects

Programmation des systèmes de contrôle/commande

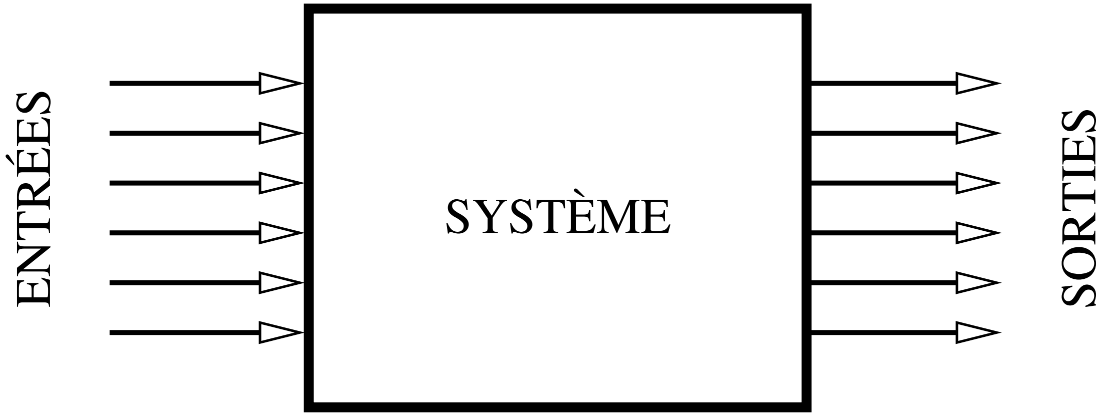
Commande = lois qui régissent l'évolution dynamique du système

- commande des actionneurs en fonctions des capteurs
- en temps continu (échantillonné)

Contrôle = lois qui régissent le comportement du système

- choix entre plusieurs lois de commande à appliquer en fonction du robot et de son environnement
- réalisation de la mission désirée et surveillance de son bon déroulement
- traitement des erreurs, des cas exceptionnels...
- en temps discret

Exemples



Contrôler le système = surveiller et agir sur son comportement

Exemples :

- véhicule automatique qui doit éviter les piétons
- robot manufacturier qui doit assembler une voiture
- robot sous-marin qui doit inspecter une installation

Contrôler le système = assurer le correct enchaînement des lois de commande

Caractéristiques de ces contrôleurs

Parallélisme : car le programme doit contrôler plusieurs automatismes en même temps

Déterminisme : car c'est plus facile de débuguer si le programme réagit toujours de la même manière aux mêmes séquences d'entrées

Temps-réel : car le système ne peut pas attendre

Sûreté de fonctionnement : car le système est critique

Approches classiques

- Systèmes câblés : hardware
- Assembleur : pour des raisons d'efficacité
- Langage classique + OS temps-réel : VxWorks, PSOS, OS9, QNX...
- Langage parallèle : ADA, OCCAM, JAVA...

Dilemme : langage de haut niveau ou langage de bas niveau ?

Critères : être parallèle, déterministe, temps-réel et sûr
➥ éviter les bugs et aller vite

Attention : les bugs temporels sont encore plus dur à trouver

L'abstraction synchrone

But : faciliter les raisonnements temporels

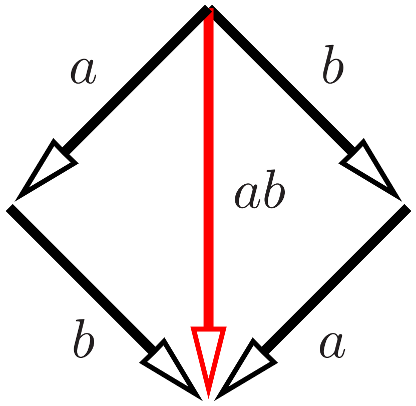
- Principe de simultanéité au lieu de l'entrelacement
  ➥ L'indéterminisme dû à l'entrelacement disparaît

- Toutes les activités parallèles partagent la même échelle de temps qui est discrète
  ➥ Elles peuvent être datées sur cette échelle

Ces deux principes constituent l'abstraction synchrone

Simultanéité contre entrelacement

$a \| b$



L'entrelacement est proche de la mise en œuvre

- L'ordre d'exécution de a et b dépend du compilateur
- Cela oblige à se poser la question

La simultanéité est volontairement de plus haut niveau

- On ne se préoccupe pas de l'ordre d'exécution
- Le compilateur se débrouille pour que ça marche

L'échelle de temps

L'échelle discrète du temps logique est projetée sur le temps physique

Rien ne se passe entre deux instants

C'est comme si l'ordinateur était infiniment rapide

Il faut par contre vérifier les contraintes temporelles

valider l'abstraction synchrone	$\equiv$	majorer le temps de réaction du programme

Aspects fondamentaux de Lustre

Historiquement : Lucid et les réseaux de Kahn

Lustre est un langage synchrone, déclaratif, fonctionnel et flot de données

Les objets manipulés (variables, constantes ...) sont des flots : séquences infinies de valeurs

Chaque flot est défini par son équation qui spécifie ses propriétés

Rien ne peut être inféré de la façon dont un flot est utilisé

- Principe de substitution
- Principe de définition

Un programme Lustre a un comportement cyclique qui définit l'horloge de base

Les horloges

Tout flot booléen définit une nouvelle horloge

➥ La suite des instants où le flot vaut `true`

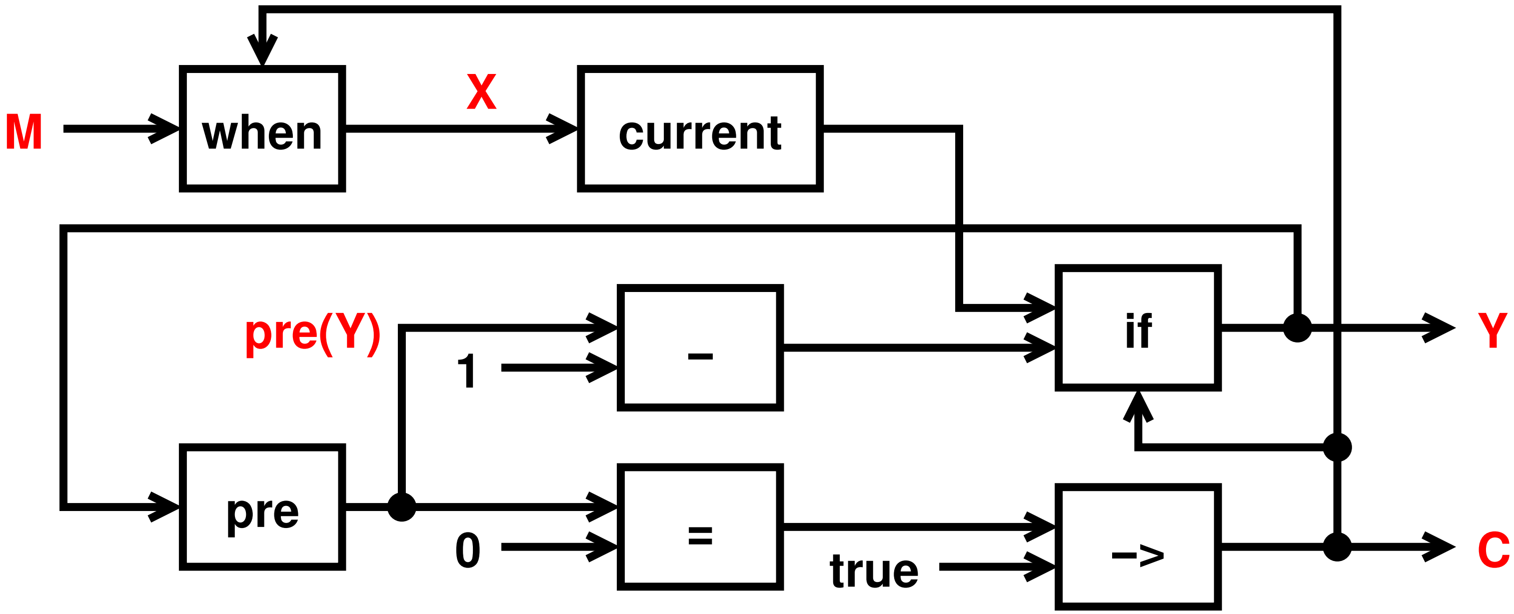
D'autres horloges peuvent être définie par sous-échantillonnage (opérateur `when`)

L'ensemble des horloges forme un arbre dont la racine est l'horloge de base

Par commodité, l'horloge de base est notée `true`

Syntaxe

```
node MUX (M:int) returns (C:bool; Y:int);
var (X:int) when C;
let
  Y = if C then current (X) else pre (Y) - 1;
  C = true -> (pre (Y) = 0);
  X = M when C;
tel;
```



Séquences sur V : le domaine est $V^\infty = V^* \cup V^\omega$

Les constantes :

TRUE	true	true	true	true	true	true	true	true	...
0	0	0	0	0	0	0	0	0	...

Extension point à point des opérateurs usuels :

X	0	3	5	2	12	9	27	10	...
Y	7	8	35	0	69	1	6	54	...
X+Y	7	11	40	2	81	10	33	64	...

Opérateurs – suite

Valeur initiale :

`X`	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	...
`Y`	y_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	...
`X -> Y`	x_0	y_1	y_2	y_3	y_4	y_5	y_6	y_7	...

Retard :

`X`	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	...
`pre(X)`	*nil*	x_0	x_1	x_2	x_3	x_4	x_5	x_6	...

Opérateurs – suite

Sous-échantillonnage :

`X`	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	...
`C`	`false`	`true`	`false`	`true`	`false`	`false`	`true`	`true`	...
`X when C`		x_1		x_3			x_6	x_7	...

Sur-échantillonnage :

`C`	`false`	`true`	`false`	`true`	`false`	`false`	`true`	`true`	...
`Y`		y_1		y_2			y_3	y_4	...
`current Y`	*nil*	y_1	y_1	y_2	y_2	y_2	y_3	y_4	...

Exemple d'exécution du nœud MUX

```
node MUX (M:int) returns (C:bool; Y:int);
var (X:int) when C;
let
  Y = if C then current (X) else pre (Y) - 1;
  C = true -> (pre (Y) = 0);
  X = M when C;
tel;
```

M	3	7	0	8	6	7	1	...
C	*true*	*false*	*false*	*false*	*true*	*false*	*false*	...
X	3				6			...
Y	3	2	1	0	6	5	4	...

Conventions typographiques : X est un flot, x est une valeur et *abs* est l'absence de valeur

Les constantes :

0 = *0*.0

Extension point à point des opérateurs usuels :

abs.X + *abs*.Y = *abs*.X+Y

x.X + *y*.Y = *x+y*.X+Y

Valeur initiale :

abs.X -> *abs*.Y = *abs*.X -> Y

x.X -> *y*.Y = *x*.Y

Sémantique de Lustre – suite

Retard :

$\mathtt{pre}\ \mathit{abs}.\mathtt{X} = \mathit{abs}.\mathtt{pre\ X}$

$\mathtt{pre}\ x.\mathtt{X} = \mathit{nil}.x.\mathtt{X}$

Sous-échantillonnage :

$\mathit{abs}.\mathtt{X}\ \mathtt{when}\ \mathit{abs}.\mathtt{C} = \mathit{abs}.\mathtt{X\ when\ C}$

$x.\mathtt{X}\ \mathtt{when}\ \mathit{true}.\mathtt{C} = x.\mathtt{X\ when\ C}$

$x.\mathtt{X}\ \mathtt{when}\ \mathit{false}.\mathtt{C} = \mathit{abs}.\mathtt{X\ when\ C}$

Sur-échantillonnage :

$\mathtt{current}\ v\ \ \mathit{abs}.\mathtt{C}\ \ \mathit{abs}.\mathtt{X} = \mathit{abs}.\mathtt{current}\ v\ \mathtt{C\ X}$

$\mathtt{current}\ v\ \ \mathit{true}.\mathtt{C}\ \ x.\mathtt{X} = x.\mathtt{current}\ x\ \mathtt{C\ X}$

$\mathtt{current}\ v\ \ \mathit{false}.\mathtt{C}\ \ x.\mathtt{X} = v.\mathtt{current}\ v\ \mathtt{C\ X}$

Calcul d'horloges

L'exécution d'un programme Lustre est une évaluation paresseuse

Il faut enrichir les flots de sortie au fur et à mesure que des valeurs sont ajoutées en entrée

Cela doit se faire en temps de réaction et en mémoire bornée

Certaines expressions doivent donc être interdites :

```
X + (X when C)
```

C'est le rôle du calcul d'horloges

Calcul d'horloges – suite

But : attribuer statiquement à chaque flot une horloge unique

- un flot qui n'est jamais échantillonné a pour horloge `true`
- un flot échantillonné a pour horloge le second argument du `when`
- les arguments d'un opérateur doivent avoir la même hrologe

Il n'est pas nécessaire de fournir au `current` l'horloge de sur-échantillonnage car le compilateur la calcule lui-même

Problèmes des définitions cycliques

Le principe de définition interdit à une variable de dépendre instantannément d'elle même

➥ `X = X + 1` n'a pas de sens

Le compilateur interdit cela grâce à une simple analyse statique

Identification des variables d'état

➥ horloges, conditionnelles, voire toutes les variables booléennes

Le compilateur simule le comportement de ces variables d'état

➥ Arbre infini des comportements

➥ Replié en un graphe fini

➥ C'est un automate d'états fini couplé à une mémoire bornée contenant les autres variables

On peut le représenter explicitement (automate)
ou implicitement (circuit)

Post-processing

Comme pour les autres langages synchrones, on peut appliquer de nombreux outils :

- générateurs de code,
- outils de minimisation d'automates/circuits,
- interfaces avec des outils de preuve formelle,
- outils de visualisation d'automates/circuits,
- générateurs d'interface,
- répartiteurs de code,
- outils de synthèse de séquences de tests,
- outils de synthèse de contrôleurs,
- ...

Développements industriels

Merlin-Gerin : Saga

Aérospatiale : SAO puis SAO+

Verilog puis Telelogic puis Esterel Tech : Scade

Certifié conforme à la norme DO178B de l'aviation civile

Applications :

- CO3N4 : contrôleur des centrales nucléaires (réacteurs 900 MW)
- A340/600 et futur A380 : commandes de vol
- Métro de Hong-Kong : logiciel de signalisation
- Eurocopter EC135 et EC155 : pilote automatique

Pour en savoir plus

Compilateur académique :
`http://www-verimag.imag.fr/~raymond/tools/lv4-distrib.html`

Compilateur industriel : `http://www.esterel-technologies.com/v2/scadeSuiteForSafetyCriticalSoftwareDevelopment`

Portail synchrone : `http://www.synalp.org`