Parallélisme à flot de données: le langage synchrone Lustre

Nicolas Halbwachs

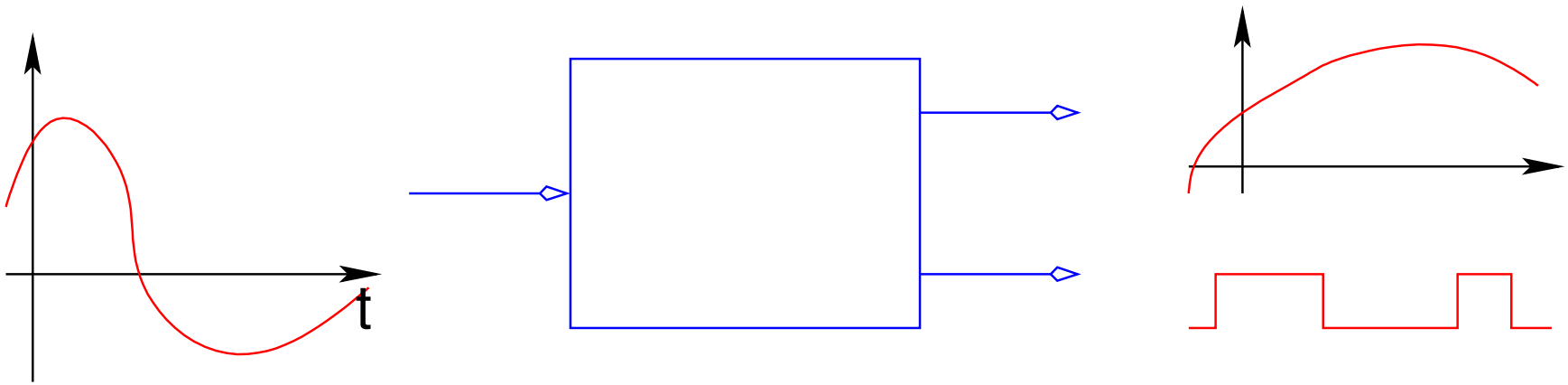
Verimag/CNRS
Grenoble

Programmation des systèmes embarqués

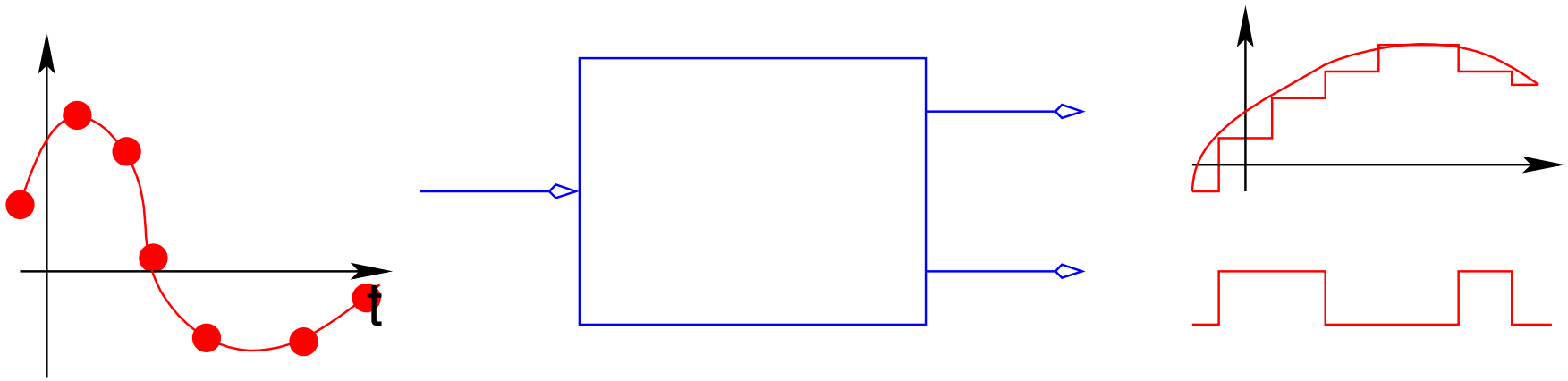
Programmes contrôlant un environnement physique

- interaction “en temps réel”
- actions irréversibles
- systèmes critiques

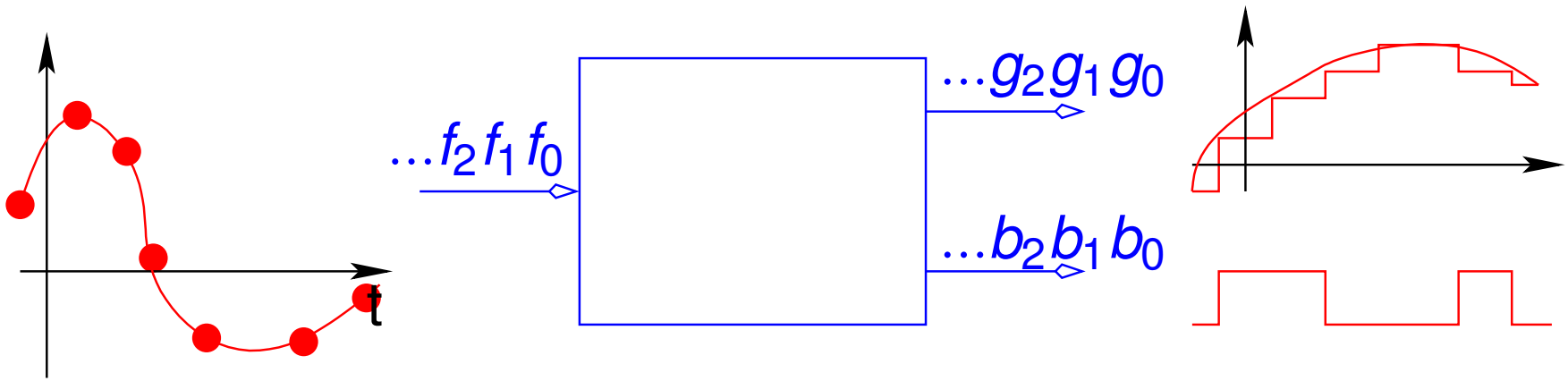
L'approche "flot de données"



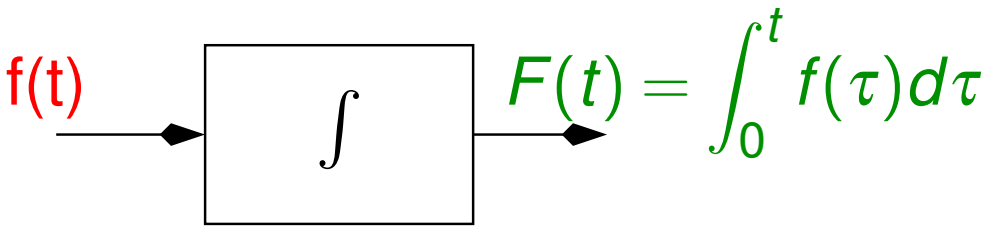
L'approche "flot de données"



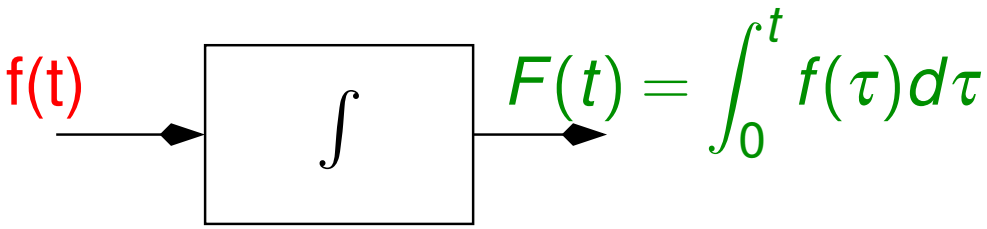
L'approche "flot de données"

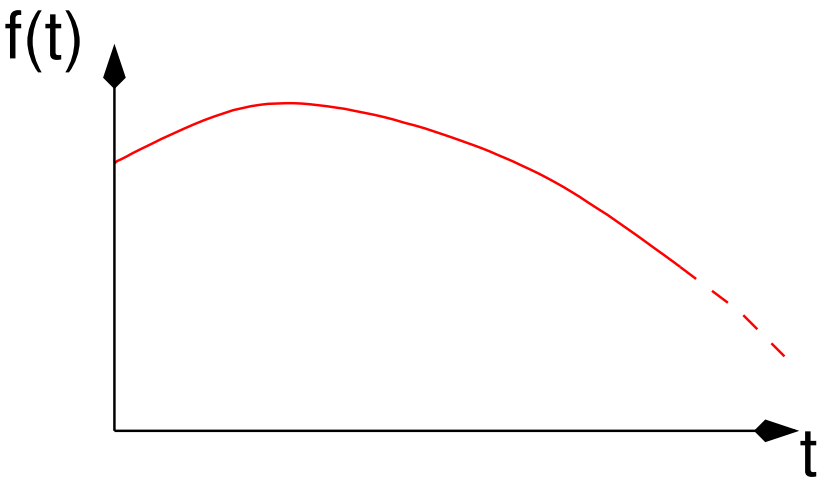


Exemple: intégration d'une fonction

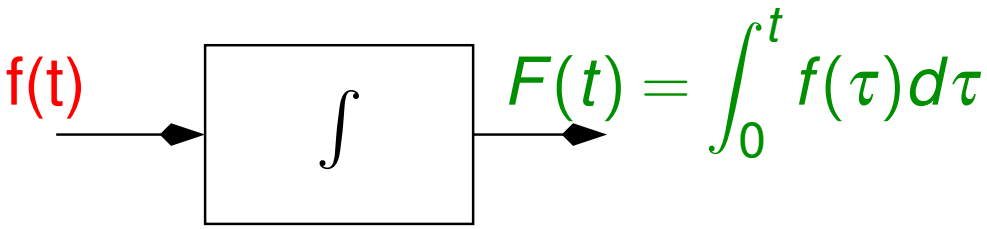


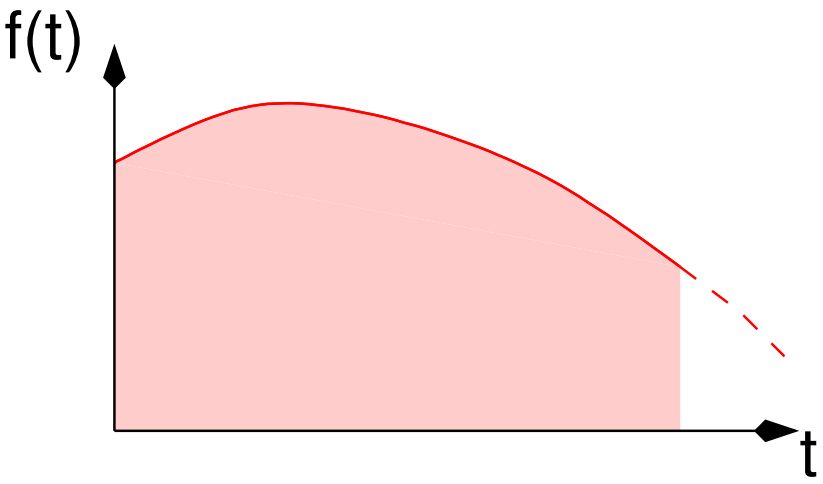
Exemple: intégration d'une fonction



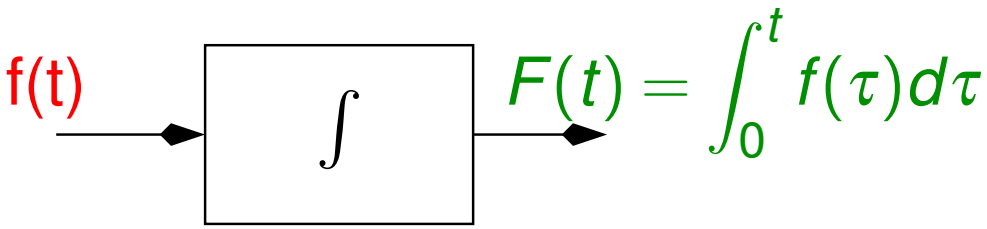


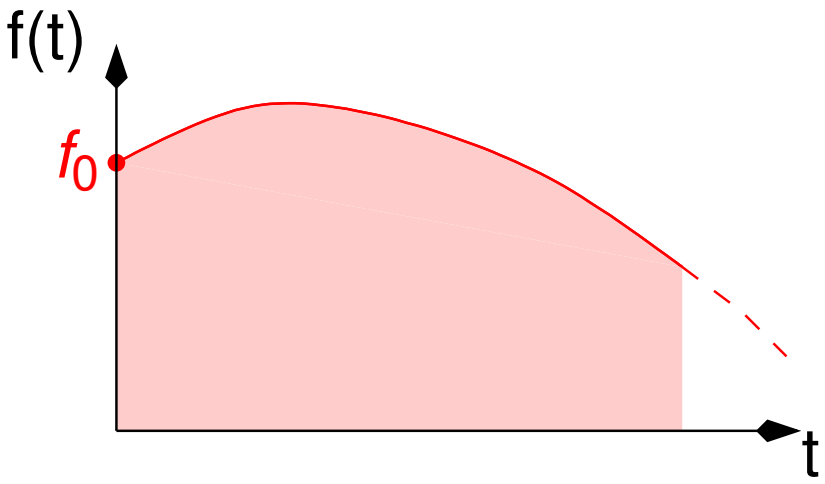
Exemple: intégration d'une fonction



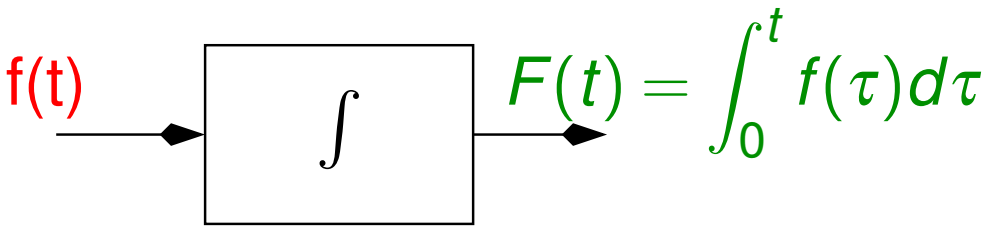


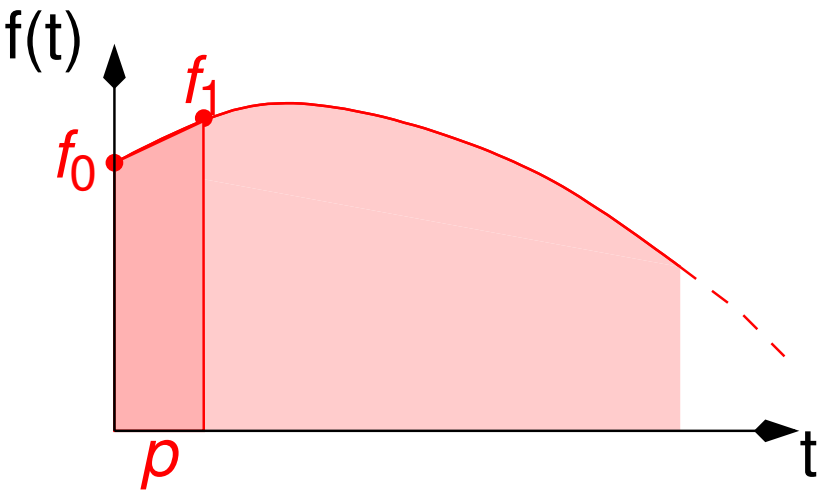
Exemple: intégration d'une fonction



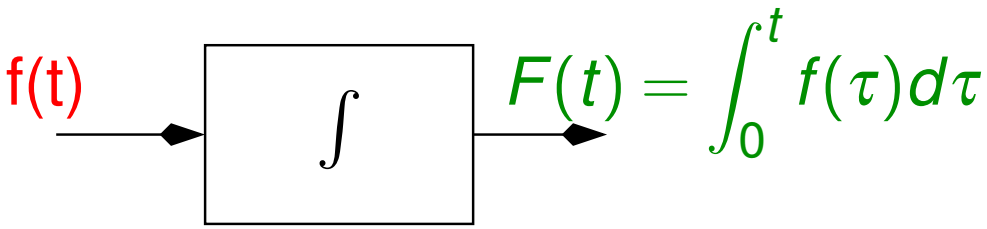


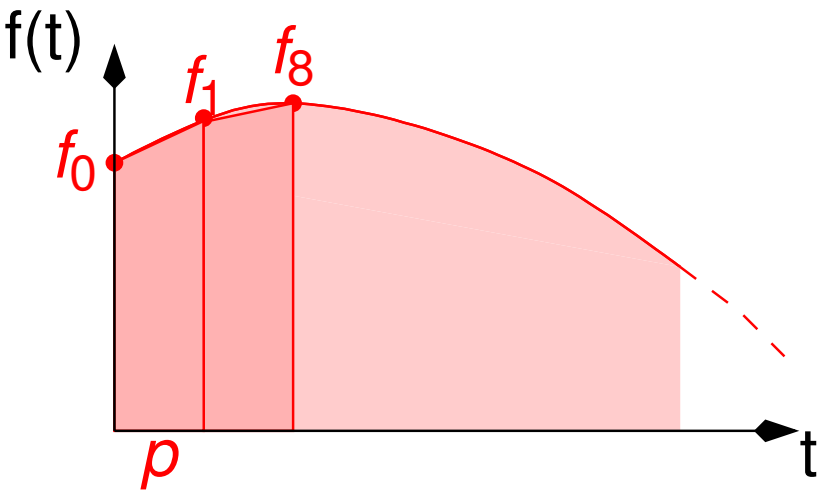
Exemple: intégration d'une fonction



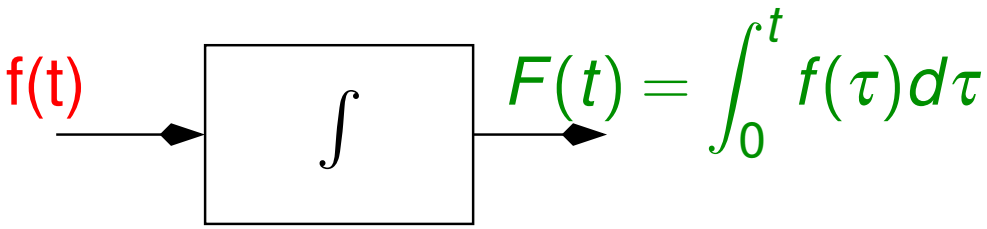


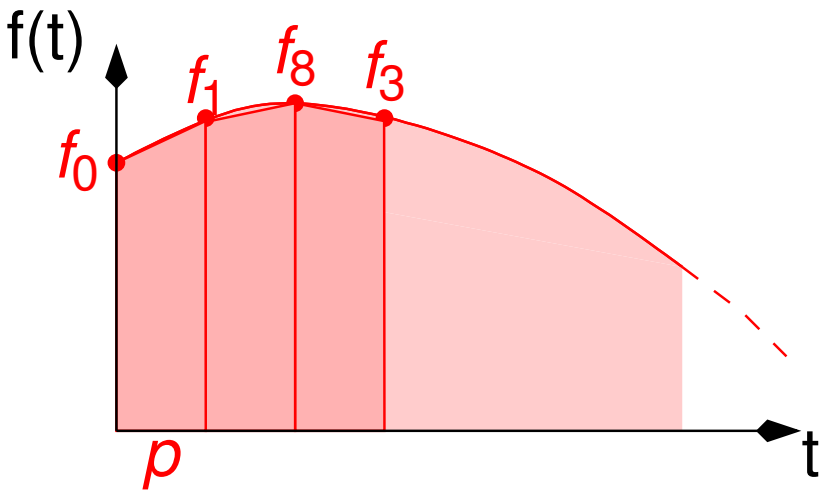
Exemple: intégration d'une fonction



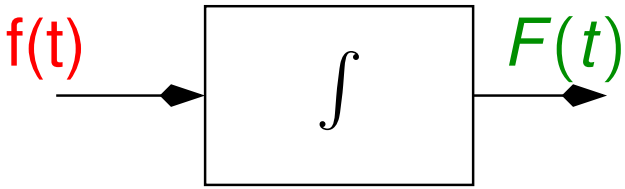


Exemple: intégration d'une fonction

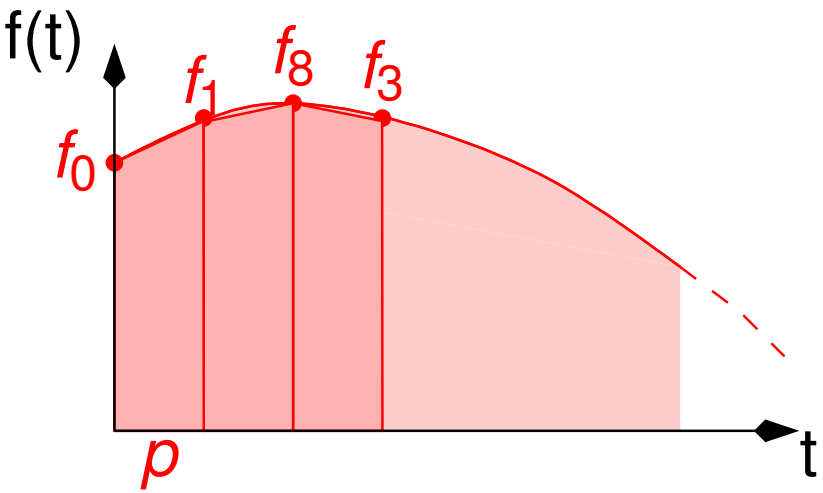




Exemple: intégration d'une fonction



$$F(t) = \int_0^t f(\tau) d\tau$$



$$F_0 = 0$$

$$F_i = F_{i-1} + p(f_i + f_{i-1})/2$$

Principe d'un langage "flot de données"

variable = suite de valeurs (prises au cours du temps)

x_n valeur de x au n-ième pas d'exécution

un programme définit des suites de sortie en fonction de suites d'entrée

Principe d'un langage "flot de données"

variable = suite de valeurs (prises au cours du temps)

x_n valeur de x au n-ième pas d'exécution

un programme définit des suites de sortie en fonction de suites d'entrée

Problèmes:

- $x_n = y_{n+1} - z_n$

Principe d'un langage "flot de données"

variable = suite de valeurs (prises au cours du temps)

x_n valeur de x au n-ième pas d'exécution

un programme définit des suites de sortie en fonction de suites d'entrée

Problèmes:

- $x_n = y_{n+1} - z_n$ (le présent dépend du futur)

Principe d'un langage "flot de données"

variable = suite de valeurs (prises au cours du temps)

x_n valeur de x au n-ième pas d'exécution

un programme définit des suites de sortie en fonction de suites d'entrée

Problèmes:

- $x_n = y_{n+1} - z_n$ (le présent dépend du futur)
- $x_n = 2x_n - 1$

Principe d'un langage "flot de données"

variable = suite de valeurs (prises au cours du temps)

x_n valeur de x au n-ième pas d'exécution

un programme définit des suites de sortie en fonction de suites d'entrée

Problèmes:

- $x_n = y_{n+1} - z_n$ (le présent dépend du futur)
- $x_n = 2x_n - 1$ (une valeur dépend d'elle-même)

Principe d'un langage "flot de données"

variable = suite de valeurs (prises au cours du temps)

x_n valeur de x au n-ième pas d'exécution

un programme définit des suites de sortie en fonction de suites d'entrée

Problèmes:

- $x_n = y_{n+1} - z_n$ (le présent dépend du futur)
- $x_n = 2x_n - 1$ (une valeur dépend d'elle-même)
- $x_{2n} = x_n + y_n$

Principe d'un langage "flot de données"

variable = suite de valeurs (prises au cours du temps)

x_n valeur de x au n-ième pas d'exécution

un programme définit des suites de sortie en fonction de suites d'entrée

Problèmes:

- $x_n = y_{n+1} - z_n$ (le présent dépend du futur)
- $x_n = 2x_n - 1$ (une valeur dépend d'elle-même)
- $x_{2n} = x_n + y_n$ (mémoire non bornée)

Principe d'un langage "flot de données"

variable = suite de valeurs (prises au cours du temps)

x_n valeur de x au n-ième pas d'exécution

un programme définit des suites de sortie en fonction de suites d'entrée

Problèmes:

- $x_n = y_{n+1} - z_n$ (le présent dépend du futur)
- $x_n = 2x_n - 1$ (une valeur dépend d'elle-même)
- $x_{2n} = x_n + y_n$ (mémoire non bornée)

Restreindre le pouvoir d'expression

Le langage Lustre [Paul Caspi, NH, 1984]

Toute variable ou expression représente une suite infinie de valeurs:

- Variables : x représente la suite infinie $(x_0, x_1, \ldots, x_n, \ldots)$
- Constantes : 42 représente la suite $(42, 42, \ldots 42, \ldots)$

Programme = système d'équations

Opérations classiques :

- $x + y$ représente la suite $(x_0 + y_0, x_1 + y_1, \ldots x_n + y_n, \ldots)$
- *if c then x else y* représente la suite $(z_0, z_1, \ldots)$ avec

$$z_n = \begin{cases} x_n & \text{si } c_n = \text{vrai} \\ y_n & \text{sinon} \end{cases}$$

Le langage Lustre [Paul Caspi, NH, 1984]

Toute variable ou expression représente une suite infinie de valeurs:

- Variables : x représente la suite infinie $(x_0, x_1, \ldots, x_n, \ldots)$
- Constantes : 42 représente la suite $(42, 42, \ldots 42, \ldots)$

Programme = système d'équations

Opérations classiques :

- $x + y$ représente la suite $(x_0 + y_0, x_1 + y_1, \ldots x_n + y_n, \ldots)$
- *if c then x else y* représente la suite $(z_0, z_1, \ldots)$ avec

$$z_n = \begin{cases} x_n & \text{si } c_n = \text{vrai} \\ y_n & \text{sinon} \end{cases}$$

ex: max = if a $>$ b then a else b

$$max_n = \begin{cases} a_n & \text{si } a_n > b_n \\ b_n & \text{sinon} \end{cases}$$

Le langage Lustre (2/6)

2 opérateurs temporels:

- “precedent” :
  pre(x) représente la suite $(nil, x_0, x_1, \ldots, x_{n-1}, \ldots)$
- “suivi-de” :
  x->y représente la suite $(x_0, y_1, y_2, \ldots, y_n, \ldots)$

Le langage Lustre (2/6)

2 opérateurs temporels:

- "precedent" :
  pre(x) représente la suite $(nil, x_0, x_1, \ldots, x_{n-1}, \ldots)$
- "suivi-de" :
  x->y représente la suite $(x_0, y_1, y_2, \ldots, y_n, \ldots)$

Exemple: x = 0 -> pre(x) + 1

pre(x)	*nil*
x	0

Le langage Lustre (2/6)

2 opérateurs temporels:

- "precedent" :
  pre(x) représente la suite $(nil, x_0, x_1, \ldots, x_{n-1}, \ldots)$
- "suivi-de" :
  x->y représente la suite $(x_0, y_1, y_2, \ldots, y_n, \ldots)$

Exemple: x = 0 -> pre(x) + 1

pre(x)	*nil*	0
x	0	1

Le langage Lustre (2/6)

2 opérateurs temporels:

- "precedent" :
  pre(x) représente la suite $(nil, x_0, x_1, \ldots, x_{n-1}, \ldots)$
- "suivi-de" :
  x->y représente la suite $(x_0, y_1, y_2, \ldots, y_n, \ldots)$

Exemple: x = 0 -> pre(x) + 1

pre(x)	*nil*	0	1
x	0	1	2

Le langage Lustre (2/6)

2 opérateurs temporels:

- "precedent" :
  pre(x) représente la suite $(nil, x_0, x_1, \ldots, x_{n-1}, \ldots)$
- "suivi-de" :
  x->y représente la suite $(x_0, y_1, y_2, \ldots, y_n, \ldots)$

Exemple: x = 0 -> pre(x) + 1

pre(x)	*nil*	0	1	2	3	4	5	...
x	0	1	2	3	4	5	6	...

Le langage Lustre (3/6)

Autre exemple: l'intégrateur

$$F_0 = 0\ , \quad F_i = F_{i-1} + p.\frac{f_i + f_{i-1}}{2}$$

F = 0 -> (pre(F) + p*(f+pre(f)/2);

Le langage Lustre (3/6)

Autre exemple: l'intégrateur

$$F_0 = 0\ , \quad F_i = F_{i-1} + p.\frac{f_i + f_{i-1}}{2}$$

```
F = 0 -> (pre(F) + p*(f+pre(f)/2);
```

```
F = 0 -> pre(F) + tpz ;
tpz = p*(f+pre(f)/2 ;
```

Le langage Lustre (3/6)

Autre exemple: l'intégrateur

$$F_0 = 0\,, \quad F_i = F_{i-1} + p.\frac{f_i + f_{i-1}}{2}$$

```
F = 0 -> pre(F) + tpz ;
tpz = p*(f+pre(f)/2 ;
```

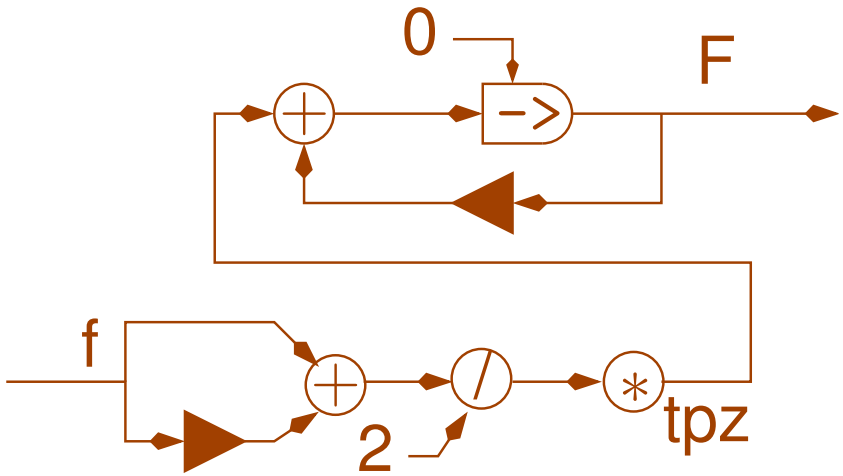
Le langage Lustre (3/6)

Autre exemple: l'intégrateur

$$F_0 = 0\,, \quad F_i = F_{i-1} + p.\frac{f_i + f_{i-1}}{2}$$

F = 0 -> pre(F) + tpz ;

tpz = p*(f+pre(f)/2 ;

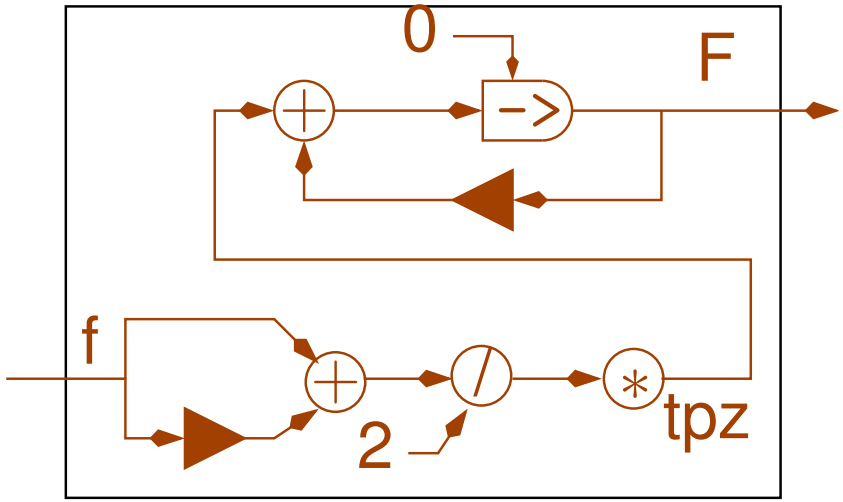


Le langage Lustre (3/6)

Autre exemple: l'intégrateur

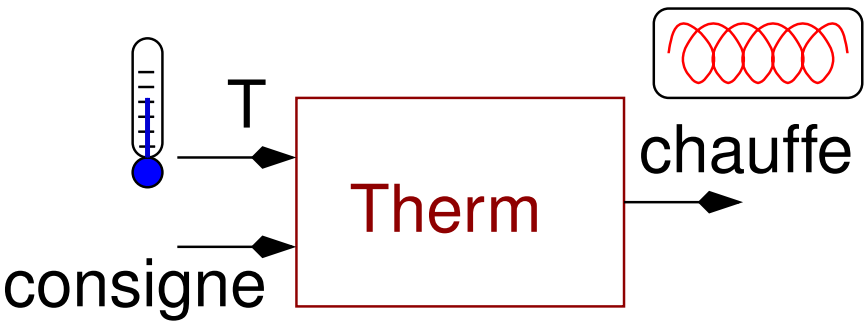
$$F_0 = 0\,, \quad F_i = F_{i-1} + p.\frac{f_i + f_{i-1}}{2}$$

```
node Integ (f: real; d: real) returns (F: real);
var tpz: real;
let
    F = 0 -> pre(F) + tpz ;
    tpz = p*(f+pre(f)/2 ;
end
```



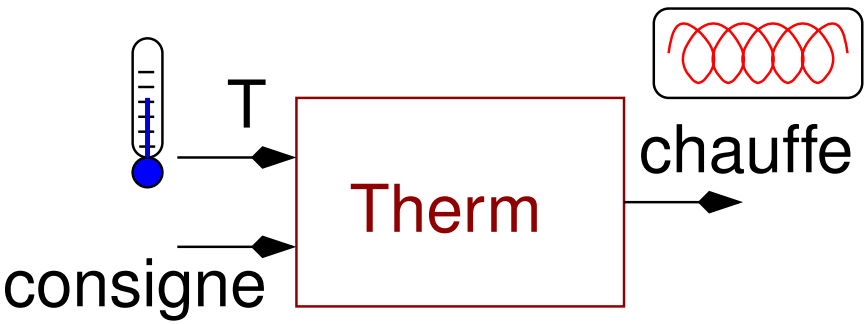
Le langage Lustre (4/6)

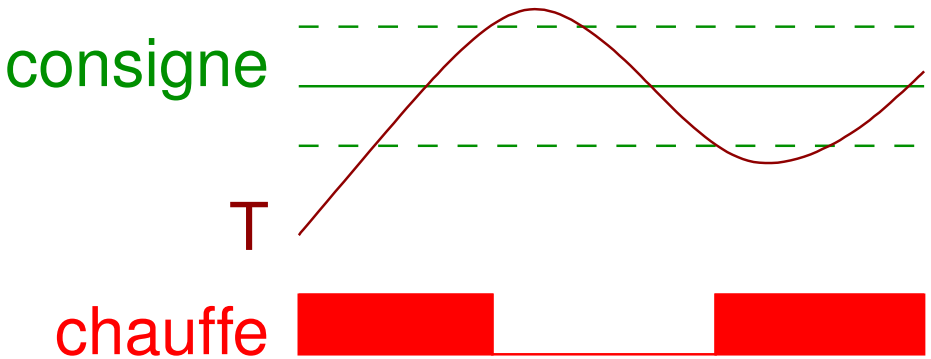
Autre exemple: un thermostat



Le langage Lustre (4/6)

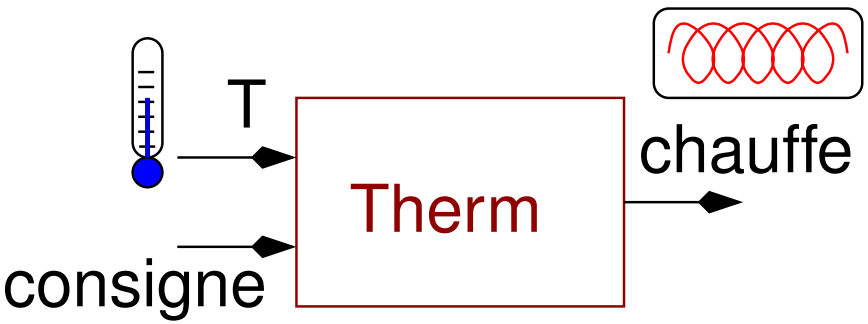
Autre exemple: un thermostat

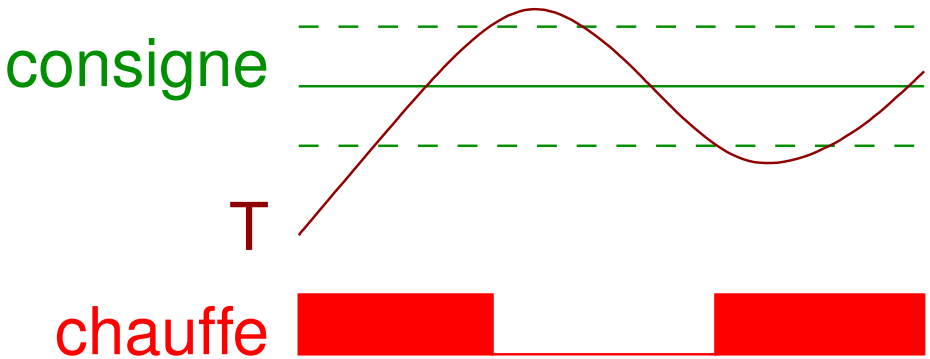




Le langage Lustre (4/6)

Autre exemple: un thermostat

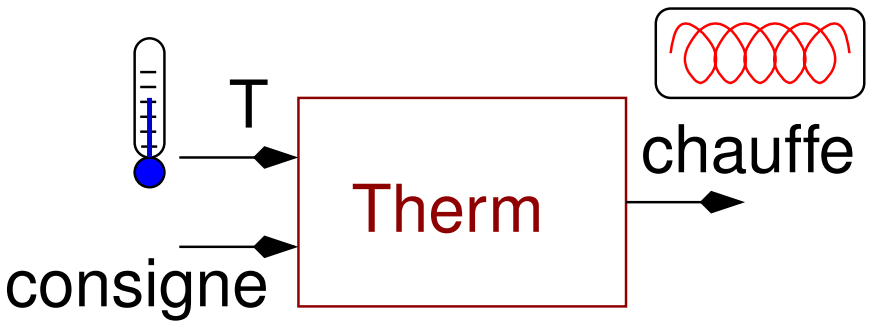


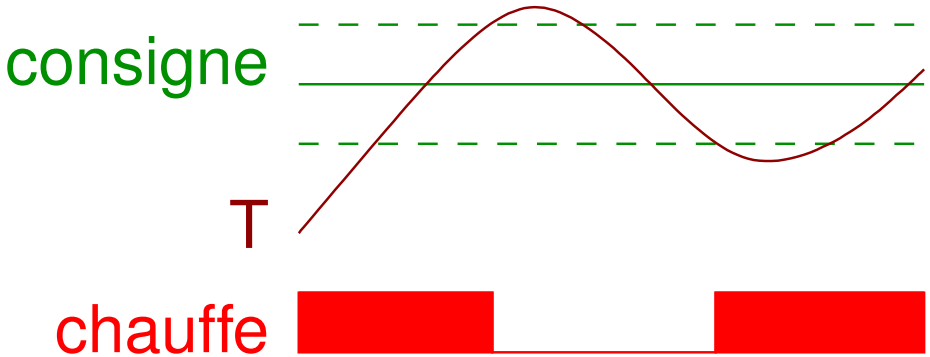



```
chauffe = true -> if (T > consigne + 1) then false
                  else if (T < consigne -1) then true
                  else pre(chauffe);
```

Le langage Lustre (4/6)

Autre exemple: un thermostat

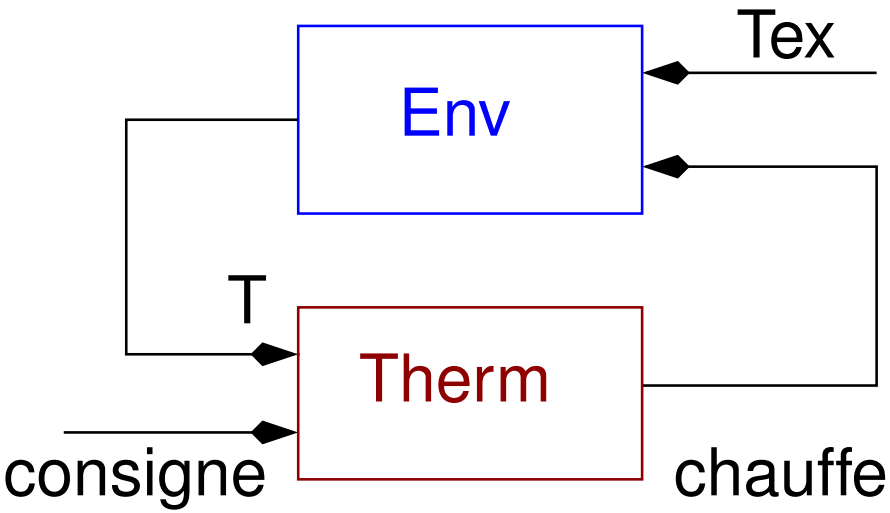





```
node Therm (T, consigne : real) returns (chauffe: bool);
let
     chauffe = true -> if (T > consigne + 1) then false
                       else if (T < consigne -1) then true
                       else pre(chauffe);
tel
```

Le langage Lustre (5/6)

Thermostat: simulation de l'environnement

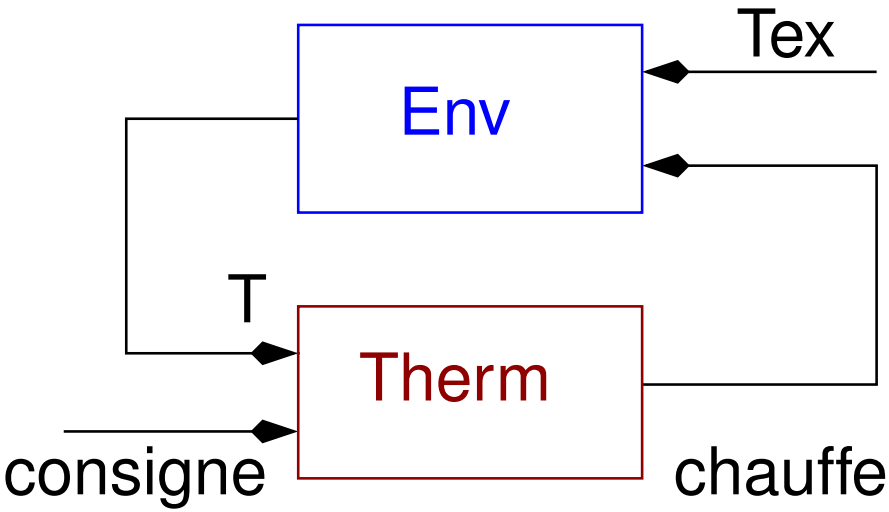


Quand le radiateur chauffe : $T' = \alpha(h - T)$

Quand il ne chauffe pas : $T' = \beta(T_{ex} - T)$

Le langage Lustre (5/6)

Thermostat: simulation de l'environnement



Quand le radiateur chauffe : $T' = \alpha(h - T)$

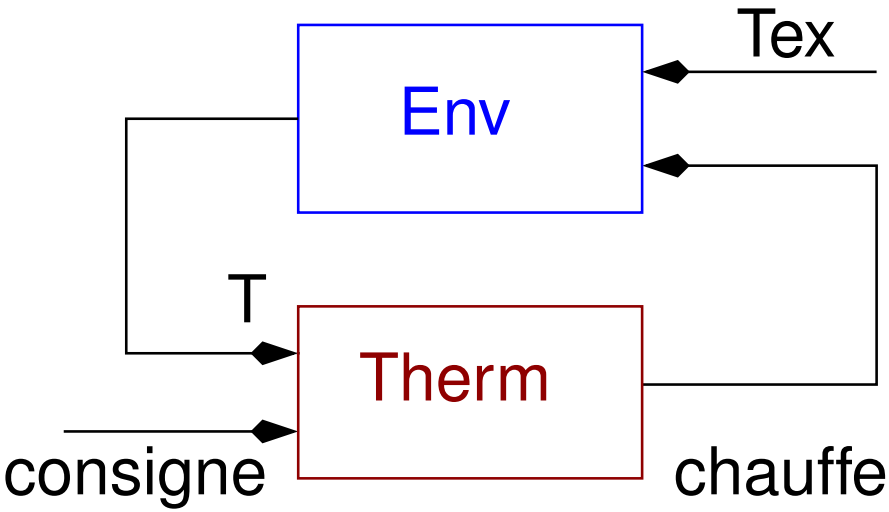
Quand il ne chauffe pas : $T' = \beta(T_{ex} - T)$

```
node Env (chauffe: bool; Tex: real) returns (T: real);
var Tprime : real;
let
   Tprime = if chauffe then alpha * (h-T) else beta * (Tex-T);
   T = Integ(Tprime, d);

tel
```

Le langage Lustre (5/6)

Thermostat: simulation de l'environnement



Quand le radiateur chauffe : $T' = \alpha(h - T)$

Quand il ne chauffe pas : $T' = \beta(T_{ex} - T)$

```
node Env (chauffe: bool; Tex: real) returns (T: real);
var Tprime : real;
let
   Tprime = if chauffe then alpha * (h-T) else beta * (Tex-T);
   T = Integ(Tprime, d);
   T = Tex -> Integ(0.0 -> pre(Tprime), d);
tel
```

Le langage Lustre (6/6)

Le programme de simulation

```
node Simul (consigne, Tex: real)
     returns (chauffe: bool; T: real);
let
     chauffe = Therm(T, consigne);
     T = Env(chauffe, Tex);
tel
```

DEMO

Les horloges

Permettre à des parties de programmes d'évoluer à des rythmes différents

Opérateurs "when" et "current

C	T	F	T	T	F	T	F	F	T
x	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
y=x when C	x_0		x_2	x_3		x_5			x_8

Les horloges

Permettre à des parties de programmes d'évoluer à des rythmes différents

Opérateurs "when" et "current

C	T	F	T	T	F	T	F	F	T
x	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
y=x when C	x_0		x_2	x_3		x_5			x_8
z=x+y									

Les horloges

Permettre à des parties de programmes d'évoluer à des rythmes différents

Opérateurs "when" et "current

C	T	F	T	T	F	T	F	F	T
x	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
y=x when C	x_0		x_2	x_3		x_5			x_8
~~z=x+y~~									

Les horloges

Permettre à des parties de programmes d'évoluer à des rythmes différents

Opérateurs "when" et "current

C	T	F	T	T	F	T	F	F	T
x	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8
y=x when C	x_0		x_2	x_3		x_5			x_8
~~z=x+y~~									
current(y)	x_0	x_0	x_2	x_3	x_3	x_5	x_5	x_5	x_8

Contrainte (mémoire bornée, synchronisme strict): les opérandes d'un opérateur doivent être sur la même horloge

Les horloges

Les opérateurs s'exécutent au rythme de leurs opérandes:

C = true -> not pre(C)	T	F	T	F	T
x = 0 -> pre(x)+1	0	1	2	3	4

Les horloges

Les opérateurs s'exécutent au rythme de leurs opérandes:

C = true -> not pre(C)	T	F	T	F	T
x = 0 -> pre(x)+1	0	1	2	3	4
y = x when C	0		2		4

Les horloges

Les opérateurs s'exécutent au rythme de leurs opérandes:

C = true -> not pre(C)	T	F	T	F	T
x = 0 -> pre(x)+1	0	1	2	3	4
y = x when C	0		2		4
zero = 0 when C	0		0		0
un = 1 when C	1		1		1
z = zero -> pre(z)+ un	0		1		2

Quelques qualités d'un langage de programmation :

- sémantique formelle, abstraite
- pouvoir d'expression adéquat (assez puissant pour être commode, assez restreint pour permettre une vérification de cohérence et une exécution efficace)
- structuration naturelle des programmes
- parallélisme "logique"

Compilation

1. vérifier la cohérence du programme
   - correction des horloges
   - causalité
2. produire du code séquentiel

```
<initialisations ;>
while true do
    lire les entrées ;
    <Calculer les variables dans un ordre causal> ;
    <Emettre les sorties> ;
    <Mettre à jour les mémoires> ;
end
```

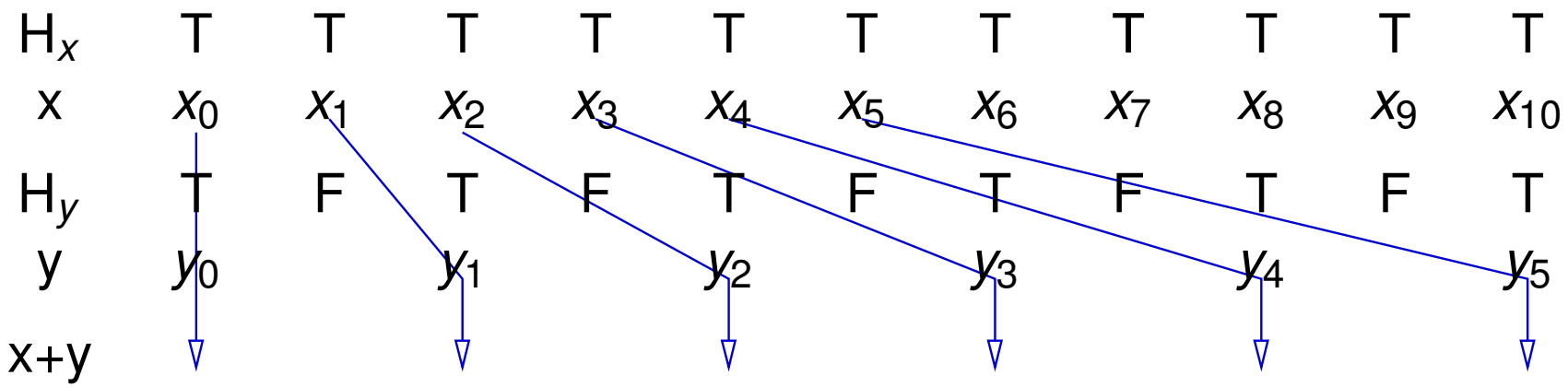
Autres résultats

- Expression de propriétés et vérification de programmes
  ex: (f >= 0) => (F >= pre(F))
- Production de code non séquentiel [Caspi]
  - multitâche
  - distribué
- Simulation synchrone de systèmes non synchrones
- *n*-synchronisme

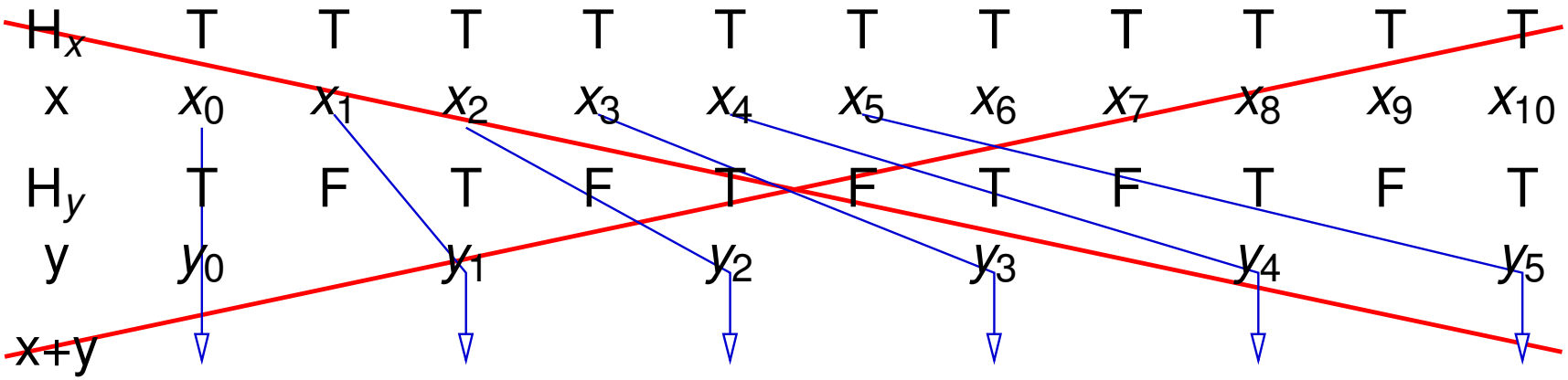
N-synchronisme [Pouzet, Plateau, Mandel, Cohen]

H_x	T	T	T	T	T	T	T	T	T	T	T
x	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
H_y	T	F	T	F	T	F	T	F	T	F	T
y	y_0		y_1		y_2		y_3		y_4		y_5
x+y											

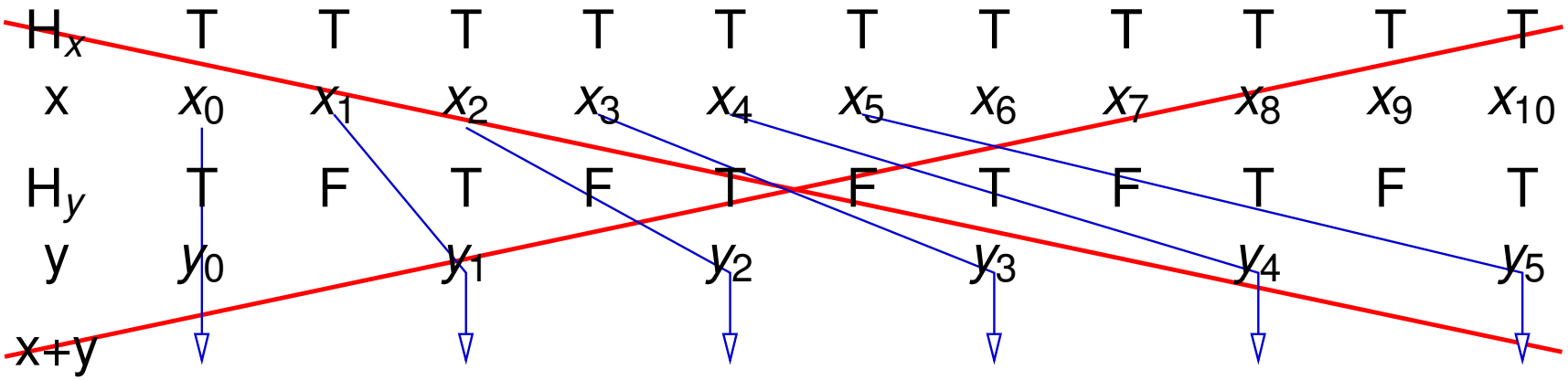
N-synchronisme [Pouzet, Plateau, Mandel, Cohen]

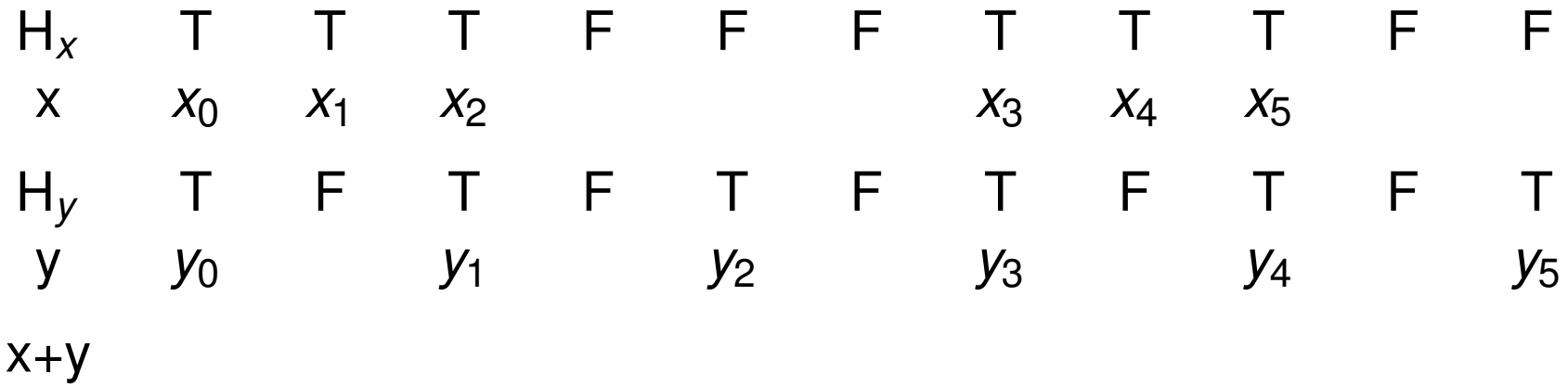


N-synchronisme [Pouzet, Plateau, Mandel, Cohen]

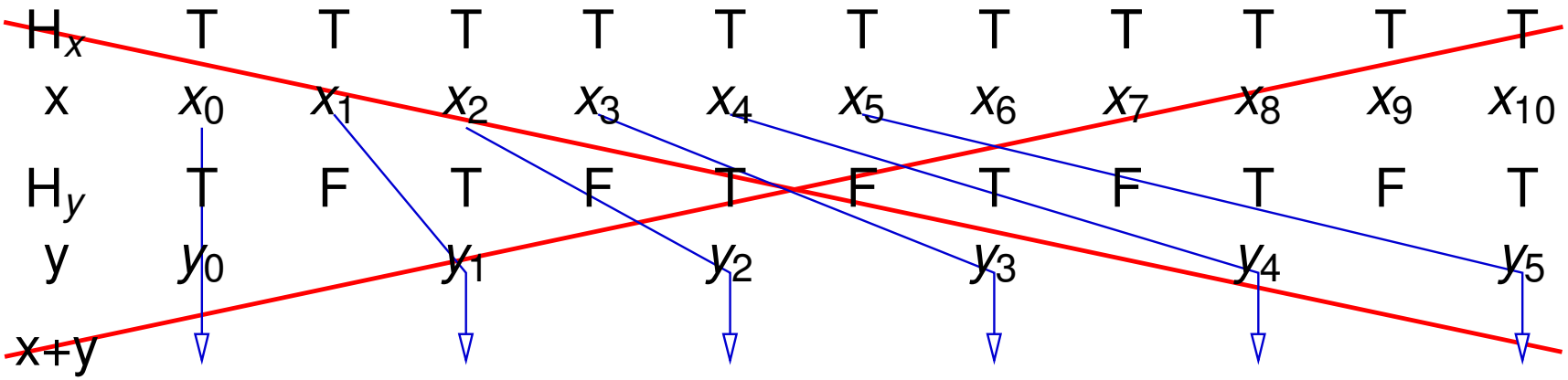


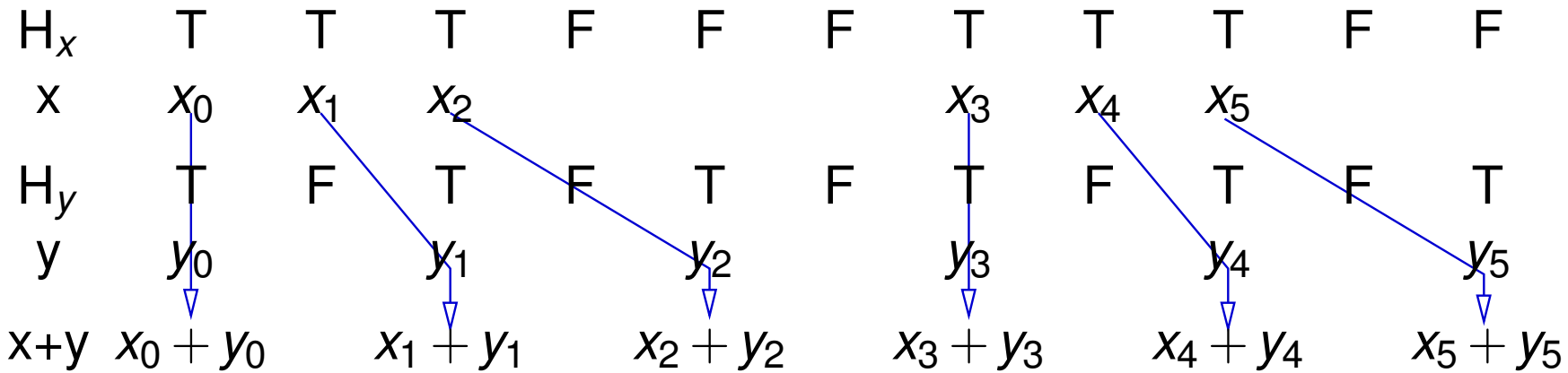
N-synchronisme [Pouzet, Plateau, Mandel, Cohen]



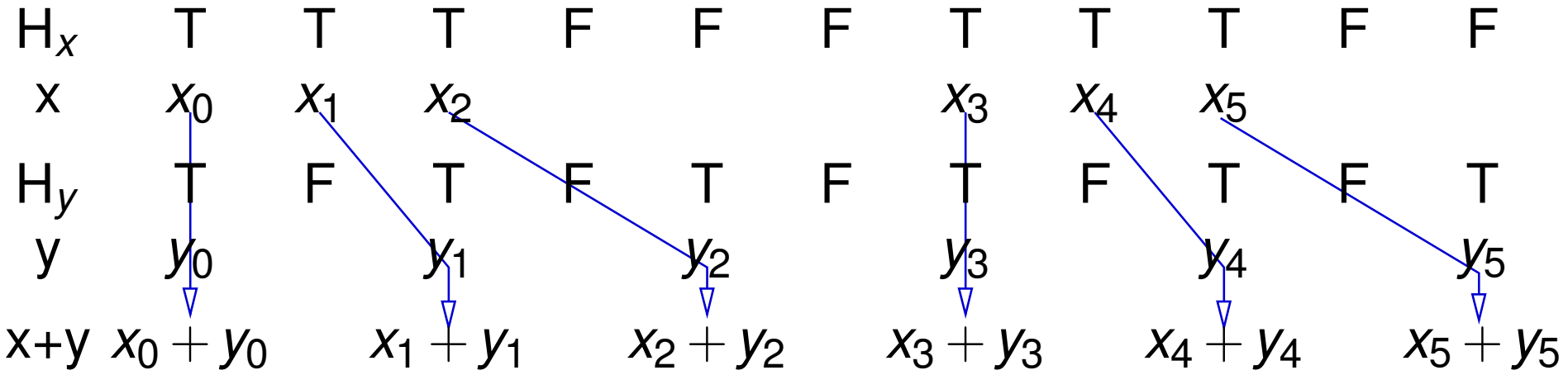


N-synchronisme [Pouzet, Plateau, Mandel, Cohen]





N-synchronisme



- Exécution possible en introduisant des tampons (réseaux de Kahn à files bornées)
- Nécessité de connaître les horloges statiquement

  mots infinis périodiques: $H_x = (T\ T\ T\ F\ F\ F)^\infty$, $H_y = (T\ F)^\infty$
- Vérifier la compatibilité de H_x et H_y, calculer la taille des tampons

Outils industriels fondés sur le modèle flot de données synchrone

Scade: Atelier fondé sur Lustre, développé par la société Esterel-technologies, utilisé dans le monde entier

Outils industriels fondés sur le modèle flot de données synchrone

Scade: Atelier fondé sur Lustre, développé par la société Esterel-technologies, utilisé dans le monde entier

RT-builder: Outil de modélisation fondé sur le langage Signal [Benveniste, Le Guernic], développé par Geensys

Matlab-Simulink: Outil très complet de simulation, standard du domaine

Contributeurs

J.-L. Bergerand
F. Carrier
P. Caspi
A. Cohen
A. Curic
C. Dumas
F. Gaucher
A. Girault
A-C. Glory-Kerbrat
N. Halbwachs
E. Jahier
C. Kossentini
L. Mandel
F. Maraninchi
J. Mikac
L. Morel
D. Pilaud
E. Pilaud
J. Plaice
F. Plateau
M. Pouzet
Ch. Ratel
P. Raymond
Y. Rémond
F. Rocheteau
Y. Roux
R. Salem
N. Scaife
S. Tripakis
D. Weber

www-verimag.imag.fr/SYNCHRONE