Langages synchrones

Jérôme Hugues (ex. ENST)
Emmanuelle Encrenaz-Tiphène

(emmanuelle.encrenaz@lip6.fr)

M2 SAR/SESI/STL LS

V3.0, septembre 2009

2. Langage Lustre

a) **Présentation du langage**

Note: les exemples du cours sont disponibles sur le site du module

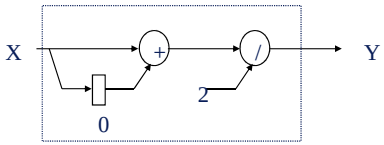
M2 SAR/SESI/STL LS

Langages de flots de données

- Les systèmes réactifs sont apparus à l'origine en automatique, en traitement du signal et conception de circuits
  - Formalisme équationnel : équations différentielles, équations aux différences finies
  - Formalisme diagramatique : schémas blocs, schémas analogiques
  - Réseau de Khan, Lucid : langages flots de données asynchrones
  - Matlab, simulink, VHDL-AMS
- Modèle data-flow
  - *flots* : suite infinie de valeurs, chaque flot est caractérisé par son équation (définition, propriété)
  - Schéma opératif pré-établi. Chaque nœud opérateur est activé par l'apparition de nouvelles entrées

Langages de flots de données (2)

- Caractéristiques
  - Parallélisme à grain fin (niveau opérateur)
  - Modèle fonctionnel :
    - *Calcul vu comme une composition de fonctions*
    - *Pas de notion de mémoire, d'affectation, d'effet de bord*
  - Représentation graphique du réseau d'opérateurs
    - *Décomposition hiérarchique*
- Exemple : filtre de convolution (traitement du signal)
  - $Y_1 = X_1/2$, $Y_n=(X_n+X_{n-1})/2$



Lustre

- **Lustre :**
  - **Langage flot de données :**
    - *Parallélisme grain fin, hiérarchie*
  - **Fonctionnel, déclaratif,**
    - *Composition et substitution*
  - **Synchrone :**
    - *Simultanéité des réactions, infiniment rapides, déterministe*
- **Comportement cyclique → *horloge de base***
- **Génération automatique de code impératif (simulation / vérification)**

Exemple

- **Syntaxe Lustre du filtre de convolution**

```
node convolution (X:real) returns (Y:real);
let Y = (X + (0.0 -> pre X)) / 2.0;
tel;
```

- **Autre écriture**

```
node convolution (X:real) returns (Y:real);
var pY: real;
let  Y = (X + pY) / 2.0;
     pY = 0.0 -> pre X;
tel;
```

- **Un programme Lustre = un ensemble d'équations de flots,**
  - **ordre sans importance**
  - **synchronisme de l'automatique (index = temps)**

Flots et horloges

- **Flot : Séquence infinie de valeurs typées, associé à une horloge**
  - **Flot sur V : V^ω**
  - **Ex : flot constant `C = 1,1,1,1, …`**
  - **Ex : `F = 1,2,1,1,3,78, -256, 170, …`**
- **Horloge : Flot booléen = `{true,false}`$^\omega$**
  - **Horloge H : la suite des instants où H vaut true**
  - **Horloge de base (tick) : `{true}`$^\omega$**
  - F = $(f_i)_{i \in IN}$: fi représente la valeur de F au i-ème instant de son horloge
  - **Ex : `H  = true, false, false, true, false, true, true, true,`**
- **Définition de nouvelles horloges par sous-échantillonnage**
  - *Arbre d'horloge de racine l'horloge de base*
  - *Permet d'indiquer l'absence de valeur dans un flot*
  - *Ex : `F when H = 1,(abs),(abs),1,(abs),78, -256, 170, …`*

Structure d'un programme Lustre

- **Une suite de déclarations de nœuds :**

```
node ID_NODE ([ID_IN : ID_TY_IN]*)
             returns ([ID_OUT : ID_TY_OUT]*);

    var [(ID_VAR : ID_TY_VAR) [when ID_HORLOGE]]*;

    Let
      ID_VAR1 = expression_1;
      ID_VAR2 = expression_2;
       …
      assert (expression_p);
      assert (expression_q);

    tel;
```

Déclaration interface du nœud (evt. sous-échantillonnées)

Déclaration variables internes (evt. sous-échantillonnées)

Équation : définition unique et non ambiguë d'une variable interne ou de sortie

Assertion : expression booléenne présupposée vraie à tout instant

- **Principes de *déclaration* et de *substitution***
- **Types de base :**
  - **`int, bool, real,`**
  - **constructeur de tuple, extensions : tableaux**
  - **types importés du langage hôte**

Expressions sur les flots

- **Expressions**
  - **Définition de flots et d'horloges**
  - **Baties sur : flots constants, variables de flots, opérateurs, instanciation de nœuds**
- **Opérateurs combinatoires**
  - **Flots constants**
  - **Opérations n-aires, éventuellement importées du langage hôte**
  - **Alternative**
- **Opérateurs temporels**
  - **Prédécesseur**
  - **Valeur initiale**
  - **Sous- et sur- échantillonnage**
- **Instanciation de nœuds précédemment déclarés (appels de fonctions)**

Opérateurs combinatoires

- **Flot**
  - Séquence de valeurs sur V : F = $(f_i)_{i \in IN}$ et $\forall i \in IN, f_i \in V$
  - Flot constant K : K = $(k_i)_{i \in IN}$ et $\forall i,j \in IN, k_i, k_j \in V$ et $k_i = k_j$,
- **Opérations n-aires (importées) (e.g. +, *, not, and, …)**
  - Application ponctuelle : opérandes et résultat sur la même horloge
  - Soient op : $V^n \rightarrow V$
    et n flots $F_1, \ldots F_n$ avec $F_i = (f_{il})_{l \in IN}$
    on définit $op(F_1, \ldots F_n) = (f_{1l}$ op $f_{2l} \ldots$ op $f_{nl})_{l \in IN}$
- **Alternative `R = if cond then F1 else F2`**
  - Application ponctuelle : `cond`, F_1 , F_2 et résultat sur la même horloge,
  - `cond` flot booléen, F_1 et F_2 flots de même type
  - Soient cond = $(c_i)_{i \in IN}$, $F_1 = (f_{1i})_{i \in IN}$, $F_2 = (f_{2i})_{i \in IN}$,
    on définit R = $(r_i)_{i \in IN}$ tq $r_i = f_{1i}$ si c_i = **`true`**, $r_i = f_{2i}$ si c_i = **`false`**

Opérateurs combinatoires

Ici, tous les flots sont sur l'horloge de base

flot	valeur au 1er tick	valeur au 2e tick	valeur au 3e tick	valeur au 4e tick
1	1	1	1	1
C	true	true	false	true
X	x_1	x_2	x_3	x_4
Y	y_1	y_2	y_3	y_4
X op Y	x_1 op y_1	x_2 op y_2	x_3 op y_3	x_4 op y_4
If C then X else Y	x_1	x_2	y_3	x_4

Liste des opérateurs combinatoires

- **Opérateurs booléens**
  - **`and , or , xor , not , #`**
  - **T = #(X,Y,…) défini tel que**
    X,Y, … T : flots booléens
    T = $(t_i)_{i \in IN}$ / t_i = **`true`** ssi au plus 1 parmi $x_i, y_i, \ldots$ vaut **`true`**
- **Alternative**
  - **`if … then … else`**
- **Opérateurs arithmétiques (entiers et réels)**
  - **`+ , - , * , / , div , mod`**
- **Comparaison**
  - **`= , <> , < , <= , > , >=`**
- **Conversion de type**
  - **`int , real`**

Opérateurs temporels (1)

- **Délais (pre)**
  - Soit F = $(f_i)_{i \in IN}$, F' = pre(F) de même horloge que F et défini tel que :
  - F' = $(f'_i)_{i \in IN}$ et f'_1 = nil,
    $f'_i = f_{i-1} \forall i > 1$

- **initialisation (->)**
  - Soit $F_1 = (f_{1i})_{i \in IN}$, $F_2 = (f_{2i})_{i \in IN}$, F' = F_1 -> F_2 est défini tel que :
  - F_1, F_2 et F sont de même horloge et de même type,
  - F' = $(f'_i)_{i \in IN}$ et $f'_{11} = f_{11}$ et $\forall i > 1$ $f'_{1i} = f_{2i}$

Illustration

Ici, tous les flots sont sur l'horloge de base

flot	valeur au 1er tick	valeur au 2e tick	valeur au 3e tick	valeur au 4e tick
x	x_1	x_2	x_3	x_4
y	y_1	y_2	y_3	y_4
pre x				
x -> y				
x -> pre y				

Exercice (1)

- **Ecrire le nœud Lustre permettant de compter le nombre d'occurrences de "i" entre deux occurrences successives de "toc". (On ne se préoccupera pas des i survenant avant la première occurrence de toc.)**

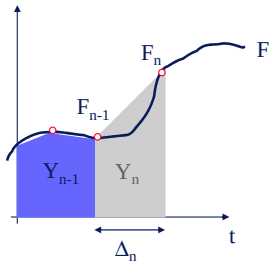
i	F	F	F	T	F	T	T	F
toc	F	T	F	F	F	F	T	F
o	0	0	0	1	1	2	0	0

Exercice (2)

- **Réalisation d'un intégrateur (intégration trapézoïdale)**

Soit F(t) une fonction à intégrer.
L'intégrale Y(t) est définie par : Y(t) = Y(t-1) + (F(t-1) + F(t)) * Δ(t) / 2

Sa discrétisation est donnée par : $Y_n = Y_{n-1} + (F_{n-1} + F_n)*DELTA_n/2$



Exercice (3)

- Représentez le nœud Lustre construisant les termes de la suite de Fibonacci définie telle que : $x_0=0$, $x_1=1$, $x_{n+2}=x_{n+1}+x_n$

- Solution avec gestion explicite des valeurs initiales

Fibonacci suite

- Le nœud fibo calcule les termes de la suite de Fibonacci à partir du rang 1.
- Représentez les valeurs des flots suivants

```
node fibo (tick : bool) returns (x:int);
let
  x = 1 -> pre (x + (0 -> pre x))
tel;
```

x											
0 -> pre x											
x + (0 -> pre x)											
pre(x + (0 -> pre x))											

Exercice (4)

- Que retourne

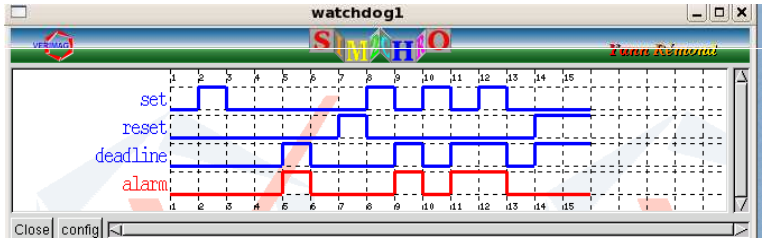
```
node Test (x:int) returns (z,t:int)
let
    z =  0 -> 1 -> 2;
    t =  0 -> pre(1 -> 2);
tel

?
```


Chien de garde

- Chien de garde ou watchdog
  - Permet de gérer les échéances, e.g. détecter automatiquement une anomalie du logiciel et réinitialiser le processeur
  - **3 arguments:** set, reset, deadline
  - **Émet** alarm **lorsque**
    - ***watchdog*** *est activé (un set est survenu depuis le dernier reset),*
    - *et* ***deadline*** *est vrai*

Chronogramme



Chien de garde

- Nœud du watchdog

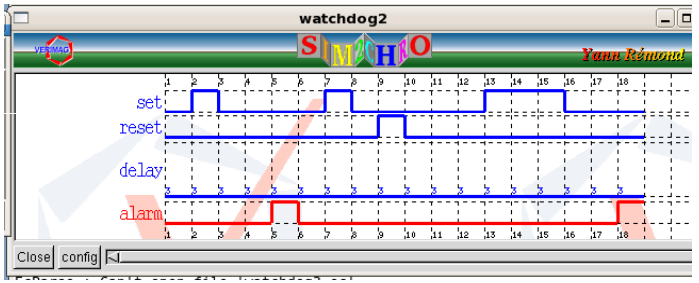
```
node WATCHDOG1 (set, reset,deadline:bool) returns (alarm:bool);
var watchdog_is_on:bool;
let
   alarm =
   watchdog_is_on =


   assert not (set and reset); -- set et reset ne sont jamais
                               -- vrais simultanément
tel;
```


Chien de garde avec timer (1)

- Le chien de garde est périodiquement activé (signal **set**) et lors de l'activation, un délai de garde (**delay**) lui est affecté. Une fois activé, le chien de garde décompte le temps à partir du délai qui lui a été affecté. Si le décompte atteint la valeur nulle (avant que le chien de garde ne soit réactivé), le chien de garde émet le signal **alarm**. Le chien de garde peut être désactivé à tout instant par le signal **reset**.
- Reçoit set, reset, delay
- Émet une alarme (alarm) lorsque set est resté à faux pendant un délai delay
- Le délai de garde est défini en nombre de ticks de l'horloge de base.

Chronogramme



Chien de garde avec timer (2)


```
node WATCHDOG_Timer (set,reset:bool; delay:int) returns (alarm:bool);
var remaining_delay:int; deadline:bool;
let
   alarm =
   deadline = EDGE (                        );
   remaining_delay =

tel;

node EDGE (b:bool) returns (edge:bool);
let
   edge =
tel;
```


Boucles de causalité

- **Rejeter les boucles de causalité, signes de verrouillage fatal (deadlock)**
  - `let x = x + 1; -- x` **dépend instantanément de lui-même**
- **Les flots définis récursivement doivent être calculables séquentiellement (en fonction des valeurs précédentes)**
- ***Condition syntaxique:*** **une variable récursive doit toujours être gardée par un délai**
- **Pas de résolution d'équation**

```
x = ( 2 * x - 1 ) / x;  -- programme rejeté
```

```
x = if c then y else z; -- programme rejeté
y = if c then t else x;
```

⇔

```
x = if c then
      if c then t else x;
      else z;
```


Opérateurs temporels (2)

Sous-échantillonage : `when`

Définir un flot plus lent que les entrées

- Soit F = $(f_i)_{i \in IN}$ un flot sur V et H = $(h_i)_{i \in IN}$ un flot booléen
  - F et H sont définis sur la même horloge,
  - X = F `when` H est défini tel que :
    - X = $(x_i)_{i \in IN}$ un flot sur V
    - $x_i = f_i$ ssi h_i = `true`
  - X est sur l'horloge H, elle-même sur une horloge plus rapide

true	true	true	true	true	true	true
B	false	true	false	true	false	false
X	x_0	x_1	x_2	x_3	x_4	x_5
Y = X when B		x_1		x_3		

Opérateurs temporels (3)

Sur-échantillonage : `current`

Mettre un flot sur l'horloge immédiatement plus rapide

B	false	true	false	true	false	false
X	x_0	x_1	x_2	x_3	x_4	x_5
Y = X when B		x_1		x_3		
Z = current Y	nil	x_1	x_1	x_3	x_3	x_3

- **L'horloge de `current(Y)` est l'horloge de l'horloge de** Y
  (donc ici l'horloge de B, i.e l'horloge de base)
- **A l'instant initial, `current(Y)` peut ne pas être initialisé:**
  - **Soit prendre une horloge de la forme `true -> clk`**
  - **Soit `current(((if clk then X else init) -> X) when clk)`**

Formalisation de `current`

- **`F = current X`**
  - X un flot d'horloge H, elle-même d'horloge Hb
    (h_i est défini $\Leftrightarrow$ hb_i = `true`)
  - v la dernière valeur de X lorsque hb_i = **`true`**
  - Définition locale d'un élément de F :
    - ***si*** hb_i = `true`, $f_i = x_i$ ***et*** $v := x_i$ ***// affectation de v! : parcours de X***
    - ***si*** hb_i = `false`, $f_i = v$ *// valeur courante de v : dépend de sa // dernière affectation*

Exemple


```
node somme(i:int) returns (s:int);
let
   s = i -> pre s + i
tel;
```

i	1	1	1	1	1	1
cond	true	false	true	true	false	true
somme i	1	2	3	4	5	6
somme (i when cond)	1		2	3		4
(somme i) when cond	1		3	4		6

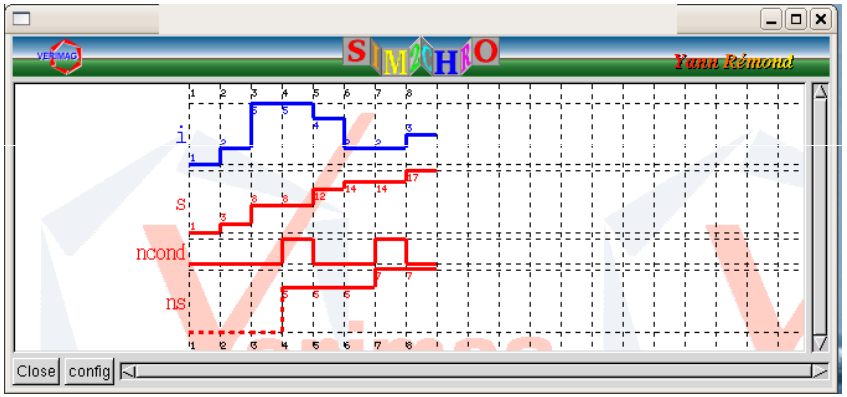
- **Remarques**
  - `f(x when c)` ≠ `(f x) when c`
  - `current (x when c)` ≠ `x`

Exemple (2)


```
node stables(i:int)  -- horloge de base (true)
  returns (s:int; ncond:bool;
           (ns:int) when ncond);  -- déclaration d'horloge
var cond:bool;
      (l:int) when cond;  -- déclaration d'horloge
let
    cond = true -> i <> pre i;
    ncond = not cond;
    l = somme (i when cond);
    s = current (l);
    ns = somme (i when ncond);
tel;
```

- **Les horloges doivent être déclarées et visibles dans l'interface du nœud**

Chronogramme



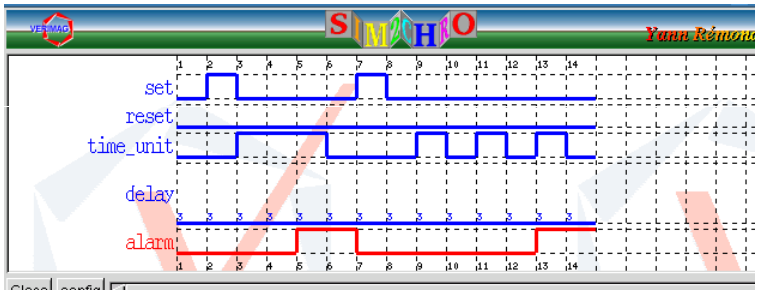
Le chien de garde (3)

- Le chien de garde reçoit un signal supplémentaire donnant son horloge. Le décompte est effectué à chaque occurrence de l'événement time_unit)

```
node WATCHDOG3(set, reset, time_unit:bool;delay:int)
   returns (alarm:bool);
var clk:bool;
let alarm = current (WATCHDOG2((set, reset, delay) when clk));
      clk = true -> set or reset or time_unit;
tel;
```

- Remarque: une composition `current (f(x when c))` est une « condition d'activation »

Chronogramme



Contraintes d'horloge (1)


```
let half = true -> not pre half;
      y = x and (x when half);
tel
```

- Correspond à la suite $\forall n, Y_n = X_n \& X_{2n}$
- Ce programme doit être rejeté : Pas d'implantation en mémoire bornée

Contraintes d'horloge (2)

- Quelques contraintes à connaître :
  - Les constantes sont sur l'horloge de base du nœud
  - Par défaut, les variables sont sur l'horloge de base du nœud,
    - clock (e1 op e2) = clock (e1) = clock (e2)
    - clock (e when c) = c
    - clock (current (e)) = clock (clock (e))
- Choix d'implantation :
  - Les horloges sont déclarées et vérifiées
  - Pas d'inférence (plus compliqué)
  - Deux horloges sont égales si elles peuvent être rendues syntaxiquement égales après substitution
- Problème étudié dans la suite du cours

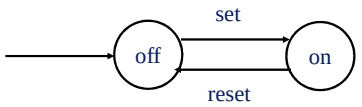
Exercice *[repris de P. Raymond]*

- Complétez le tableau suivant

X	4	1	-3	0	2	7	8	13
Y	true	false	true	true	true	false	false	true
C	true	true	false	true	true	false	true	true
Z = X when C								
H = Y when C								
T = Z when H								
current (T)								

Codage d'automate

- Problème: comment coder un automate ?



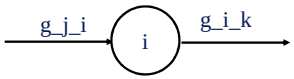

```
node TWO_STATES(set, reset:bool) returns (on:bool);
let
   on = false ->
           if set and not (pre on) then true
           else if reset and (pre on) then false
           else (pre on);
tel;
```

- Penser en terme d'invariants: quelle est l'expression définissant la valeur de 'on' à chaque instant
  - Contrainte: au plus un état actif à chaque instant

Codage systématique des automates

- Une variable booléenne par état :
  - e_i **est associée à l'état** i : e_i = TRUE ⇔ l'automate est dans l'état i
  - g_i_j : garde de la transition menant de l'état i vers l'état j
  - Expanser la fonction de transition pour chaque état

```
e_i = init_i ->
        if pre e_i
           and (    g_i_j
                 or g_i_k
                    … )
         then false
       else if pre e_j and g_j_i
                 or pre e_k and g_k_i
                 …..
            then true
         else pre e_i;
```



Note: récupérer le résultat d'un nœud

- Procéder par identification de flots

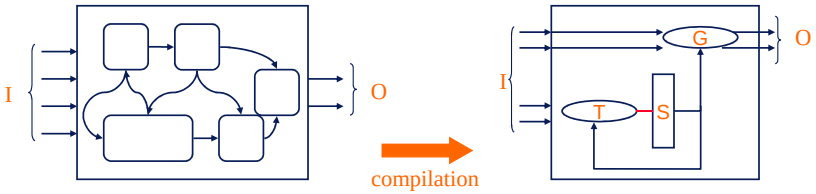
```
node position (tick : bool) returns (x,y : int);
let
   x = 0 -> pre x + 1;
   y = 0 -> pre y + 1;
Tel

node distance_carre (tick : bool) returns (d : int);
var x, y : int;
let
   (x, y) = position (tick);
   d = x * x + y * y;
tel
```

Constructeur de tuples

Compilation en code C exécutable

La description Lustre est transcrite en un programme exécutable (C)



Description réactive parallèle

Description séquentielle (machine de Mealy)

Structure du programme exécutable

- **Code séquentiel le plus simple : une boucle infinie**

```
var I, O, S;
proc P_step() … ; // traitement : évaluation de toutes les équations

S := S0;          // initialisation

foreach step do   // boucle infinie
     read(I);
     P_step();
     write(S);
end foreach
```

- **Le corps de la boucle est instantané**
- **`step` est actionné sur l'horloge de base du système**
- ***Détaillé dans la suite du cours***

Vérification

- **Cohérence du programme (analyse statique) : avant la compilation**
- **Vérification par introduction *d'observateurs***
  - **Un module concurrent observant l'évolution de certains flots**
  - **Non intrusif**
  - **Utilisation de clauses `assert` / signaux de sortie**
- **Analyse par simulation**
- **Analyse exhaustive de l'espace d'états du programme**
  - **Propriétés de sûreté :**
    - *Le système n'entre pas dans un ensemble d'états mauvais*
- ***Détaillé dans la suite du cours***

Exercice: contrôleur de souris

Émettre double **si on a reçu 2** click **en moins de 4** top**, sinon** single

```
node counter (res, event: bool) returns (count: int);
var x:int;
let   count =
      x =
tel;

node mouse (click, top:bool) returns (single, double:bool);
var counting, res:bool; count:int;
let counting =


      count =
      single =
      double =
      res =
tel;
```

Chronogramme