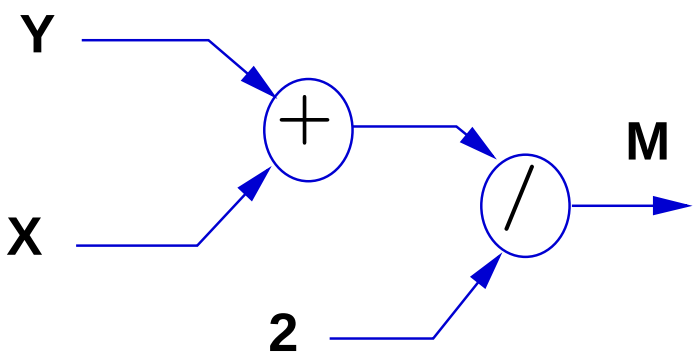
Le langage Lustre

Pascal Raymond, Verimag-CNRS

Approche flot de données

Classique en automatique et en conception de circuits




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

Interprétation synchrone : temps $= \mathbb{N}$

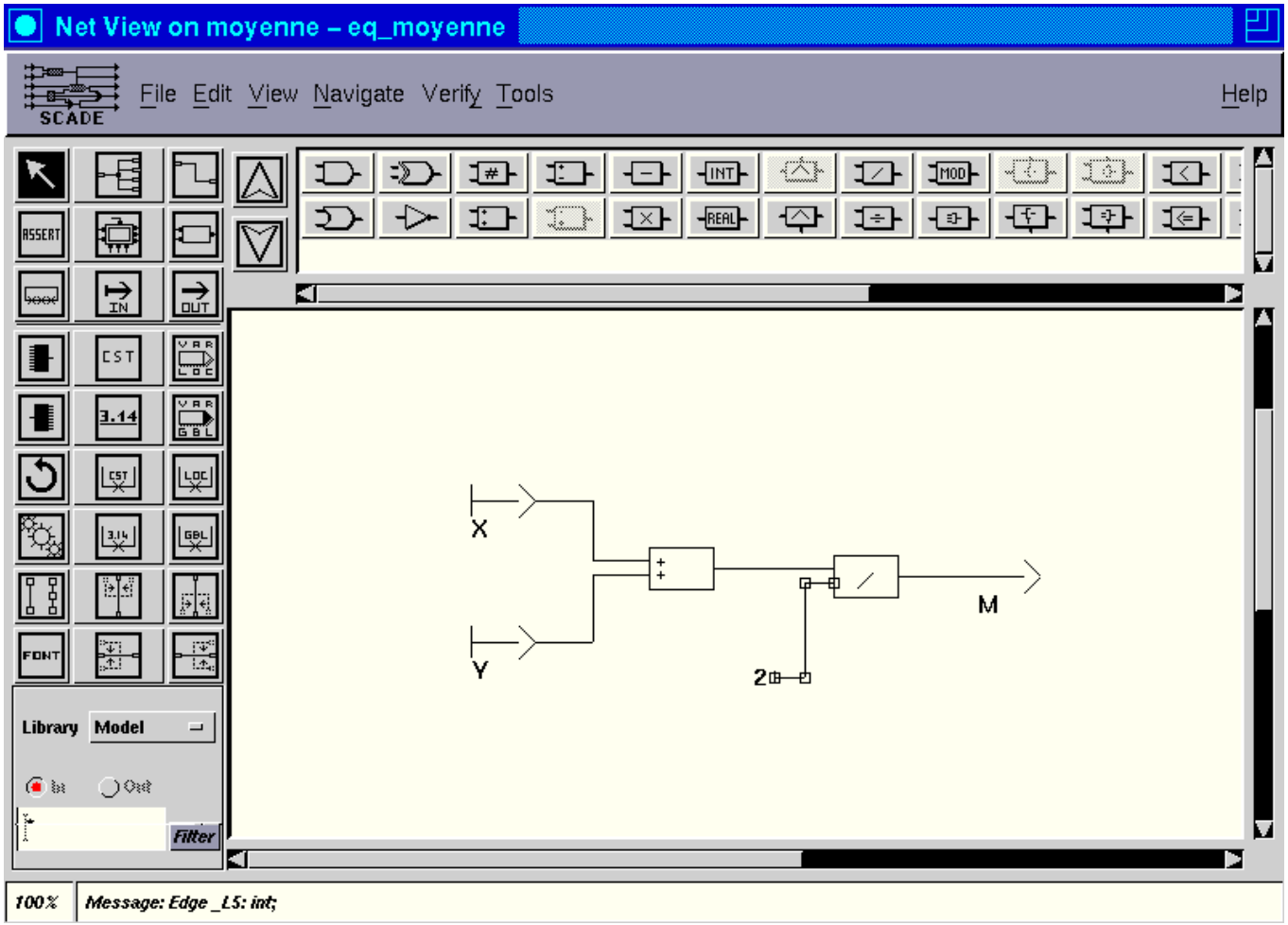
$$\forall t \in \mathbb{N} \;\; M_t = (X_t + Y_t)/2$$

Autre version

```
node Moyenne(X, Y : int)
returns (M : int);
var S : int;        ← variable auxiliaire
let
   M = S / 2;       ← équations
   S = X + Y;       (ordre non significatif)
tel
```

- une définition pour chaque sortie et variable auxiliaire
- ordre indifférent
- principe de substitution
- X, Y, M, S dénotent des séquences infinies de valeurs

Lustre (académique) vs SCADE (graphique)



Lustre combinatoire

Les types de base

- booléen (`bool`), entier (`int`), flottant (`real`)

Les constantes

- **2** $\equiv 2, 2, 2, 2, \ldots$
- `true` $\equiv vrai, vrai, vrai, vrai, \ldots$

Les opérateurs “point à point”

- opérateurs arithmétiques et logiques classiques **X** $\equiv x_0, x_1, x_2, x_3 \ldots$ **Y** $\equiv y_0, y_1, y_2, y_3 \ldots$

  $\Rightarrow$ **X + Y** $\equiv x_0 + y_0, x_1 + y_1, x_2 + y_2, x_3 + y_3 \ldots$

Exemple booléen

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

Exécution :

X	*vrai*	*vrai*	*faux*	*vrai*	*vrai*	*faux*	...
Y	*faux*	*vrai*	*faux*	*faux*	*vrai*	*faux*	...
U	*faux*	*vrai*	*faux*	*faux*	*vrai*	*faux*	...
Z	*vrai*	*faux*	*vrai*	*vrai*	*faux*	*vrai*	...

Exemple : l'opérateur `if`

```
node Max(A,B: real) returns (M: real);
let
   M = if (A >= B) then A else B;
tel
```

- **Erreur classique :**

```
let
  if (A >= B) then M = A ;
  else M = B ;
tel
```

- `if` : (flot bool) $\times$ (flot T) $\times$ (flot T) $\longrightarrow$ (flot T)

Mémoire

Opérateur `pre` (“précédent”)

- retard élémentaire

X	x_0	x_1	x_2	x_3	x_4	...
pre X	nil	x_0	x_1	x_2	x_3	...

- i.e. $(\texttt{pre}X)_0$ indéfini et $\forall i \neq 0 \ (\texttt{pre}X)_i = X_{i-1}$

Opérateur `->` (“suivi de”)

- initialisation

X	x_0	x_1	x_2	x_3	x_4	...
Y	y_0	y_1	y_2	y_3	y_4	...
X ->Y	x_0	y_1	y_2	y_3	y_4	...

- i.e. $(X\texttt{->}Y)_0 = X_0$ et $\forall i \neq 0 \ (X\texttt{->}Y)_i = Y_i$

Détection des fronts montants

```
node Edge (X : bool) returns (E : bool);
let
   E = false -> X and not pre X ;
tel
```

X	*faux*	*faux*	*vrai*	*vrai*	*faux*	*vrai*	...
pre X	nil	*faux*	*faux*	*vrai*	*vrai*	*faux*	...
E	*faux*	*faux*	*vrai*	*faux*	*faux*	*vrai*	...

Minimum et maximum d'une séquence

```
node MinMax(X : int) returns (min, max :  int);
let
  min = X -> if (X < pre min) then X else pre min;
  max = X -> if (X > pre max) then X else pre max;
tel
```

X	12	5	7	−2	21	0	...
min	12	5	5	−2	−2	−2	...
max	12	12	12	12	21	21	...

⇒ **Définition récursive de flot**

Définitions récursives

Définition correcte

- La séquence peut être *calculée pas-à-pas*
- i.e. la récursion ne porte que sur le passé
- i.e. pas de court-circuit
- Exemple : `alt = false -> not pre alt`

  faux *vrai* *faux* *vrai* *faux* ...

EXO. Un flot `nat` qui vaut $0, 1, 2, 3, 4, \ldots$?

Définitions incorrectes

- `X = 1 / (2 - X);`
- Il y a bien une unique solution : `X = 1`
- MAIS pas constructible pas-à-pas
- Refusée par le(s) compilateur(s)
  - ⋆ cas général : indécidabilité
  - ⋆ cas particulier(s) : trop cher

Remarque

- Problème plus vaste : causalité
- Inhérent au synchronisme

Règle à retenir : pas de court-circuit en Lustre

Exercices

(Double initialisation) Définir :

- un flot **P** $\equiv$ *faux*, *faux*, *vrai*, *faux*, *vrai*, *faux*, *vrai*...
- (Fibonacci) un flot **F** $\equiv$ 1, 1, 2, 3, 5, 8, 13 ...
- `node Compteur(X:bool) returns (C:int);`
  tel que **C** est incrémenté quand **X** est vrai
- `node CompteurReset(X,reset:bool) returns (C:int);`
  qui remet à zéro la sortie quand `reset` est vrai
- `node Bascule(on,off:bool) returns (X:int);`
  qui met à **X** à vrai si `on` et à fuax si `off`

Modularité

Réutilisation

- Tout nœud défini par l'utilisateur peut être réutilisé
- Instanciation dans un style fonctionnel

Exemple

- temporisation à la montée
  - ⋆ X n'est prise en compte que si elle est maintenue plus de n centièmes de seconde,
  - ⋆ cs est vraie à chaque centième de seconde,
  - ⋆ la sortie Y est la commande temporisée.

```
node Tempo(X,cs:  bool; n:int)
returns (Y:bool);
var cpt :  int;
let
  cpt = CompteurReset(cs, not X);
  Y = (cpt >= n);
tel
```

Exos

- Le même, mais tel que `cpt` ne puisse pas diverger ?
- Que vaut

  `A = CompteurReset(true, true -> (pre A = 4));` ?

Réutilisation d'un nœud à plusieurs résultats

```
node MoyenneMinMax(X : int) returns (M : int);
var min, max :  int;
let
  M = Moyenne(min, max);
  min, max = MinMax(X);
tel
```

Horloges

Échantillonnage : opérateur `when`

- Définir un flot “plus lent” que les entrées

`X`	4	1	-3	0	2	7	8
`C`	*vrai*	*faux*	*faux*	*vrai*	*vrai*	*faux*	*vrai*
`X when C`	4			0	2		8

- Quand `C` est faux, `X when C` n’existe pas
- On ne peut opérer que sur des flots de même horloge

  exemple : “`X + (X when C)`” interdit !

Projection : opérateur `current`

- **Ramène un flot sur une horloge plus rapide**

X	**4**	**1**	**-3**	**0**	**2**	**7**	**8**
C	*vrai*	*faux*	*faux*	*vrai*	*vrai*	*faux*	*vrai*
Y = X when C	**4**			**0**	**2**		**8**
Z = current(Y)	**4**	**4**	**4**	**0**	**2**	**2**	**8**

- **N.B.** `current(X when C)` $\neq$ `X`

Erreur classique : défaut d'initialisation

X	4	1	-3	0	2	7	8
C	*faux*	*faux*	*faux*	*vrai*	*vrai*	*faux*	*vrai*
current(X when C)	nil	nil	nil	0	2	2	8

Astuce : échantillonner avec des horloges initialement vraies

C1 = true -> C	*vrai*	*faux*	*faux*	*vrai*	*vrai*	*faux*	*vrai*
current(X when C1)	4	4	4	0	2	2	8

Autre solution, forcer une valeur par défaut :

```
E = if C then current(X when C) else (dft -> pre E);
```

Ou encore (si on veut éviter la mémoire supplémentaire) :

```
X1 = (if C then X else dft) -> X;
E = current(X1 when C1);
```

Nœuds et horloges

- **Horloge effective d'une instance de nœud $=$ l'horloge de ses paramètres effectifs d'entrée**
- **échantillonner les entrées d'un nœud $\Rightarrow$ forcer tout "l'intérieur" du nœud appelé à fonctionner plus lentement que le nœud appelant**

Échantillonner les entrées $\neq$ Échantillonner les sorties

C	*vrai*	*vrai*	*faux*	*faux*	*vrai*	*faux*	*vrai*
`Compteur(true when C)`	1	2			3		4
`Compteur(true) when C`	1	2			5		7

Sous-échantillonnage

- **On peut sous-échantillonner un flot déjà échantillonné**
- **`X when C` correct ⇔ `X` et `C` ont la même horloge**
- **`current` projette sur l'horloge *immédiatement* plus rapide**

`X`	4	1	-3	0	2	7	8	13
`Y`	*vrai*	*faux*	*vrai*	*vrai*	*vrai*	*faux*	*faux*	*vrai*
`C`	*vrai*	*vrai*	*faux*	*vrai*	*vrai*	*faux*	*vrai*	*vrai*
`Z = X when C`	4	1		0	2		8	13
`H = Y when C`	*vrai*	*faux*		*vrai*	*vrai*		*faux*	*vrai*
`T = Z when H`	4			0	2			13
`current T`	4	4		0	2		2	13

Cohérence des horloges

- Pour un nœud, l'horloge la plus rapide est l'horloge de base
- Les paramètres d'entrée sont sur l'horloge de base*
- Les constantes sont sur l'horloge de base
- On ne peut opérer que sur des flots qui ont la même horloge
- L'horloge de `X when C` est `C`
- L'horloge de `current X` est l'horloge de l'horloge de `X`
- L'horloge de `X op Y` est l'horloge de `X` = l'horloge de `Y`

Problème : décider de la cohérence des horloges ?

* sauf indication contraire

Inférence ou Vérification ?

- Problème classique $\equiv$ inférence vs vérification de type
- Inférer : le compilateur doit, à partir de sa définition, “calculer” l’horloge de chaque variable
- Vérifier : le programmeur doit déclarer l’horloge de chaque variable, le compilateur se contente de vérifier que le programme est cohérent

En Lustre :

- pas d’inférence
- équivalence des horloges purement syntaxique

  Astuce : nommer tous les flots utilisés comme horloge

Exemple de nœud “multi-horloges”

```
node MultiHorloge(
  X, Y : int; C : bool;    ← horloge de base
  (Z : int) when C    ← horloge d'interface
) returns (
  (S : int) when C    ← horloge d'interface
);
var (H : bool) when C;
  (U : int) when H;
let
  H = (true when C) -> ((X + Y) when C) < Z;
  U = (Z when H) -> Moyenne(Z when H, pre U);
  S = current(U);
tel
```

Exercices : circuits en Lustre

Additionneur 3 bits

- **Les booléens sont interprétés comme des chiffres binaires**

  faux $= 0$, *vrai* $= 1$

- **Écrire un nœud (combinatoire) dont l'en-tête est :**

  `node Add3b(cin, x, y : bool) returns (cout, s : bool`

  et qui réalise l'addition binaire de cin, x et y.

  i.e., $\forall t$ $\texttt{cin}_t + \texttt{x}_t + \texttt{y}_t = 2 * \texttt{cout}_t + \texttt{s}_t$

Additionneur série

- Les flots de booléens sont interprétés (quand c'est possible) comme des nombres binaires "arbitrairement long" :
  $X \equiv X_0 + X_1 * 2 + X_2 * 2^2 + ... + X_t * 2^t + ...$
- Quel est l'entier représenté par le flot `false` ?
- Quel est l'entier représenté par le flot `true -> false` ?
- Écrire un additionneur série dont l'en-tête est :
  `node AddSerie(X, Y : bool) returns (S : bool);`
  tel que le flot **S** représente la somme des nombres binaire **X** et **Y**.
- Que vaut le flot `AddSerie(true->false, true)` ?
- Comment peut-on interpréter le flot `true` ?

Multiplicateur par 3

- Écrire un nœud dont l'en-tête est :

  `node Fois3(X : bool) returns (T : bool);`

  tel que le nombre binaire représenté par `T` soit égal à 3 fois le nombre binaire représenté par `X`.

- Soit le flot `alt = true -> not pre alt;` (i.e. `alt` $\equiv 1\,0\,1\,0\,1\,0\ldots$)

  Quel est le résultat de `Fois3(alt)` ?

- Comment peut-on (raisonnablement) interpréter le flot `alt` ?