INF 321
Paradigme réactif synchrone

Eric Goubault

Cours 10

23 juin 2014

Dans le dernier épisode...

On a vu:

- Retour sur la programmation fonctionnelle (Caml, Haskell...)!
- Stratégies d'évaluation (*eager/lazy*)
- Typage à la Hindley-Milner (Robin Milner, prix Turing 1991)

On va voir:

- Les langages synchrones, réseaux de Kahn
- LUSTRE et la programmation réactive
- Quelques conseils avant les vacances

Lustre?

Introduction

- Mariage du contrôle et de l'informatique
- Description par schéma-bloc (comme Matlab/Simulink, représentation d'un programme par un graphe!) - mais également langage textuel
- Langage déclaratif: ensemble d'équations

En pratique

- Versions académique LUSTRE, industrielle SCADE (Esterel Technologies)
- Nombreux outils, de preuve, test etc.
- Belle sémantique et utilisé couramment dans le contrôle commande (Airbus, Siemens etc.)!

Réseau de processus synchrones

- Conception démarrée en 1984 au laboratoire Vérimag, à Grenoble (Nicolas Halbwachs, Paul Caspi)
- Transféré depuis 1993 à Esterel Technologies (maintenant ANSYS) sous le nom de SCADE (clients: Airbus, Scheider Electric etc.)
- Synchrone: un message par arc du réseau est envoyé/reçu à chaque "tic" d'horloge (globale) - permet d'éviter l'utilisation de *tampons de communications* potentiellement non bornés; puissance de calcul similaire aux réseaux de Kahn généraux

Réseau de Kahn synchrone!

- Un programme Lustre opère sur un flot = une *suite de valeurs*: une variable `x` en Lustre représente une suite infinie de valeurs $(x_0, x_1, \ldots, x_n, \ldots)$
- x_i est la valeur de `x` au temps i
- Un programme prend un flot et renvoie un flot
- Toutes les opérations sont globales sur un flot:
  - L'équation de flot $x = e$ est un raccourci pour $\forall n,\ x_n = e_n$
  - L'expression arithmétique sur les flots $x + y$ renvoie le flot $(x_0 + y_0, x_1 + y_1, \ldots, x_n + y_n, \ldots)$

Noeuds des équations

- Un programme Lustre est un ensemble d'équations
- Chaque équation (a priori potentiellement mutuellement récursives) est définie par un noeud identifié par le mot clé `node`
- Une équation ou noeud est une fonction prenant des flots en argument, renvoyant un flot en résultat

Structure d'un programme LUSTRE

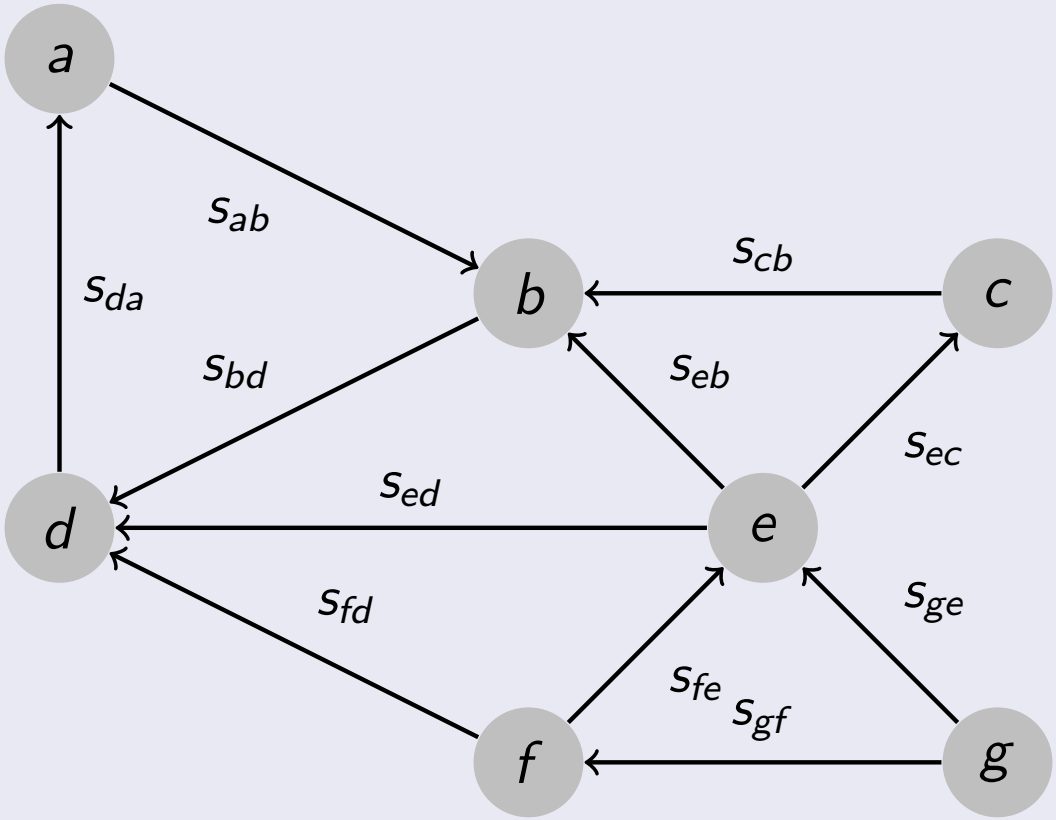
- Définit un ensemble d'équations aux flots de données, entre des noeuds
- Syntaxiquement:

```
[ declaration de constantes ]
[ declaration de types et de fonctions externes ]
node NOM (declaration de flots d'entree)
  returns(declaration de flots de sortie);
[ var declaration de flots internes; ]
let
[ assertions; ]
   Systeme d'equation des flots internes et de
   sortie en fonction des flots internes/d'entree
  EQ1;
  EQ2;
  ...;
  EQn;
tel

[ autres noeuds ]
```

Exemple

Les noeuds et les flots de données entre les noeuds:

Fonctions a, b, c, d, e, f et g des arcs entrant; flots s transitant sur les arcs:



Exemple

```
node saturate(C1, C2: real)
  returns(D: bool);
let
  D=((C1+C2)<20.0)
tel
```

Exemple d'exécution

Horloge	1	2	3	4	...
C1	10	20	15	5	...
C2	5	3	2	10	...
D	true	false	true	true	...

Ce qui peut être "transporté" par les flots

- types de base: `int`, `bool`, `real` (`bool` est le type des flots de booléens etc.)
- tableaux: `int^3`, `real^5^2`, `[int, bool, [int, real]]`, etc.
- n-uplets: `(bool,bool)`, `int,bool,real)` etc.

Aucune fonction récursive, aucun type récursif, aucunes boucles (par contre, les définitions de flots peuvent être mutuellement récursives!)

Opérateurs de base

Constantes

`true` est un flot! (c'est le flot (*true*, *true*, *true*, ...)) de même pour toute constante numérique.

Arithmétiques et comparaison

`+`, `-`, `*`, `div`, `mod`, `/`, `-` unaire, `=`, `<>`, `<`, `>`, `<=`, `>=` toujours sur les flots synchrones (s'évaluent à chaque instant et s'éxécutent "en temps zéro").

Logiques

`or`, `xor`, `and`, `=>`, `not`.

Contrôle

`if . then . else` s'applique encore une fois comme un filtre sur chaque entrée à chaque instant, évalué en "temps zéro".

Exemple

```
node diff(E: real) returns (B: bool; S: real);
let
  B = E>1.0;
  S = if B then 0.0 else E-1.0;
tel
```

Qu'est-ce que cela fait?

Exemple

```
node diff(E: real) returns (B: bool; S: real);
let
  B = E>1.0;
  S = if B then 0.0 else E-1.0;
tel
```

Qu'est-ce que cela fait?

Opérateurs temporels

`pre` (*précédent*)

Donne la valeur au temps précédent, d'un flot argument: `pre(x)` est le flot $(\bot, x_0, \ldots, x_{n-1}, \ldots)$

`->` (*suivi de*)

Est utilisé pour donner des valeurs initiales d'un flot: `x->y` est le flot $(x_0, y_1, \ldots, y_n, \ldots)$

Exemple

Compteur d'événements

```
node Count(evt, reset: bool) returns (count: int);
let
  count = if (true->reset) then 0
          else if evt then pre(count)+1
          else pre(count);
tel
```

Explication informelle

- `true->reset` est un flot booléen: égal à vrai à l'instant initial et quand `reset` est vrai
- quand il est vrai, la valeur de `count` est renvoyée égale à zéro
- sinon, quand `evt` est vrai, on renvoie la valeur à l'instant précédent de `count` plus 1; sinon on conserve l'ancienne valeur

Exemple d'utilisation

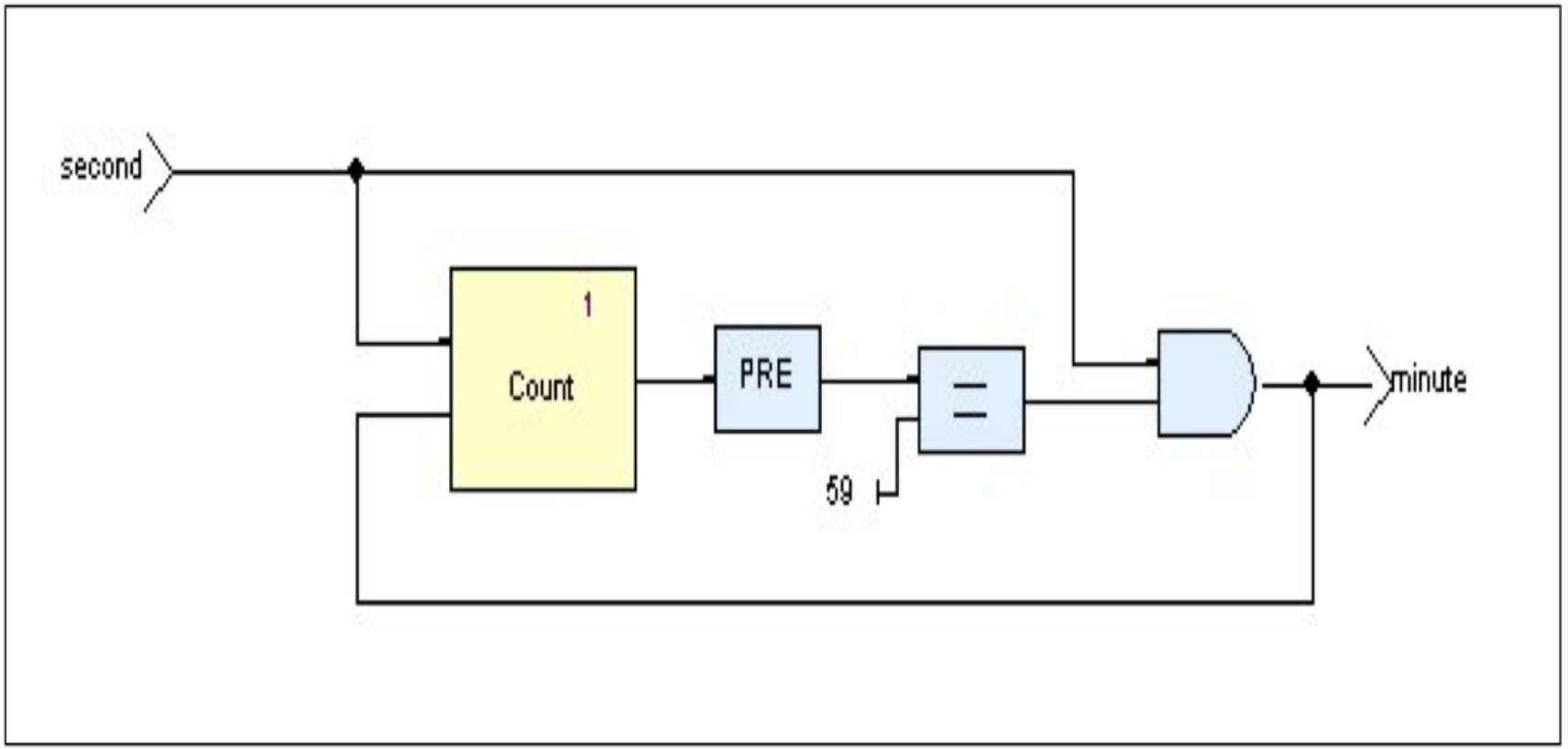
Compter les minutes et secondes

```
mod60 = Count(second, minute);
minute = second and pre(mod60)=59;
```

Explication

- `mod60` est la sortie du noeud `Count`, qui compte les secondes, et se remet à zéro chaque minute
- `minute` est vrai quand `seconde` est cadencé et que sa valeur précédente est de 59

Représentation graphique



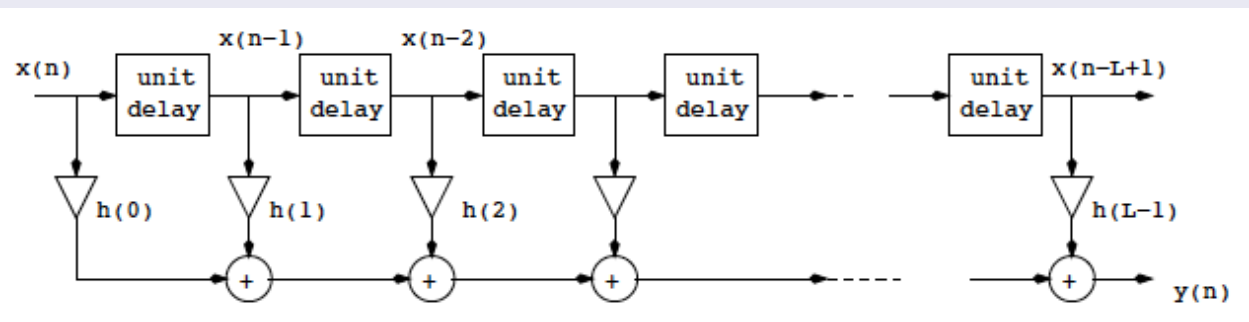
Autre exemple: FIR

Filtres linéaires à réponse finie

- Entrée à l'instant n, x_n
- Sortie à l'instant n, y_n donnée par:

$$y_n = \sum_{m=0}^{L-1} b_m x_{n-m}$$

Graphiquement



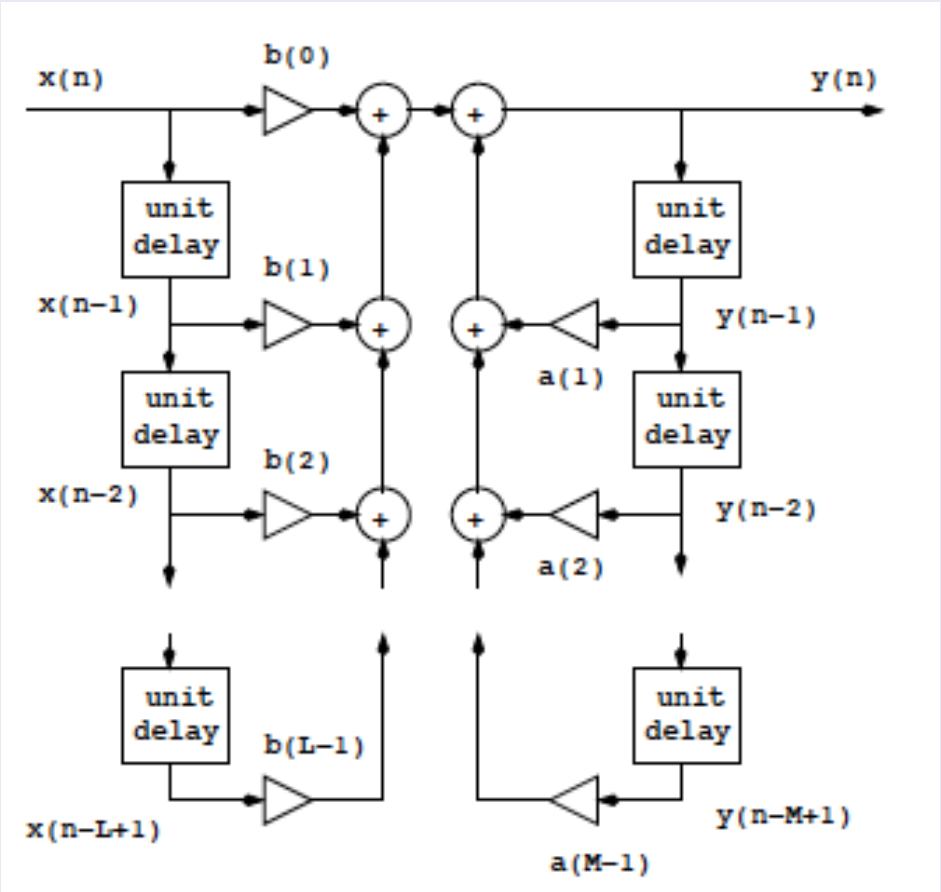
Autre exemple: IIR

Filtres linéaires à réponse infinie (filtres récursifs)

- Entrée à l'instant n, x_n
- Sortie à l'instant n, y_n donnée par:

$$y_n = \sum_{m=0}^{L-1} b_m x_{n-m} + \sum_{m=1}^{M-1} a_m y_{n-m}$$

Graphiquement



Exemple: IIR

Code Lustre

Exemple: $y_n = x_n + 0.9y_{n-1}$:

```
node filter(x: real) returns (y: real);
  let
    y = x+0.0 -> 0.9*pre(y);
  tel;
```

Exemple: chien de garde

"Watchdog"

- permet de gérer des échéances
- émet `alarm` quand `watchdog` est en attente et que `deadline` est vrai:

```
node WATCHDOG1(set, reset, deadline: bool)
     return (alarm: bool);
var watchdog_is_on: bool;
let
  alarm = deadline and watchdog_is_on;
  watchdog_is_on = false -> if set then true
                            else if reset then false
                            else pre(watchdog_is_on);
  assert not(set and reset);
tel;
```

(les flots booléens `set` et `reset` ne doivent pas être vrais en même temps!)

Pause

Et questions? (10 minutes)

Restriction syntaxique

- `let x=x+1;` : le flot `x` dépend instantanément de lui-même, pas possible! (ou alors résolution d'équations, qui ne donnerait pas de solution ici!)
- condition syntaxique imposée: une variable récursive doit être gardée par un délai. On ne peut pas écrire les choses suivantes:

```
x = x+1;
```

ni:

```
x = if b then y else z;
y = if b then t else x;
```

En fait...

- Un flot de données est un couple fait:
  - d'une suite infinie de valeurs d'un certain type
  - d'une horloge définissant les instants où ces valeurs sont définies
- Une horloge est soit:
  - l'horloge de base (celle que l'on a utilisée jusqu'à présent)
  - un flot de données de type booléen

Opérateur de sous échantillonnage `when`

- Permet de cadencer différemment des processus (=noeuds), mais toujours selon un multiple du temps de base
- Opérateur de sous-échantillonage `X when B`, où `X` est un flot quelconque, `B` un flot booléen
- Exemple:

`B`	*false*	*false*	*true*	*true*	*false*	*true*
`X`	X_0	X_1	X_2	X_3	X_4	X_5
`Y=X when B`			X_2	X_3		X_5

Opérateur current

- Injecter un flot lent dans un nouveau flot rapide (cadencé au temps de base)
- Exemple:

B	*false*	*false*	*true*	*true*	*false*	*true*
X	X_0	X_1	X_2	X_3	X_4	X_5
Y=X when B			X_2	X_3		X_5
Z=current Y	$\perp$	$\perp$	X_2	X_3	X_3	X_5

- Remarque: au début Z n'a pas de valeur; on utilise souvent current ...->Y plutôt que current Y

Exemple (Marc Pouzet)

Additionneur

- On considère le code:

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

- On a par exemple:

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

- Donc en général: `f(x when c)`$\neq$`(f x) when c`; de même `current(x when c)`$\neq$`x`

On pourrait vouloir écrire:

```
let half = true -> not (pre half);
    o = x & (x when half)
```

Compilation?

- Le code correspond au calcul $y_n = x_n \& x_{2n}$
- Il faudrait donc un mécanisme de passage de valeurs par buffers
- Qui ici ne serait pas borné! $(n, \ldots, 2n!)$
- Ceci est interdit pas un calcul d'horloge

Calcul d'horloges, encore!

- Les horloges utilisées par un noeud doivent être déclarées et visibles dans l'interface du noeud:
- Exemple: déclaration d'horloge

```
node stables(i: int)
returns (s:int;ncond:bool;(ns: int) when ncond);
```

- puis déclaration d'horloges locales:

```
var cond :bool;
    (l: int) when cond;
```

- puis le code lui-même:

```
let
  cond = true -> i <> pre i; ncond = not cond;
  l = somme(i when cond); s = current(l);
  ns = somme(i when ncond):
tel;
```

Déclarations d'horloges et choix d'implémentation

Les horloges

- les constantes sont cadencées sur l'horloge de base du noeud courant
- par défaut, les variables sont sur l'horloge de base du noeud
- $clock(e_1\ op\ e_2) = clock(e_1) = clock(e_2)$
- $clock(e\ when\ c) = c$
- $clock(current(e)) = clock(clock(e))$

Choix d'implémentation

- Les horloges sont déclarées et vérifiées
- Pas d'inférence, tout est déclaré (ou règles implicites, voir plus haut)
- Deux horloges sont exactes si elles sont syntaxiquement égales

Preuve

- On peut vérifier des *propriétés temporelles*, parlant d'événements dans le futur ("toujours dans le futur" ou "un jour dans le futur") - plus général que les invariants de la preuve à la Hoare
- "Si à un instant n, x $(=x_n)$ est positif, alors il existe un instant $m > n$ tel que pour tous les instants $k \geq m$, y $(=y_k)$ est positif"
- Une approche: propriétés codables en Lustre! (processus "observateur" - cf. JML, model-checking etc.)

Lucid synchrone

Mariage du paradigme fonctionnel, et réactif...

- Outil de preuve par "model-checking" de programmes LUSTRE
- Assertions dans le code, dans une logique un peu plus compliquée que celle du cours 6...

Exemple d'assertions

- Sémantique de `assert(expr)`: doit être vrai à chaque instant, quel que soit le chemin d'exécution
- `assert(false)` : ne doit jamais être atteignable!
- `assert(not(X and Y))` : les flots `X` et `Y` ne doivent jamais être vrais en même temps
- `assert(true->not(X and pre(X)))` : `X` n'a jamais deux valeurs consécutives vraies.

LESAR

- On a un programme que l'on veut vérifier `P(E):...`
- On écrit un noeud "observateur de propriétés" `OBS(E,S):bool`. `OBS` prend le flot `E` d'entrée (le même que pour `P`), prend un flot de sortie (du même type que le flot de sortie de `P`) et renvoie un chaque instant un booléen disant si la propriété à vérifier est vraie ou pas
- On interprète donc le programme LUSTRE `OBS(E,P(E))`
- LESAR prend `OBS` et `P` et essaie de vérifier que l'observateur va toujours rendre vrai (peut se faire partiellement par test également, "vérification dynamique" à la JML)

Exemple

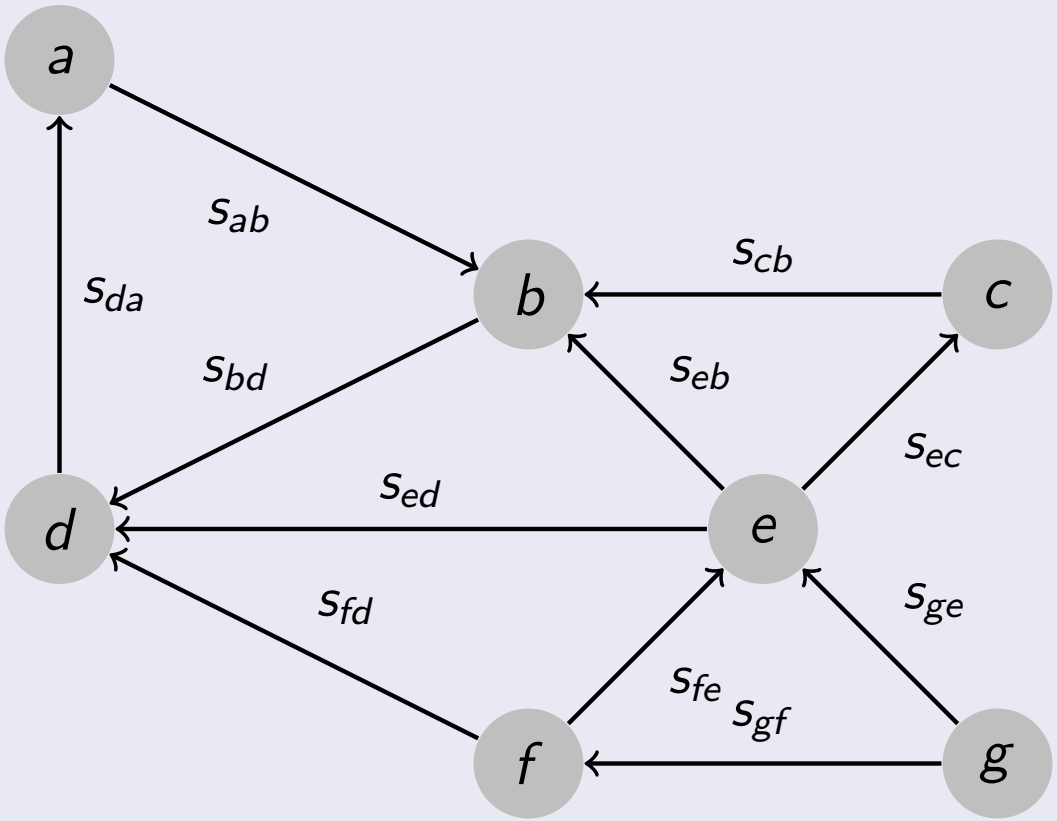
```
node P(E:int) returns (S:bool);
let
  S=if (E=0) then not(pre(S)) else (E>0);
tel

node OBS(E:int, S:bool) returns (B:bool);
let
  assert(E=0);
  B=true->(S xor pre(S));
tel

node RESULT(E:int) returns (B:bool);
let
  B=OBS(E,P(E));
tel
```

de Gilles Kahn (1974)

- Machine théorique formée d'un graphe:
  - dont les noeuds traitent des informations envoyées d'autres noeuds, envoyées par message, par une file non-bornée de messages
  - et renvoient sur les arcs sortant des messages à d'autres noeuds
- Abstrait en quelque sorte l'échantillonage et le traitement discret des données (automatique, traitement du signal etc.)
- Il va falloir imposer une restriction sur le traitement fait par les noeuds pour que cela ait un sens...

Exemple

Fonctions a, b, c, d, e, f et g des arcs entrant; flots s transitant sur les arcs:



DOMAINE SÉMANTIQUE

- Le domaine des données $\mathcal{S}$ est celui des *suites* de valeurs (dans Val) finies $(x_0, \ldots, x_n)$ ou pas $(x_0, \ldots, x_n, \ldots)$.
- On identifiera la suite finie $(x_0, \ldots, x_n)$ avec la suite infinie à valeur dans $\mathsf{Val} \cup \{\bot\}$: $(x_0, \ldots, x_n, \bot, \ldots, \bot, \ldots)$ donc $\mathcal{S} = \{x : \mathbb{N} \to \mathsf{Val}_\bot \mid x_i = \bot \Rightarrow (\forall j \geq i, x_j = \bot)\}$

Ordre partiel

- On définit l'ordre partiel *préfixe* sur $\mathcal{S}$ par, pour $x, y \in \mathcal{S}$, $x \leq y$ si:

$$x_i \neq \bot \Rightarrow y_i = x_i$$

- Dit de façon plus simple, x est un préfixe de y; et l'ordre préfixe est la restriction à $\mathcal{S}$ de l'ordre défini au cours 8 pour $\mathbb{N} \to \mathrm{Val}_\bot$ ($\mathrm{Val}_\bot$ étant un CPO)!
- $\mathcal{S}$ est donc un CPO! (vérification triviale)

Exemple pour l'ordre préfixe

- $\bot \leq (0, 1, 2, \ldots)$
- $(0) \leq (0, 1) \leq (0, 1, 2) \leq \ldots \leq (0, 1, 2, \ldots)$

Prérequis

- En quelque sorte, on veut qu'elles soient calculables
- On impose la continuité! (cf. cours 8)
- On impose pour $f : \mathcal{S}^n \to \mathcal{S}^m$ la croissance et la commutation aux *sup*; celle-ci peut s'imposer coordonnée par coordonnée (on suppose ici $m = n = 1$): pour toute ω-chaîne $x^0 \leq x^1 \leq \ldots \leq x^j \leq \ldots$ de $\mathcal{S}$,

$$f\left(\bigcup_{j\in\mathbb{N}} x^j\right) = \bigcup_{j\in\mathbb{N}} f(x^j)$$

Pour toute ω-chaîne $x^0 \leq x^1 \leq \ldots \leq x^n \leq \ldots$ on a, pour tout $j \in \mathbb{N}$:

$$\begin{aligned}
\left(\bigcup_{i\in\mathbb{N}} f(x^i)\right)_j &= \begin{cases} f(x^k)_j & \exists k \in \mathbb{N},\ f(x^k)_j \neq \bot \\ & \text{et } \forall l \geq k,\ f(x^l)_j = f(x^k)_j \\ \bot & \text{sinon} \end{cases} \\
&= \\
f\left(\bigcup_{i\in\mathbb{N}} x^i\right)_j &= f\left(y \to \begin{cases} x_j^k & \exists k' \in \mathbb{N},\ x_j^{k'} \neq \bot \\ \bot & \text{sinon} \end{cases}\right)(x)
\end{aligned}$$

Intuitivement

- Pour j fixé, la $j$ième valeur du flot de sortie de f est déterminée par l'image par f sur un préfixe fini du flot d'entrée
- Sorte d'axiome de "causes finies"!

Exemple de fonction non continue sur $\mathcal{S}$

Soit $g : \mathcal{S} \to \mathcal{S}$ telle que:

$$g(x) = \begin{cases} (0, \ldots, 0, \ldots) & \text{si } x \text{ est fini} \\ (1, \ldots, 1, \ldots) & \text{si } x \text{ est infini} \end{cases}$$

Soit y flot infini et y^i, $i = 0, 1, \ldots$, tous ses préfixes finis:
$\bigcup_{i \in \mathbb{N}} y^i = y$ mais $f(\bigcup_{i \in \mathbb{N}} y^i) = (1, \ldots)$ et
$\bigcup_{i \in \mathbb{N}} g(y^i) = \bigcup_{i \in \mathbb{N}} (0, \ldots) = (0, \ldots)$!

Principe

- Chaque noeud N est une fonction continue

$$f^N : \mathcal{S}^n \to \mathcal{S}^m$$

- On écrit une équation de point fixe, décomposée sur chaque noeud N, pour tous les m arcs sortants de N vers M_1 à M_m:

$$\begin{cases} X_{M_1} &= f_1^N(X_{L_1}, \ldots, X_{L_n}) \\ \ldots &= \ldots \\ X_{M_m} &= f_n^N(X_{L_1}, \ldots, X_{L_n}) \end{cases}$$

  Les $X_{L_1}, \ldots, X_{L_n}$ vers N sont les n arcs entrants en N

- Par Kleene, on a un plus petit point fixe (tous les f_i^N sont continues sur un CPO): sémantique (dans les suite infinies de valeurs) du réseau!

Avant de partir en vacances...

Vous aurez bientôt de vrais choix à faire...

- En fin de deuxième année: choisir un PA
- Puis choisir un stage de recherche...
- Puis choisir sa quatrième année
- Continuer après la quatrième année? (recherche ou pas recherche?)

La formation universitaire en général...

Réforme européenne LMD

Produit du *processus de Bologne* (1999):

- L: "license" 3 ans(+), prépa plus année 2 de l'X
- M: "master" 2 ans, PA (Informatique, Conception et Management de Systèmes Informatiques Complexes, Electrical Engineering, Bio-informatique etc.) et stage et quatrième année
- D: "doctorat" 3 ans(+), en laboratoire ou mixte laboratoire/industrie

Corps, écoles d'applis, master à l'etranger?

- Corps: 2/3 ans, essentiellement un MPA ("Master of Public Administration") plus corps de fonctionnaire
- Ecoles d'application, 1 an fin de M2 (Télécom, Ensimag, Supélec etc.)
- Master en France (MPRI, COMASIC etc.), Master à

Surtout...

Commencer à réfléchir...

- Parlez à vos enseignants...
- Profitez des stages
- Visitez des entreprises
- Parlez avec vos anciens
- Allez voir les conférences multiples d'anciens etc.

Mais faites un choix avant d'être "choisi"!

Cohérence d'un projet professionnel!

- Industrie, recherche ou administration?
- Après un M ou un D?

C'est tout pour aujourd'hui..

La prochaine fois

- Pâle!!! (30 juin)

Bon TD!