Le langage LUSTRE : Programmation data-flow synchrone pour les systèmes embarqués

N7 – 3SN

Cours 1 : Lustre basic

Frédéric Boniol

ONERA - 2, av. E. Belin - 31055 Toulouse

frederic.boniol@onera.fr

Préambule

Objectif de la présentation

présenter le langage data-flow synchrone LUSTRE pour la programmation des systèmes temps réel

⇒ Modèle de programmation « data-flow »

⇒ Sémantique « synchrone »

Plan

1. Cible : systèmes de contrôle commande
2. Synchrone versus asynchrone
3. Le langage Lustre de base

→ Lundi 21/11 (2h30)

4. Le langage Lustre avec les horloges → Mercredi 30/11 (2h30)
5. La vérification de programmes → Mardi 6/12 (1h15)
6. TP : robot Légo

7. Exam : vendredi 6 janvier (1h)

1. La cible : programmation des systèmes de contrôle-commande

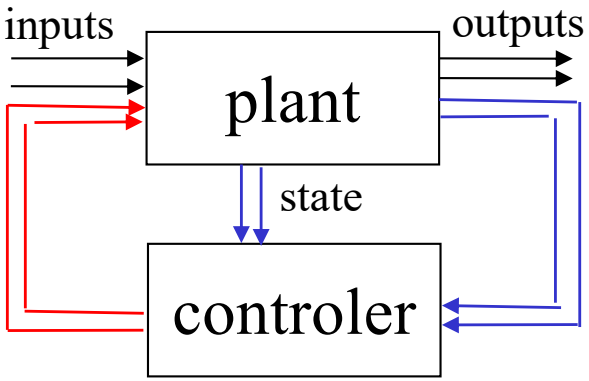
1. Contrôle-commande…

Command: Laws which govern the dynamical evolution of a system

- Command of actuators regarding the sensors
- In continuous time

Examples:

- Regulation of a liquid level between thresholds
- Command of flight surfaces



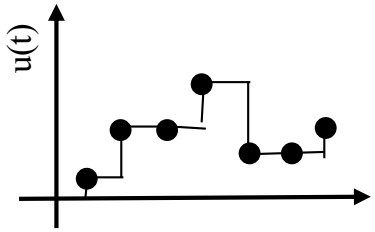
Equations of state

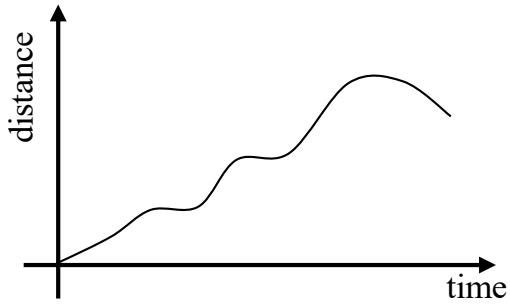
$$\begin{cases} dx = f(x,u) \\ y = h(x) \end{cases}$$

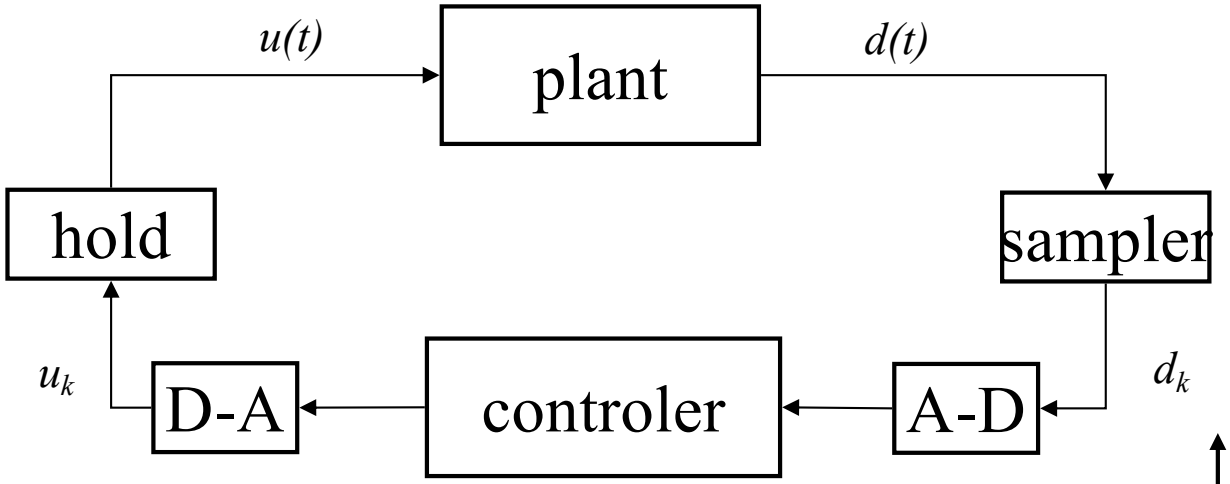
x internal state,
y output,
u input

Exemple : flight control system

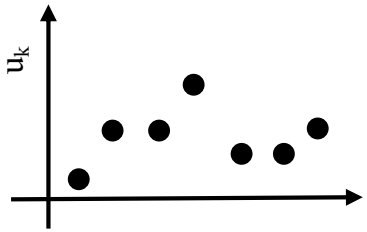
1. Contrôle-commande…



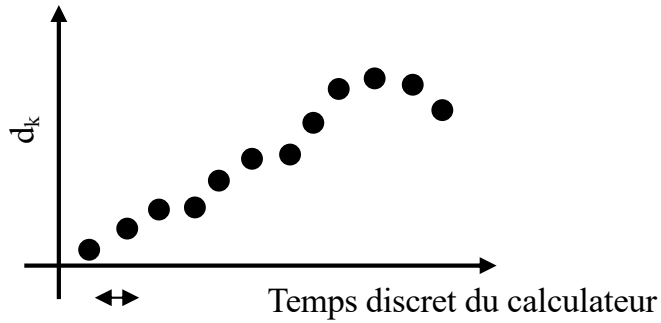




vitesse: $v = \frac{d(\text{distance})}{dt}$



$$v_k = \frac{d_k - d_{k-1}}{T}$$



Ordre élévateur

T période d'échantillonnage

=> Controler = système échantillonné dont le comportement est décrit par des équations

2. Principes synchrones et data-flow…

2.1 Rappel sur la concurrence asynchrone

Concurrence asynchrone =

- Entrelacement non-déterministe des threads
- Pourquoi : indépendance des vitesses d'exécution des processus les uns par rapport aux autres

  ⇒pas de temps global

  ⇒les actions ont une durée non déterministe

⇒ Conséquence : il est nécessaire de synchroniser les processus (mutex, sémaphore..)

Intérêt :

Correspond assez bien au mode de fonctionnement des machines informatiques.

Problème :

- non-déterministe
- complexité

2.1 Rappel sur la concurrence asynchrone

Exemple :

```
Signal X : integer ;

X <- 0;
[
   X <- 1;
   X <- 2;
||
   Y <- X+1;
]
```

Asynchronisme
=> chaque branche d'un programme parallèle se déroule à son propre rythme (indéterminé)

⇒ si sémantique asynchrone,
alors plusieurs entrelacements possibles
et donc plusieurs résultats possibles pour Y : 1, 2 ou 3

2.2 Principes synchrone et data-flow

Idée : simplifier la programmation

=> principe synchrone = abstraction du temps

- on suppose l'existence d'un temps global discret
- on suppose que les actions du controleur sont atomiques et s'exécutent dans un instant discret

=> deux actions exécutées dans le même instant discret seront considérées comme simultanées

=> principe data-flow

- l'exécution des actions est dirigée par les données

=> Intérêt :

- déterminisme
- simplification des programmes

⇒ Mais : le principe synchrone doit être confirmer par confrontation avec le monde concret

Remarque : Même démarche que les physiciens ou les chimistes quand il s'agit de modéliser un phénomène

=> démarche par simplification (i.e., cacher des détails inutiles)

=> Ici : abstraction du temps !

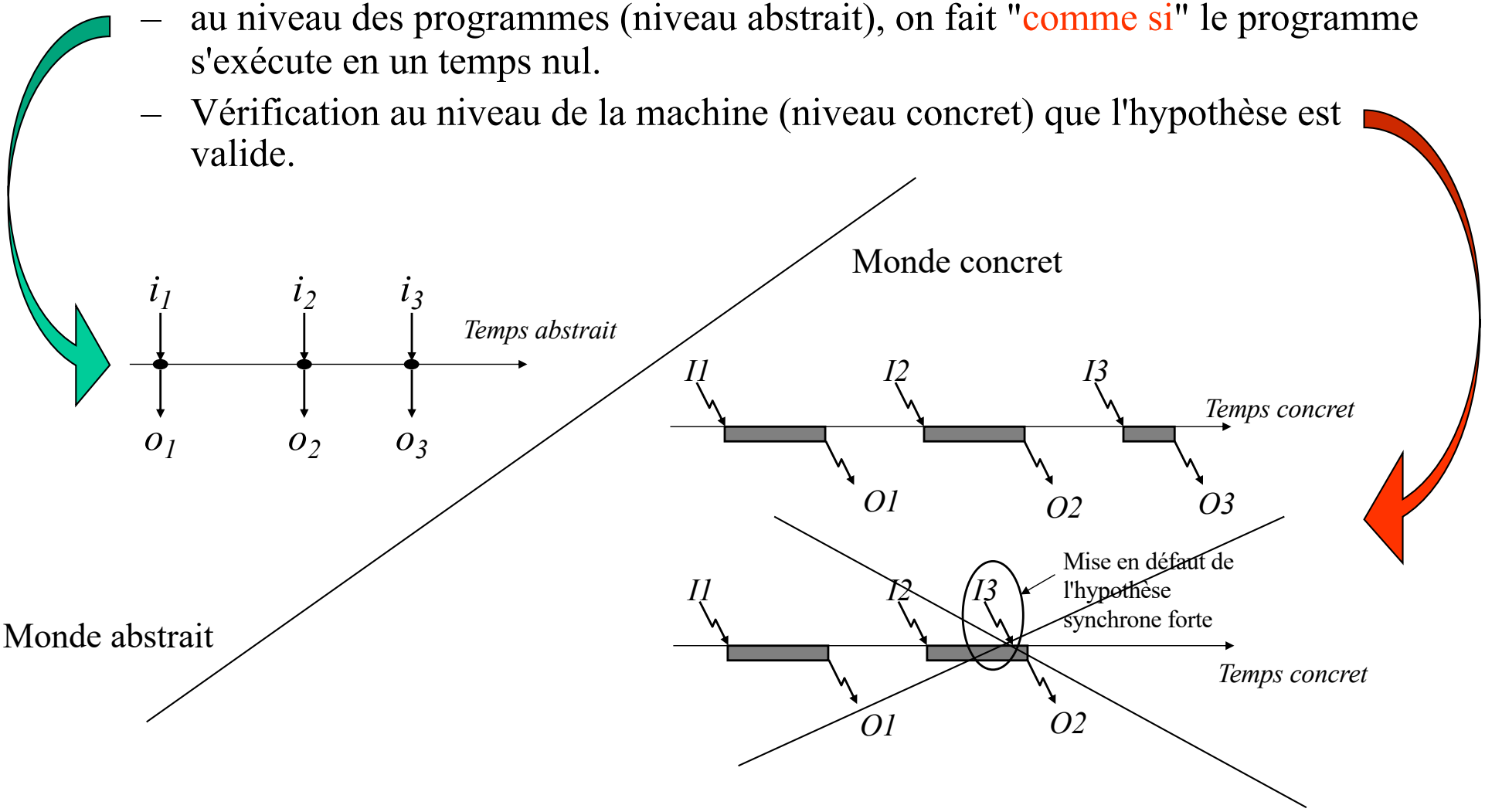
2.2 Principe synchrone

Principe synchrone : *tout s'exécute dans l'instant discret*

=> Abstraction des temps d'exécution et de communication réels (et donc du support matériel) :

– au niveau des programmes (niveau abstrait), on fait "comme si" le programme s'exécute en un temps nul.

– Vérification au niveau de la machine (niveau concret) que l'hypothèse est valide.

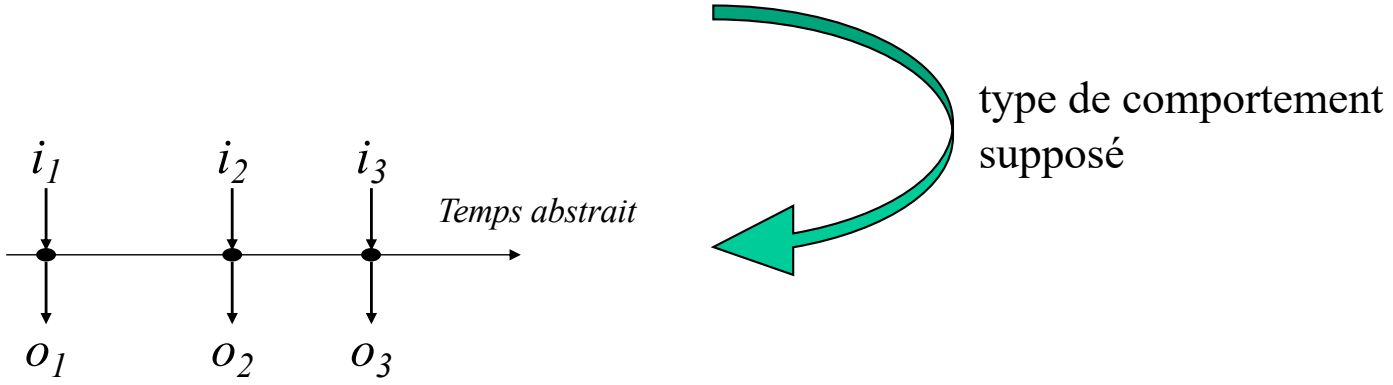


2.2 Principes synchrone

Démarche :

S'il est possible de considérer que les temps de réactions réels sont négligeables devant la dynamique de l'environnement

=> alors, idée : ne pas considérer ces temps d'exécution



=> application à la programmation des tâches d'un système temps réel pourvu que les temps d'exécution de celles-ci soient négligeables (les tâches ne doivent donc pas être préemptées)

Intérêt :

1 . Offrir une vision simplificatrice du comportement interne des tâches :

→ simplifie la programmation interne du système

2 . Rendre déterministe le comportement des tâches

2.2 L'idée du synchronisme fort

Exemple :

```
Signal X : integer combine with + ;

X <- 0;
[
   X <- 1;
   X <- 2;
||
   Y <- X+1;
]
```

Synchronisme
=> toutes les branches s'exécutent simultanément et instantanément

⇒ si sémantique synchrone,
alors un seul comportement (tout en même temps)
et donc un unique résultat pour Y = 4

exécution simultanée + loi de combinaison => assignation unique pour X => déterminisme pour Y

2.4. Les langages Synchrones

Deux langages :

- LUSTRE pour la programmation de processus opérant majoritairement sur des flots semi-continus

- ESTEREL pour la programmation de processus opérant majoritairement sur des événements

⇒Suite du cours : Lustre

- Le langage Lustre… simple
- Le langage Lustre… avec des horloges
- La vérification de programmes Lustre
- La sémantique formelle
- Résumé

3. Lustre de base

3.1. Le langage Lustre : généralités

Motivation :

Permettre la programmation « naturelle » et « sûre » de systèmes de contrôle commande.

Moyen

Techniques de programmation proches des descriptions traditionnelles utilisées par les ingénieurs de ces domaines :

=> blocs diagrammes et flots de données

=> systèmes échantillonnés

=> LUSTRE

langage de programmation formel défini en 1985 par P. Caspi et N. Halbwachs à Grenoble (Vérimag)

- Distribution commerciale : SCADE - Esterel Technologie
- Utilisations industrielles : Airbus, Dassault Aviation, Thales, Schneider Electric…

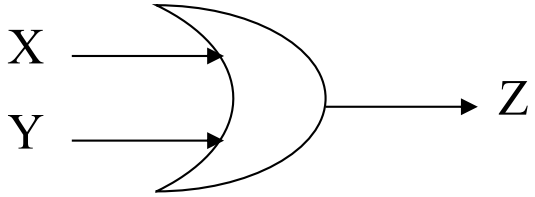
3.1. Le langage Lustre : généralités

- Caractéristiques générales du langage
  - Objets central : les « flots » :
    - portée locale, entrée, sortie
    - typé
    - Définis par des « équations »
  - Principe data-flow
    - Résolution d'une équation uniquement lorsque tous ses flots « d'entrée » sont présents et calculés
    - Effet = rendre présent et évalué le flot de sortie de l'équation
  - Principe synchrone
    - Portée temporelle des calculs = l'instant courant
    - Accès aux valeurs des instants précédents par mémorisation du passé

3.1. Le langage Lustre : exemple introductif

Exemple :

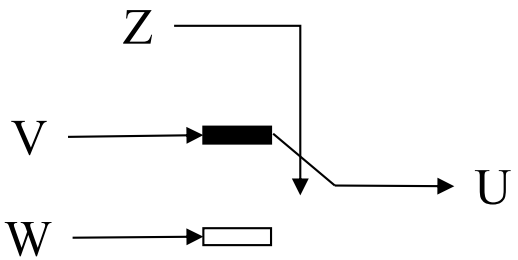
Une porte logique




```
Z = X or Y;
```

pour tout $n \geq 0$, $Z_n = X_n$ *or* Y_n

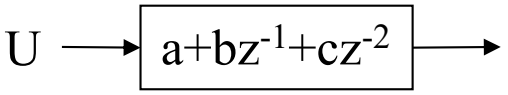
Un switch




```
U = if Z then V
    else W;
```

pour tout $n \geq 0$, U_n = *if* Z_n *then* V_n *else* W_n

Un filtre



pour tout $n \geq 2$, $O_n = aU_n + bU_{n-1} + cU_{n-2}$

```
O = a*U
    + b*pre(U)
    + c*pre(pre(U));
```

(attention : équation incorrectement initialisée)

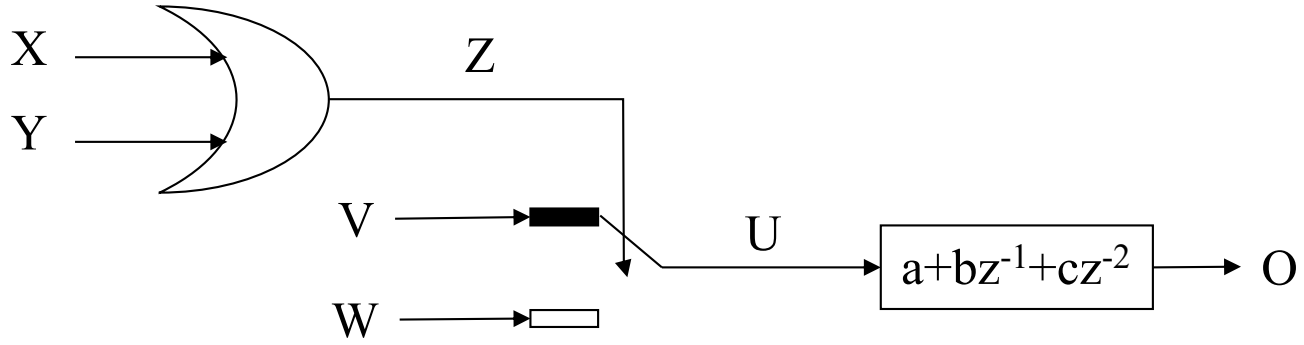
3.1. Le langage Lustre : généralités…

Généralisation :

- description d'un système en terme de suites de valeurs échantillonnées : les flots de données

=> un système = un réseau d'opérateurs opérant sur des flots de données

Exemple :




```
Z = X or Y;
U = if Z then V else W;
O = a * U + B * pre(U) + c * pre(pre(U));
```

(attention : programme incorrectement initialisé)

3.2. Le langage Lustre…

Généralisation :

- Notion de flot de données :

  X = suite de valeurs X_n pour n≥0 (flot infini de valeurs)

  X_n = valeur de X à l'instant *n* ($n^{ième}$ top d'horloge)

  Exemples :

  - `1` est le flot infini (`1,1,1,1,1,`…)
  - `true` = (`vrai`, `vrai`, `vrai`, `vrai`,…)

- Chaque flot interne ou de sortie est défini par une équation

  `O = X op Y`

  calculant O_n en fonction de X_n et Y_n (au même instant)

  => `op` opère à chaque instant sur les valeurs de l'instant courant (application point à point)

=> un programme LUSTRE =

- un ensemble d'équations
- qui modélise un processus rythmé par une horloge logique (pas forcément régulière) : à chaque top de cette horloge, le processus LUSTRE calcule des flots de sorties en fonction des valeurs des flots d'entrée à cet instant

3.2. Le langage Lustre…

Un programme LUSTRE = un ensemble de nœuds…

```
[déclaration de types et de fonctions externes]
node nom (déclaration des flots d'entrée)
returns (déclaration des flots de sortie)
[var déclaration des flots locaux]
let
   [assertions]
   système d'équations définissant une et une seule fois les flots
   locaux et de sortie en fonction d'eux mêmes et des flots d'entrée
tel.

[autres nœuds]
```

Déclaration de flots :

```
NomDuFlot : TypeDuFlot;
```

Flots constant :

```
const NomDuFlot : TypeDuFlot = valeur ;
```

Les types :

- les types de bases : `int`, `bool`, `real`
- les tableaux : `int^3`, `real^5^2`…

3.2. Le langage Lustre…

Les équations

– une équation définit un flot interne ou de sortie en fonction de flots internes, d'entrée ou de sortie

$$\begin{cases} X = Y + Z \\ Z = U \end{cases}$$

signifie

pour tout $n \geq 0$, $X_n = Y_n + Z_n$ *et* $Z_n = U_n$

=> principe de substitution : une équation définit une égalité mathématique, et non une affectation informatique => un flot peut être remplacé par sa définition dans toutes les équations du nœud

$$\begin{cases} X = Y + Z \\ Z = U \end{cases} \quad \text{équivalent à} \quad \begin{cases} X = Y + U \\ Z = U \end{cases}$$

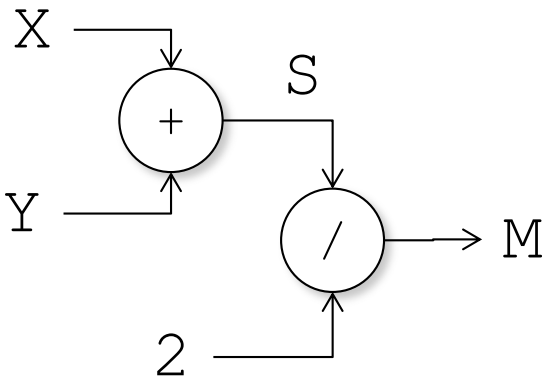
=> Principe data-flow : les équations n'ont pas d'ordre

$$\begin{cases} X = Y + Z \\ Z = U \end{cases} \quad \text{équivalent à} \quad \begin{cases} Z = U \\ X = Y + Z \end{cases}$$

3.2. Le langage Lustre…

- Exemple : calcul d'une moyenne de deux valeurs

```
node Moyenne (X, Y : int)
returns (M : int);
var S : int;
let
   M = S / 2 ;
   S = X + Y;
tel.
```



Une équation pour chaque sortie et chaque variable locale

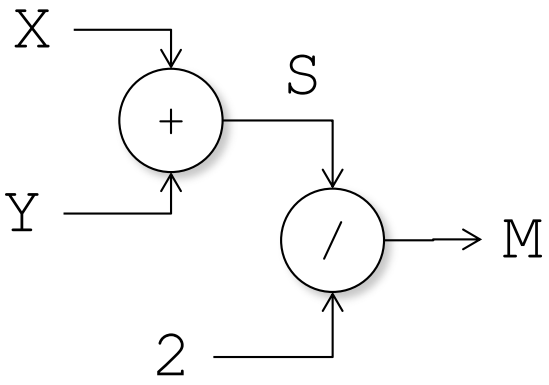
Interprétation du temps : pour tout $n \geq 0$,

- $S_n = X_n + Y_n$
- $M_n = S_n / 2$

3.2. Le langage Lustre…

- Exemple (suite) : équivalent à (par principe de substitution)

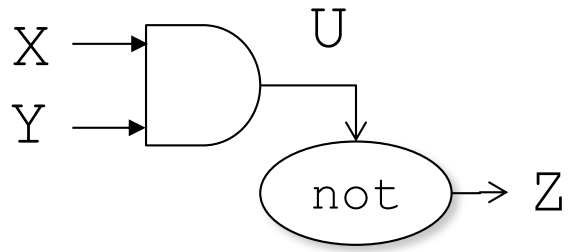
```
node Moyenne (X, Y : int)
returns (M : int);
let
   M = (X + Y) / 2 ;
tel.
```



3.2. Le langage Lustre…

- Exemple :

```
node Nand (X, Y : bool) returns (Z : bool)
var U : bool;
let
   U = X and Y;
   Z = not U;
tel.
```



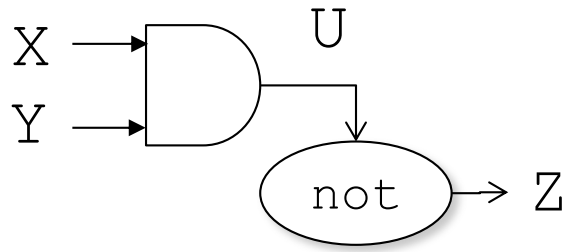
	tick	0	1	2	3	4	5	6	7
inputs	X	true	true	false	false	true	true	false	false
inputs	Z	false	true	false	false	true	false	false	true
local	U	false	true	false	false	true	false	false	false
output	Z	true	false	true	true	false	true	true	true

Équivalent à (par principe de substitution) =>

3.2. Le langage Lustre…

- Exemple :

```
node Nand (X, Y : bool) returns (Z : bool)
let
   Z = not (X and Y);
tel.
```



	tick	0	1	2	3	4	5	6	7
inputs	X	true	true	false	false	true	true	false	false
inputs	Z	false	true	false	false	true	false	false	true
output	Z	true	false	true	true	false	true	true	true

3.3. Les opérateurs

Opérateurs classiques

Les opérateurs arythmétiques :

Binaire : `+`, `-`, `*`, `div`, `mod`, `/`, `**`

Unaire : `-`

Les opérateurs logiques :

Binaire : `or`, `xor`, `and`, `=>`

Unaire : `not`

Les opérateurs de comparaison :

=, <>, <, >, <=, >=

Les opérateurs de contrôle :

`if . then . else`

Opérateurs temporels :

`pre` (précédent) : opérateur permettant de travailler sur le passé d'un flot

`->` (suivi de) : opérateur permettant d'initialiser un flot

`when` : opérateur de sous-échantillonnage

`current` : opérateur de sur-échantillonnage

3.3.1. pre et ->

L'opérateur `pre` (précédent)

Permet de mémoriser la valeur précédente d'un flot ou d'un ensemble de flots

Soit

X le flot $(X_0, X_1, \ldots, X_n, \ldots)$

alors

`pre(X)` est le flot $(\text{nil}, X_0, X_1, \ldots, X_n, \ldots)$

Par extension, l'équation

```
(Y, Y') = pre(X, X')
```

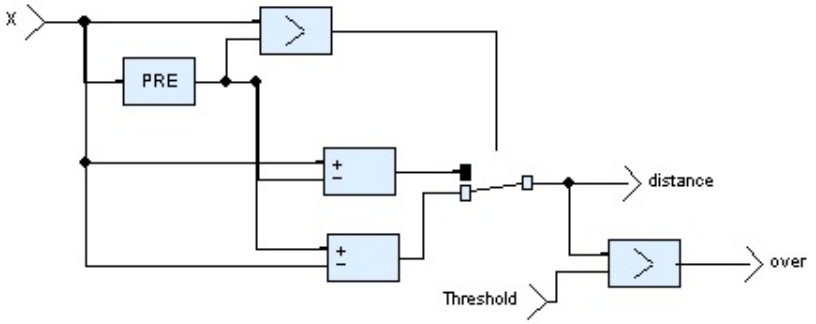
signifie

$Y_0 = \text{nil}, \quad Y'_0 = \text{nil}$

et pour tout $n \geq 1$, $Y_n = X_{n-1}$ et $Y'_n = X'_{n-1}$

Exemple : détection de dépassement de seuil

```
distance = if (X>pre(X))
               then X-pre(X)
               else pre(X)-X ;
over = (distance > Threshold) ;
```



3.3.1. pre et ->

L'opérateur -> (suivi de)

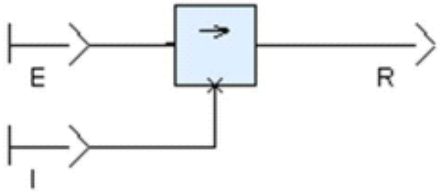
Permet d'initialiser un flot ou un ensemble de flots

Soit

X le flot $(X_0, X_1, \ldots, X_n, \ldots)$ et Y le flot $(Y_0, Y_1, \ldots, Y_n, \ldots)$

alors

`R = Y -> X` est le flot $(Y_0, X_1, \ldots, X_n, \ldots)$



Par extension, l'équation

```
(Z, Z') = (Y, Y') -> (X, X')
```

signifie

$Z_0 = Y_0$, $Z'_0 = Y'_0$ et pour tout $n \geq 1$, $Z_n = X_n$ et $Z'_n = X'_n$

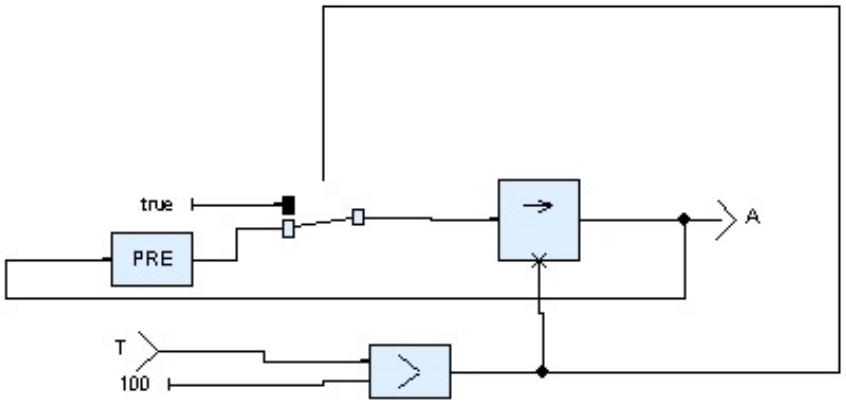
Exemple : surveillance d'une température

```
A = (T>100) ->
        if (T>100)
        then true
        else pre(A) ;
```

équivalent à :

$A_0 = (T_0 > 100)$

$A_n = \begin{cases} \text{true si } (T_n > 100) \\ A_{n-1} \text{ sinon} \end{cases}$



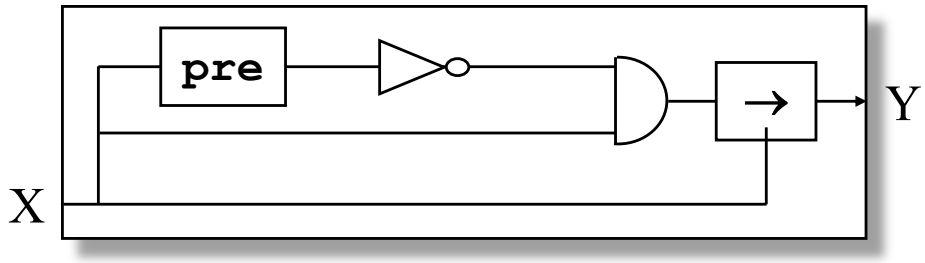
3.3.2. Exemples

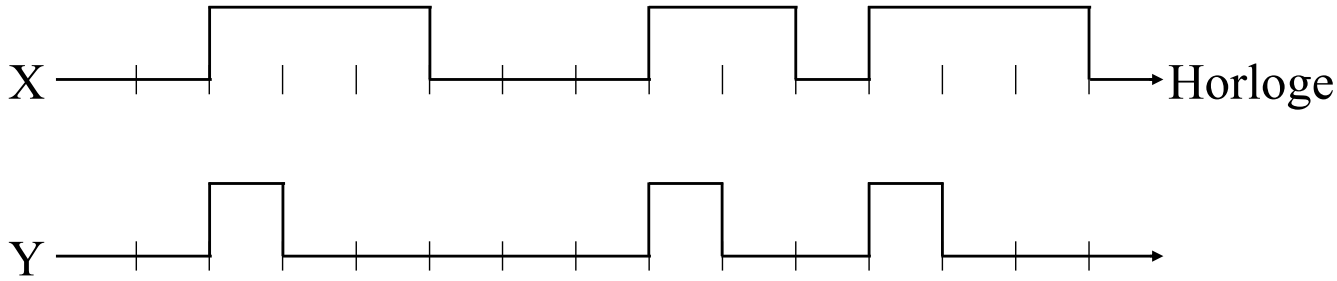
Exemple : détection de fronts montants

Soit X un flot d'entrée booléen

Soit Y un flot de sortie booléen

```
node rising_edge (X : bool) returns (Y : bool) ;
let
   Y = X -> (X and not pre(X));
tel ;
```



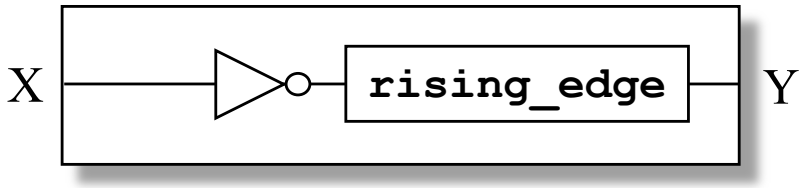


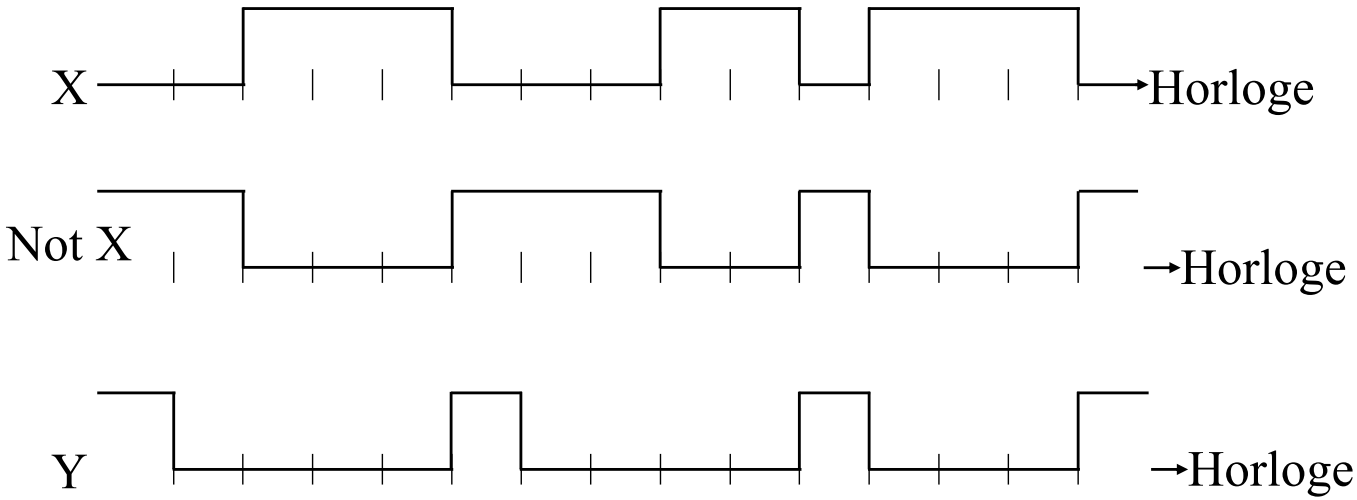
3.3.2. Exemples

Exemple : détection de fronts descendants

Réutilisation de l'opérateur **EDGE**

```
node falling_edge (X : bool) returns (Y : bool) ;
let
   Y = rising_edge (not X) ;
tel ;
```





3.3.3. Les assertions

La notion d'assertion

Permet au concepteur d'expliciter les hypothèses faites sur l'environnement et/ou sur le programme lui-même

=> permet d'optimiser la compilation

=> permet la vérification de propriétés sous conditions

=> simplifie la conception des programmes

Exemple :

```
assert (not (X and Y))
```

affirme que les flots booléens X et Y ne doivent jamais être vrais simultanément

```
assert (true -> not (X and pre(X)))
```

affirme que le flot booléen X ne transporte jamais deux valeurs vraies consécutives

3.3.3. Les assertions : exemple

Exemple : Justin, le loup, la chèvre et le chou…

Les entrées du programmes : les actions de Justin

- m : Justin traverse la rivière seul
- mw : Justin traverse la rivière avec le loup
- mg : Justin traverse la rivière avec la chèvre
- mc : Justin traverse la rivière avec le chou

=> flots booléens

b_n = vrai signifie que l'action b est effectuée à l'instant *n*

b_n = faux signifie que l'action b n'est pas effectuée à l'instant *n*

Hypothèse : les actions sont instantanées

Les sorties du programmes

- J : position de Justin
- W : position du loup
- G : position de la chèvre
- C : position du chou

=> flots entiers à valeurs dans {0, 1, 2}

X_n = 0 signifie X est sur la rive 0 à l'instant *n*

X_n = 1 signifie X est sur la rive 1 à l'instant *n*

X_n = 2 signifie X a été mangé à l'instant *n*

3.3.3. Les assertions : exemple

Exemple : Justin, le loup, la chèvre et le chou…

```
node justin(m, mw, mg, mc : bool) returns (J, W, G, C : int);
let
   assert (m or mw or mg or mc);
   assert( not (m and mw));
   assert( not (m and mg));
   assert( not (m and mc));
   assert( not (mw and mg));
   assert( not (mw and mc));
   assert( not (mg and mc));
   assert( true -> not (mw and not (pre(J)=pre(W))));
   assert( true -> not (mg and not (pre(J)=pre(G))));
   assert( true -> not (mc and not (pre(J)=pre(C))));

   J = 0 -> 1 - pre(J);
   W = 0 -> if mw then 1 - pre(W) else pre(W);
   G = 0 -> if pre(G) = 2 then pre(G)
            else if mg then 1 - pre(G)
                     else if (pre(G)=pre(W) and not mw) then 2
                              else pre(G);
   C = 0 -> if pre(C) = 2 then pre(C)
            else if mc then 1 - pre(C)
                     else if (pre(C)=pre(G) and not mg) then 2
                              else pre(C);
tel.
```

3.3.3. Les assertions : exemple

Exemple : Justin, le loup, la chèvre et le chou…

```
…
J = 0 -> 1 - pre(J);
W = 0 -> if mw then 1 - pre(W) else pre(W);
G = 0 -> if pre(G) = 2 then pre(G)
              else if mg then 1 - pre(G)
                       else if (pre(G)=pre(W) and not mw) then 2
                                else pre(G);
C = 0 -> if pre(C) = 2 then pre(C)
              else if mc then 1 - pre(C)
                       else if (pre(C)=pre(G) and not mg) then 2
                                else pre(C);
```

Séquence d'actions gagnante : . , mg, m, mw, mg, mc, m, mg

Que se passe-t-il pour la séquence : m, m, mw, m … ?

3.3.4. Les tableaux

Les tableaux

types scalaires

`bool`, `int`, `real`

=> types tableaux

`bool^4`, `int^n` avec n constante, `real^4^8`, ...

Mais : un programme Lustre doit s'exécuter en temps et espace borné. Les dimensions et indices des tableaux doivent donc être statiques et connus à la compilation

=> pré-compilation en Lustre sans tableau.

3.3.4. Les tableaux

Les tableaux : exemple

```
node Tdelay (const d : int; X : bool) returns(Y : bool);
var A : bool^(d+1);
let
   A[0] = X;
   A[1..d] = false^d -> pre(A[0..d-1]);
   Y = A[d];
tel


node Main (A : bool) returns (A_delayed : bool);
let
    A_delayed = Tdelay(10,A);
tel
```

3.3.5. La récursion

La récursion

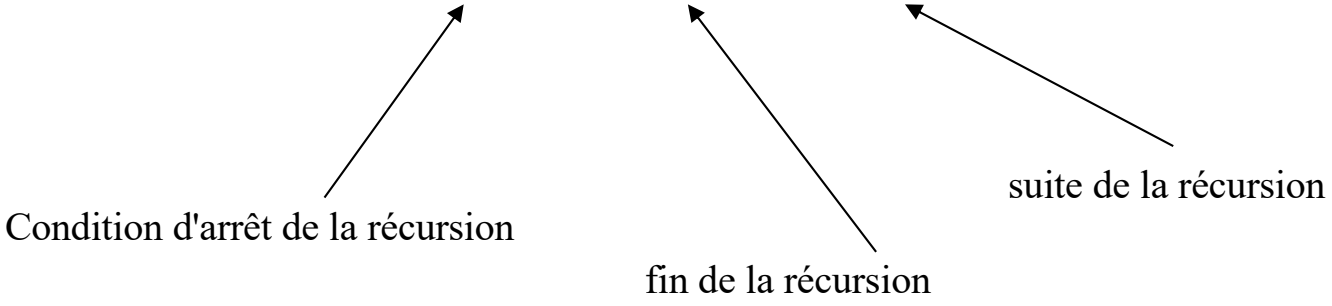
Un programme Lustre doit s'exécuter en temps et espace borné

=> la profondeur de la récursion doit être bornée statiquement et connue à la compilation

=> compilation en Lustre sans récursion

=> un opérateur conditionnel statique

```
X = with … then … else … ;
```



3.3.5. La récursion

La récursion : exemple

```
node Rdelay (const d : int; x : bool) returns(y : bool);
let
  Y = with d=0 then X else (false -> pre(Rdelay(d-1, X)));
tel


node Main (A : bool) returns (A_delayed : bool);
let
   A_delayed = Rdelay(10, A);
tel
```

3.4. La causalité

Problème de causalité :

Rappel : un réseau d'opérateurs de flots de données calcule un point fixe

Problème : un tel point fixe peut ne pas exister, être partiellement indéterminé, ou être multiple

=> programmes non causaux

Exemple :

`Y = X+Y`	non causal
`Y = X + pre(Y)`	indéterminé
`Y = X -> X + pre(Y)`	OK

=> Le rebouclage instantané des sorties sur les entrées est interdit

=> Le rebouclage retardé des sorties dur les entrées est autorisé à condition qu'il passe par au moins autant d'instance de"->" que de "pre"