INF 321
Paradigme réactif synchrone

Eric Goubault

Cours 10

29 juin 2012

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:

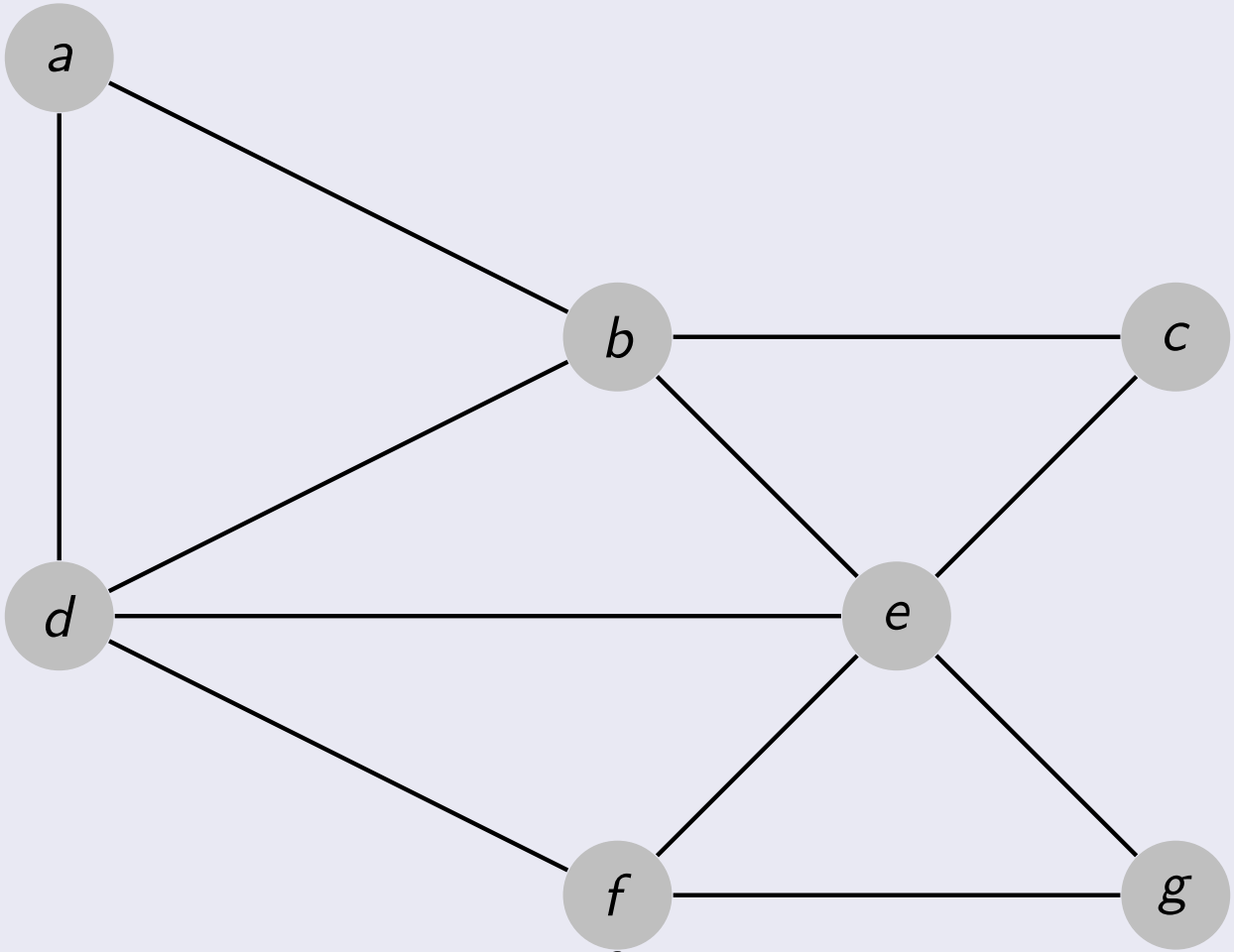
- Un peu de graphes (pour le TD, et pour introduire les réseaux de Kahn)
- Les langages synchrones, réseaux de Kahn
- LUSTRE et la programmation réactive
- Quelques conseils avant les vacances

Pourquoi des graphes?

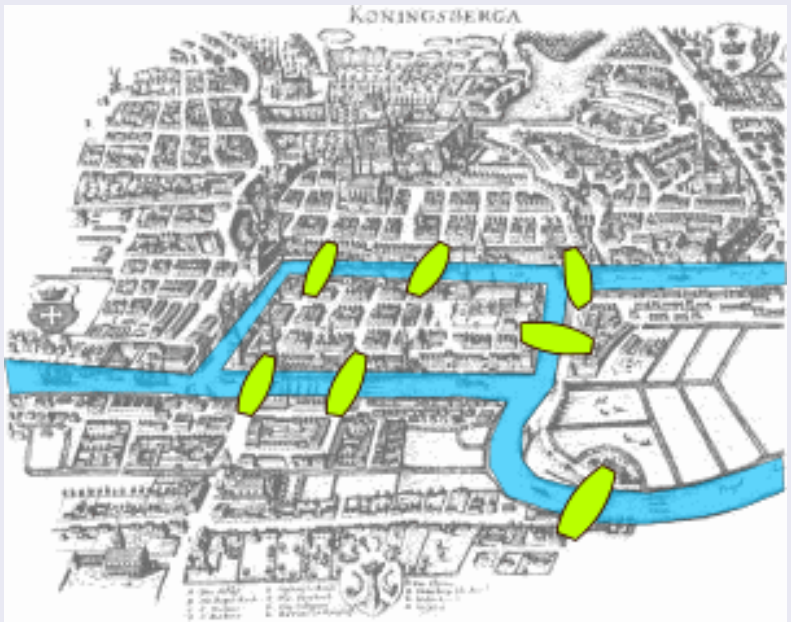
Vous avez vu les arbres en taupe...

On pousse l'idée du partage au maximum...

Exemple, représentation graphique



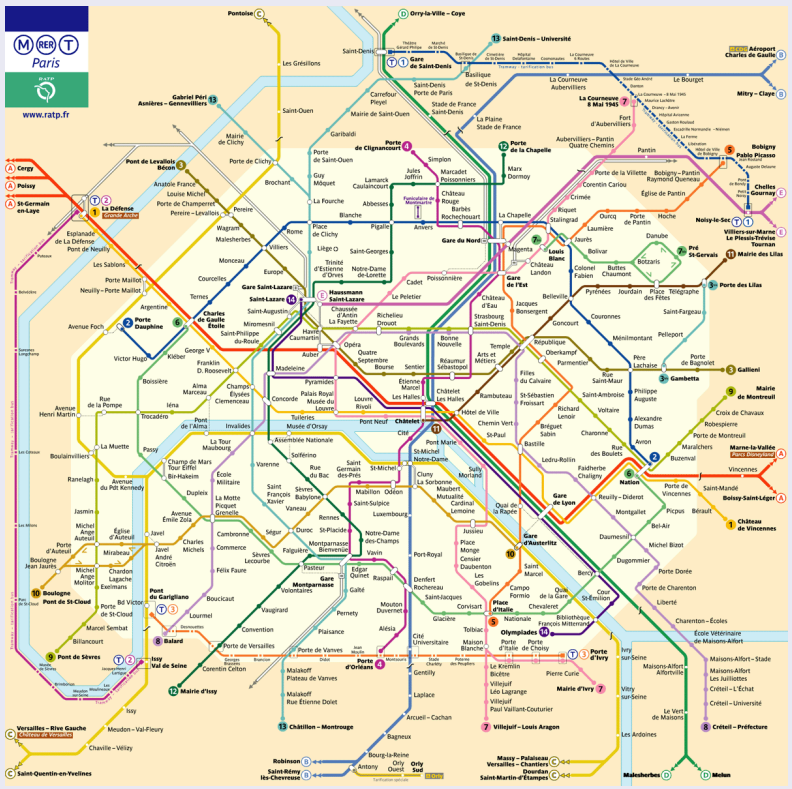
Les 7 ponts de Königsberg (Euler, 1735)



...pas de promenade possible en passant une et une seule fois par chaque pont (*graphe Eulérien*)

Pourquoi des graphes?

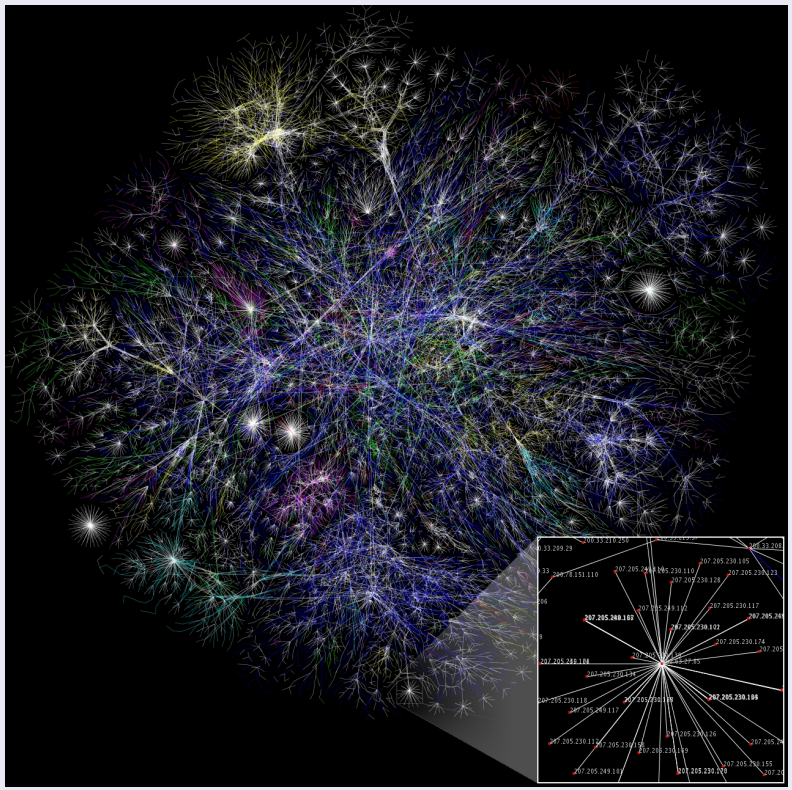
En un peu plus compliqué...réseau de transport



Modélisation de réseaux de distribution

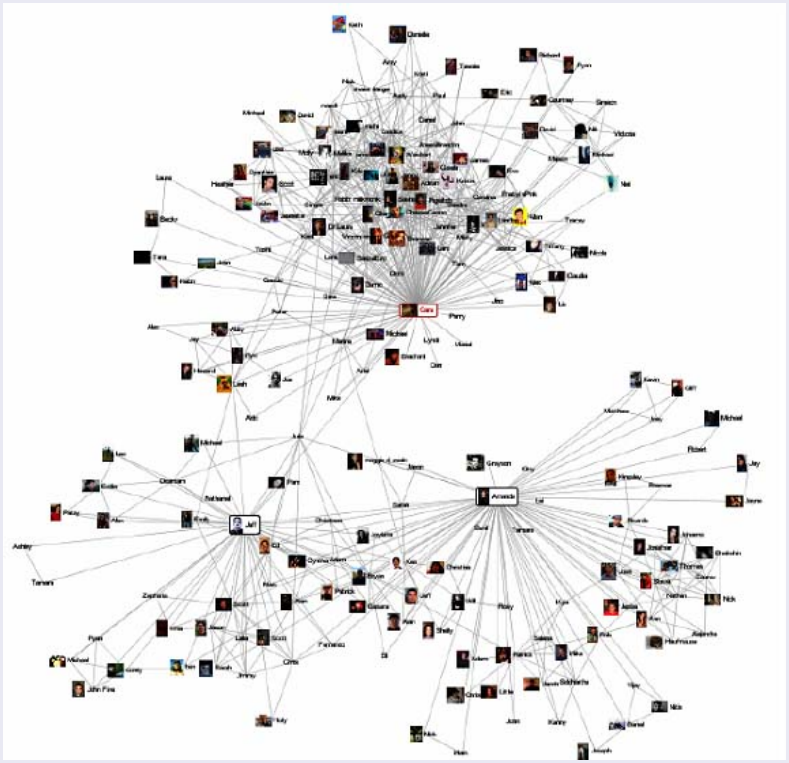
...électriques par exemple

Modélisation de réseaux



(internet)

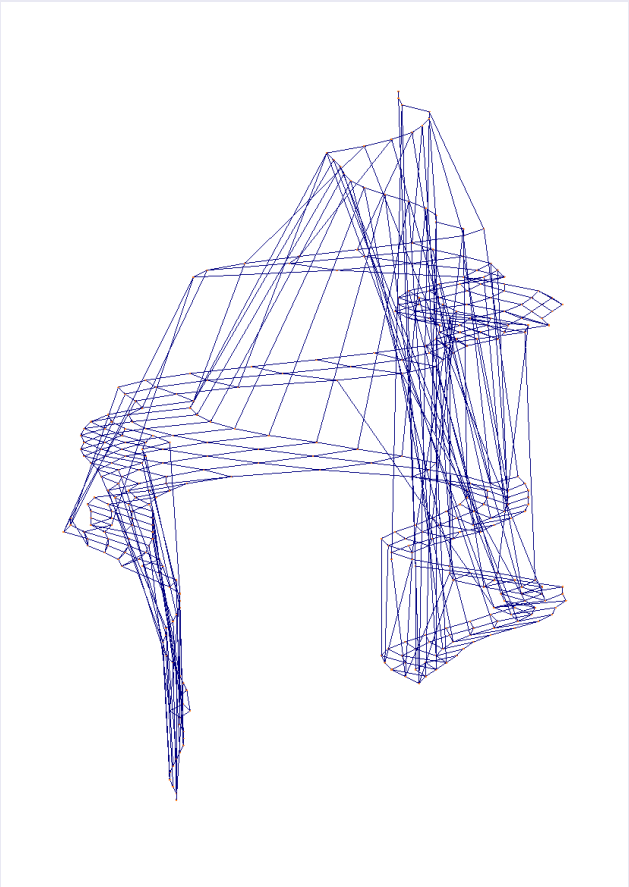
Modélisation de relations



Ici, réseau social... plus généralement, relations de dépendances (de tâches par exemple)

Pourquoi des graphes?

Sémantique de langages de programmation...

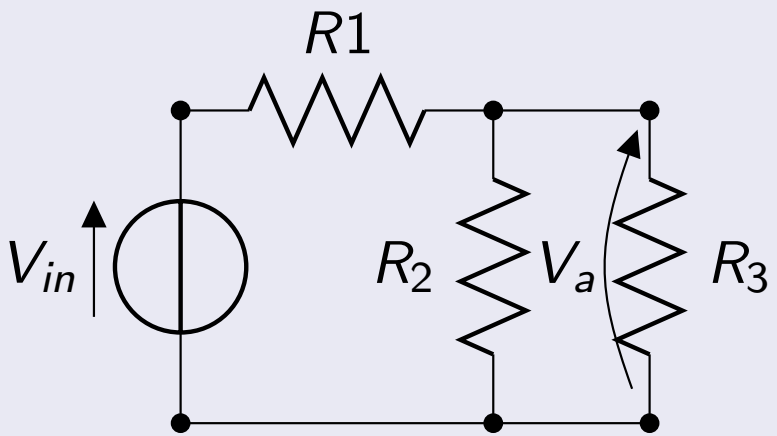


(ici, algorithme d'exclusion mutuelle de Dekker, entre 2 processus)
$\longrightarrow$ systèmes dynamiques discrets

Etc.

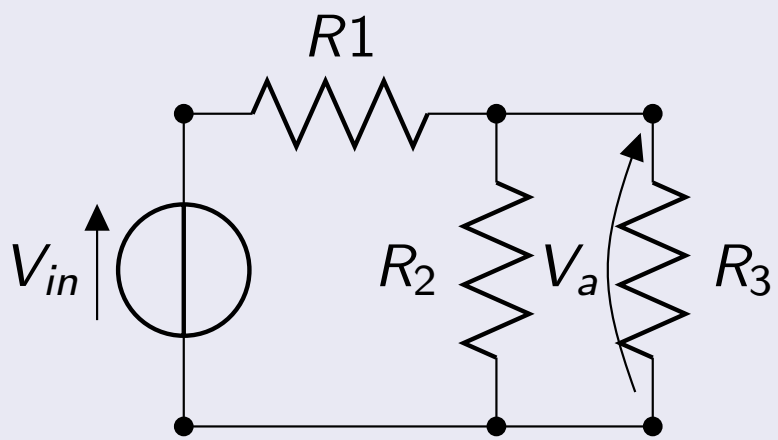
Vous en avez déjà manipulé au lycée...

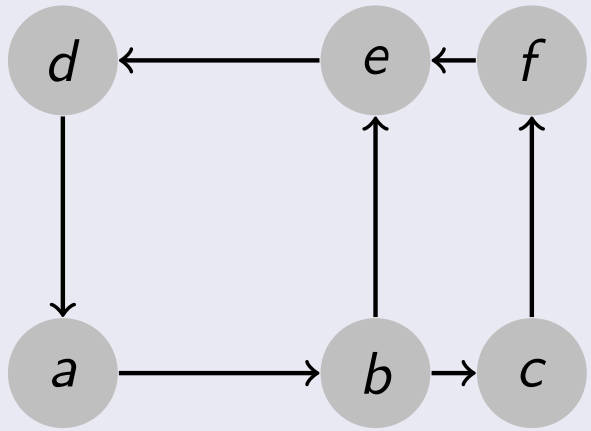
Circuits électriques!



Vous en avez déjà manipulé au lycée...

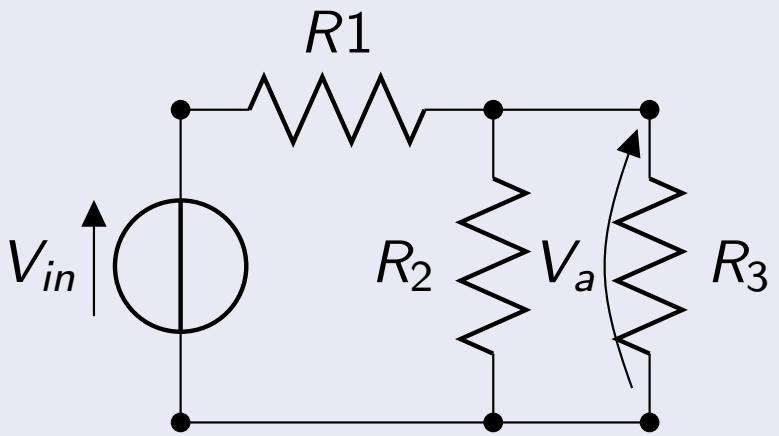
Circuits électriques!





Les lois de Kirschoff sont liées au graphe sous-jacent...

Circuits électriques!



$$V_a = \frac{R_2 R_3}{R_1 R_2 + R_1 R_3 + R_2 R_3} V_{in}$$

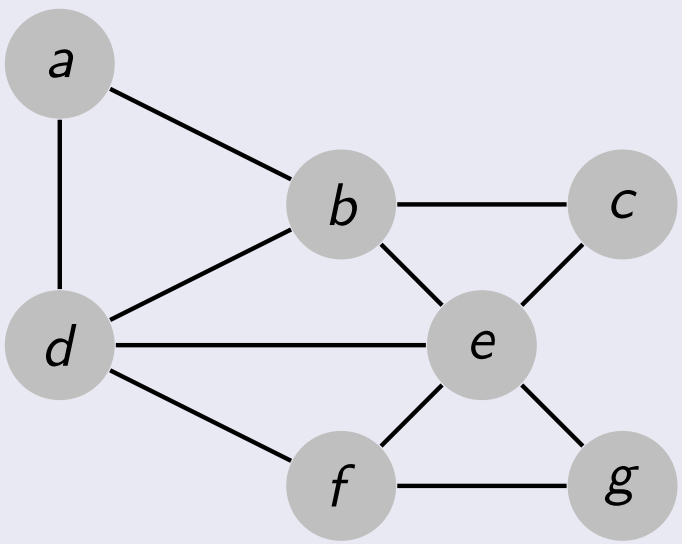
Définition

Un graphe est constitué de:

- noeuds ou sommets (N)
- arêtes (=paire de noeuds, E)

On écrit $G = (N, E)$, $E \subseteq N \times N$

Dans l'exemple précédent:



- $N = \{a, b, c, d, e, f, g\}$
- $E = \{(a, b), (a, d), (b, d), (b, c), (b, e), (c, e), (e, f), (e, d), (e, g), (f, d), (f, g)\}$

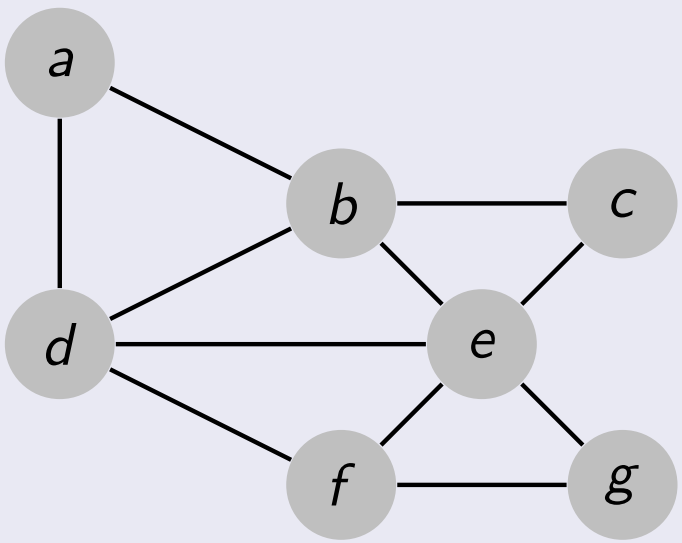
Définition

Un graphe est constitué de:

- noeuds ou sommets (N)
- arêtes (=paire de noeuds, E)

On écrit $G = (N, E)$, $E \subseteq N \times N$

Dans l'exemple précédent:



- $N = \{a, b, c, d, e, f, g\}$
- $E = \{(\mathbf{a}, \mathbf{b}), (a, d), (b, d), (b, c), (b, e), (c, e), (e, f), (e, d), (e, g), (f, d), (f, g)\}$

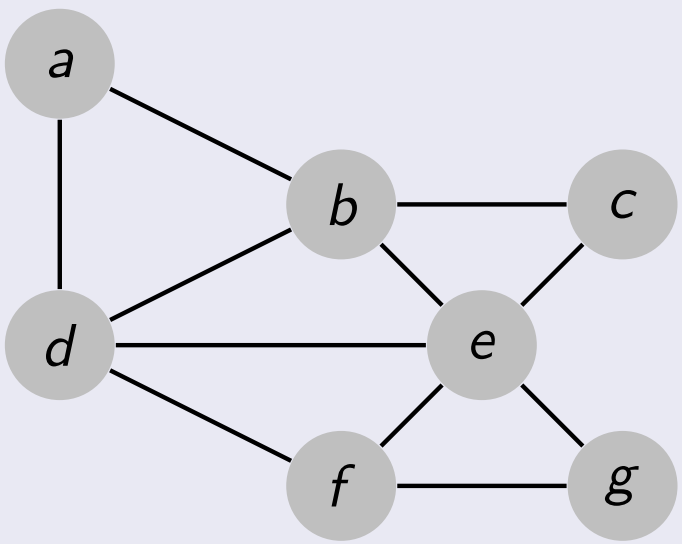
Définition

Un graphe est constitué de:

- noeuds ou sommets (N)
- arêtes (=paire de noeuds, E)

On écrit $G = (N, E)$, $E \subseteq N \times N$

Dans l'exemple précédent:



- $N = \{a, b, c, d, e, f, g\}$
- $E =$
  $\{(a, b), (\mathbf{a}, \mathbf{d}), (b, d), (b, c),$
  $(b, e), (c, e), (e, f), (e, d),$
  $(e, g), (f, d), (f, g)\}$

Représentation par matrice d'adjacence

Matrice d'adjacence

Supposons que G, graphe, contienne p noeuds numérotés $n_1, \ldots, n_p$, on construit une matrice M de taille $p \times p$ avec:

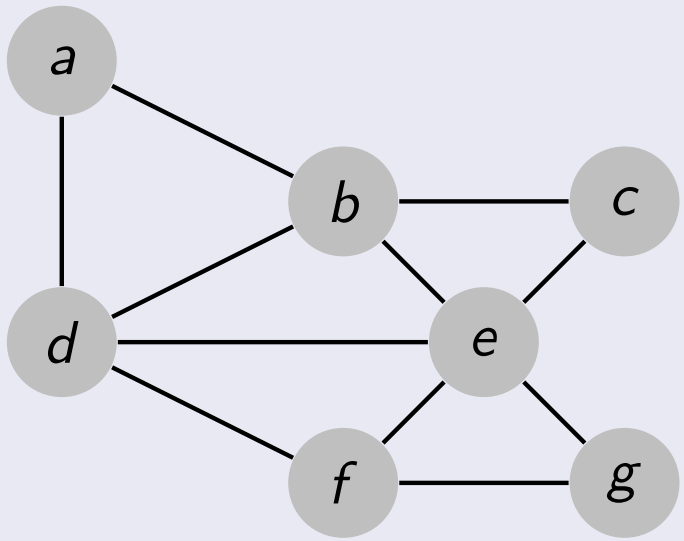
$$M_{i,j} = \begin{cases} 1 & \text{si } (n_i, n_j) \in E \\ 0 & \text{sinon} \end{cases}$$

Implémentation Java

```
class Graph {
  int p;
  int[][] M;
  Graph(int n) {
    p = n;
    M = new int[n][n];
  }
}
```

Exemple de matrice d'adjacence

Graphe



$$M = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

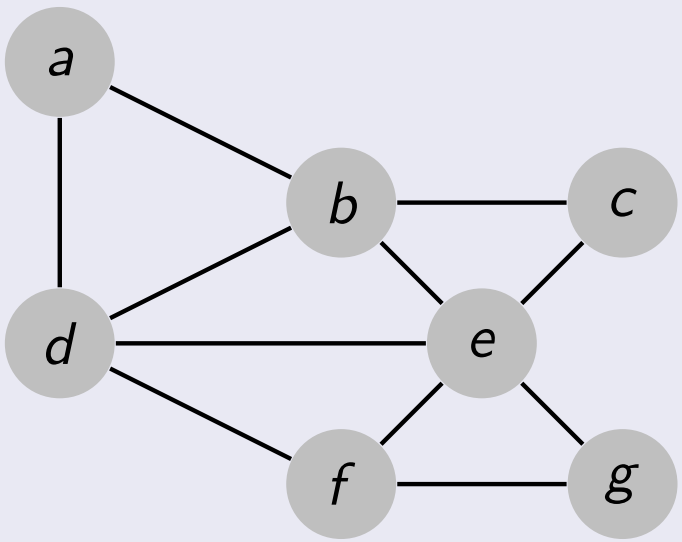
(en numérotant de 1 à 7, a, b, c, d, e, f, g resp.)

Avantages

- Certains calculs sont faciles, ex. l'existence de chemins de longueur n par exemple, grâce au théorème suivant:

$$M^n_{i,j} = 1 \text{ ssi } \exists \text{ chemin de } i \text{ à } j \text{ de longueur } n$$

Exemple



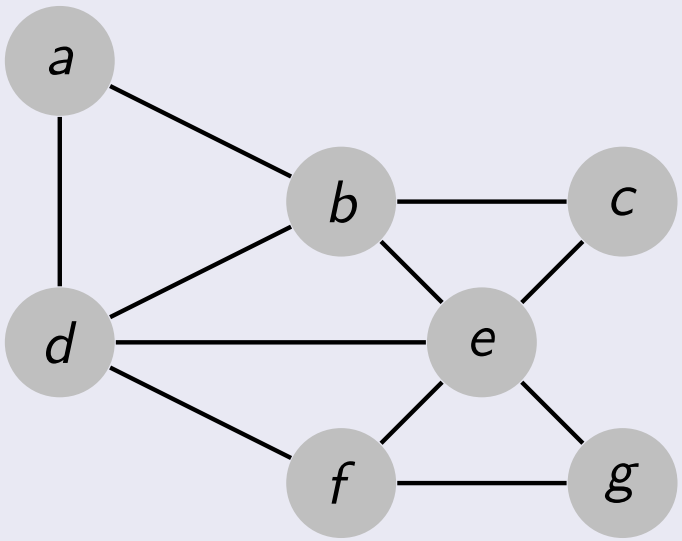
$$M^2 = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Avantages

- Certains calculs sont faciles, ex. l'existence de chemins de longueur n par exemple, grâce au théorème suivant:

$$M^n_{i,j} = 1 \text{ ssi } \exists \text{ chemin de } i \text{ à } j \text{ de longueur } n$$

Exemple



$$M^2 = \begin{pmatrix} 1 & 1 & 1 & 1 & \mathbf{1} & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Inconvénients

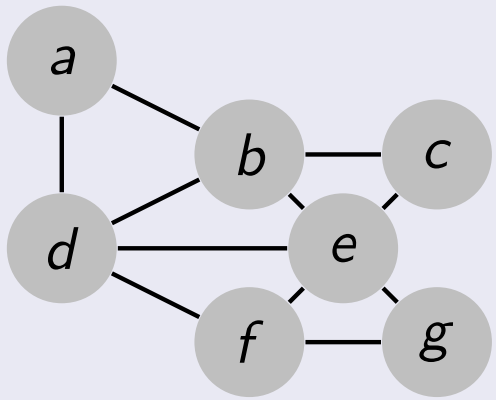
- Représentation peu économique!
- Il vaut mieux utiliser une représentation *creuse* de la matrice (ne représenter que les 1 et pas les 0)

En Java, représentation creuse de matrice d'adjacence

```
class Graphcreux {
  int p;
  List[] M;
  Graphcreux(int n) {
    p = n;
    M = new List[n];
  }
}
```

Autre représentation: matrice d'incidence, un exemple

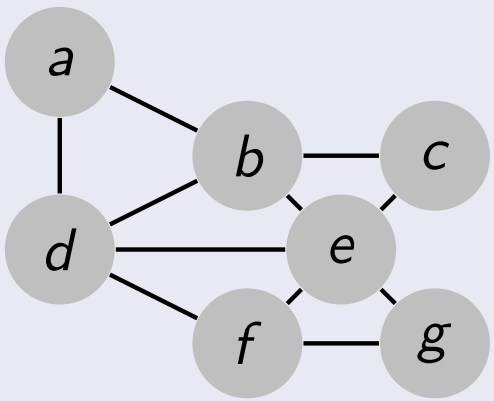
Matrice arêtes/sommets:



	a	b	c	d	e	f	g
(a,b)	1	-1	0	0	0	0	0
(a,d)	1	0	0	-1	0	0	0
(b,d)	0	1	0	-1	0	0	0
(b,c)	0	1	-1	0	0	0	0
(b,e)	0	1	0	0	-1	0	0
(d,f)	0	0	0	1	0	-1	0
(e,c)	0	0	-1	0	1	0	0
(e,d)	0	0	0	-1	1	0	0
(e,f)	0	0	0	0	1	-1	0
(e,g)	0	0	0	0	1	0	-1
(f,g)	0	0	0	0	0	1	-1

Représentation par matrice d'incidence - exemple

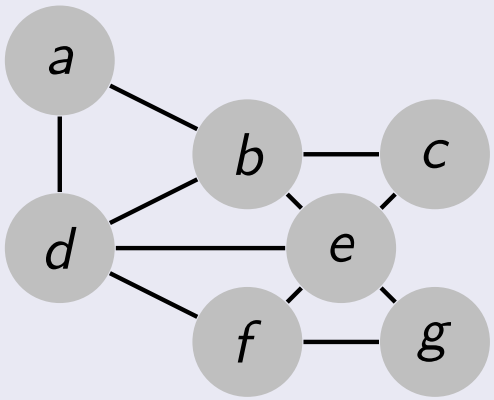
Graphe



	a	b	c	d	e	f	g
$(\mathbf{a}, \mathbf{b})$	**1**	**−1**	0	0	0	0	0
(a, d)	1	0	0	−1	0	0	0
(b, d)	0	1	0	−1	0	0	0
(b, c)	0	1	−1	0	0	0	0
(b, e)	0	1	0	0	−1	0	0
(d, f)	0	0	0	1	0	−1	0
(e, c)	0	0	−1	0	1	0	0
(e, d)	0	0	0	−1	1	0	0
(e, f)	0	0	0	0	1	−1	0
(e, g)	0	0	0	0	1	0	−1
(f, g)	0	0	0	0	0	1	−1

Représentation par matrice d'incidence - exemple

Graphe



	a	b	c	d	e	f	g
(a,b)	1	-1	0	0	0	0	0
(a,d)	**1**	0	0	**-1**	0	0	0
(b,d)	0	1	0	-1	0	0	0
(b,c)	0	1	-1	0	0	0	0
(b,e)	0	1	0	0	-1	0	0
(d,f)	0	0	0	1	0	-1	0
(e,c)	0	0	-1	0	1	0	0
(e,d)	0	0	0	-1	1	0	0
(e,f)	0	0	0	0	1	-1	0
(e,g)	0	0	0	0	1	0	-1
(f,g)	0	0	0	0	0	1	-1

Extensions: matroïdes algébriques...Plus en INF 550!

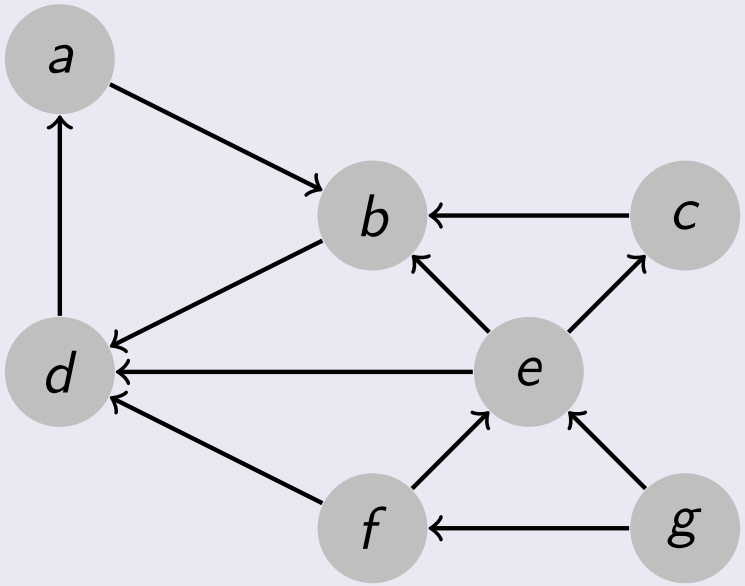
Définition

Un graphe orienté est la donnée de:

- N un ensemble de noeuds
- A un ensemble d'*arcs*, $A \subseteq N \times N$

La différence par rapport aux graphes non-orientés est qu'un arc (n_i, n_j) est orienté de n_i vers n_j

Représentation graphique



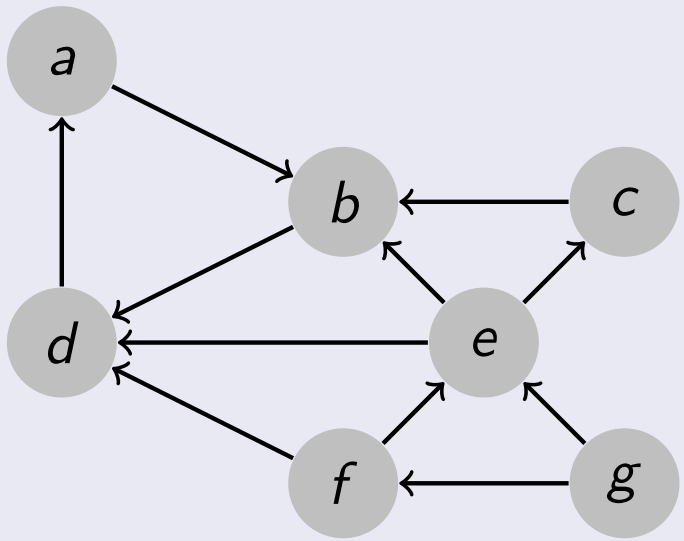
On appelle *carquois* un graphe orienté...

...car il y a des arcs et des flèches ;-)

Matrices d'adjacence

Cette fois-ci, $M_{i,j} = 1$ ssi il existe un arc de i *vers* j. M n'est plus nécessairement symétrique

Exemple



$$M = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

(en numérotant de 1 à 7, a, b, c, d, e, f, g resp.)

Implémentation Java

```
class Graphmat {
  int p;
  int [][] M;
```

Egalement représentation creuse similaire...

Quelques notions utiles...

Chemins, cycles...

- Un *chemin* (orienté) est une suite de noeuds connectés par des arcs, donc une suite $(n_{i_1}, n_{i_2}), (n_{i_2}, n_{i_3}), \ldots, (n_{i_k}, n_{i_{k+1}})$
- Un *circuit* (ou *cycle*) est un chemin (orienté) dont le début est égal à la fin

Et composantes "fortement connexes"

- On peut partitionner l'ensemble des noeuds d'un graphe orienté en *composantes fortement connexes*: x, et y dans la même composante $\Rightarrow$ x et y appartiennent à un cycle (notion d'interdépendance)
- Il y a moyen de modifier un peu le parcours en profondeur d'abord, des arbres, pour les graphes (DFS), pour déterminer les cycles d'un graphe, et un peu plus...
- Algorithme de Tarjan par exemple
- Plus en INF431!!

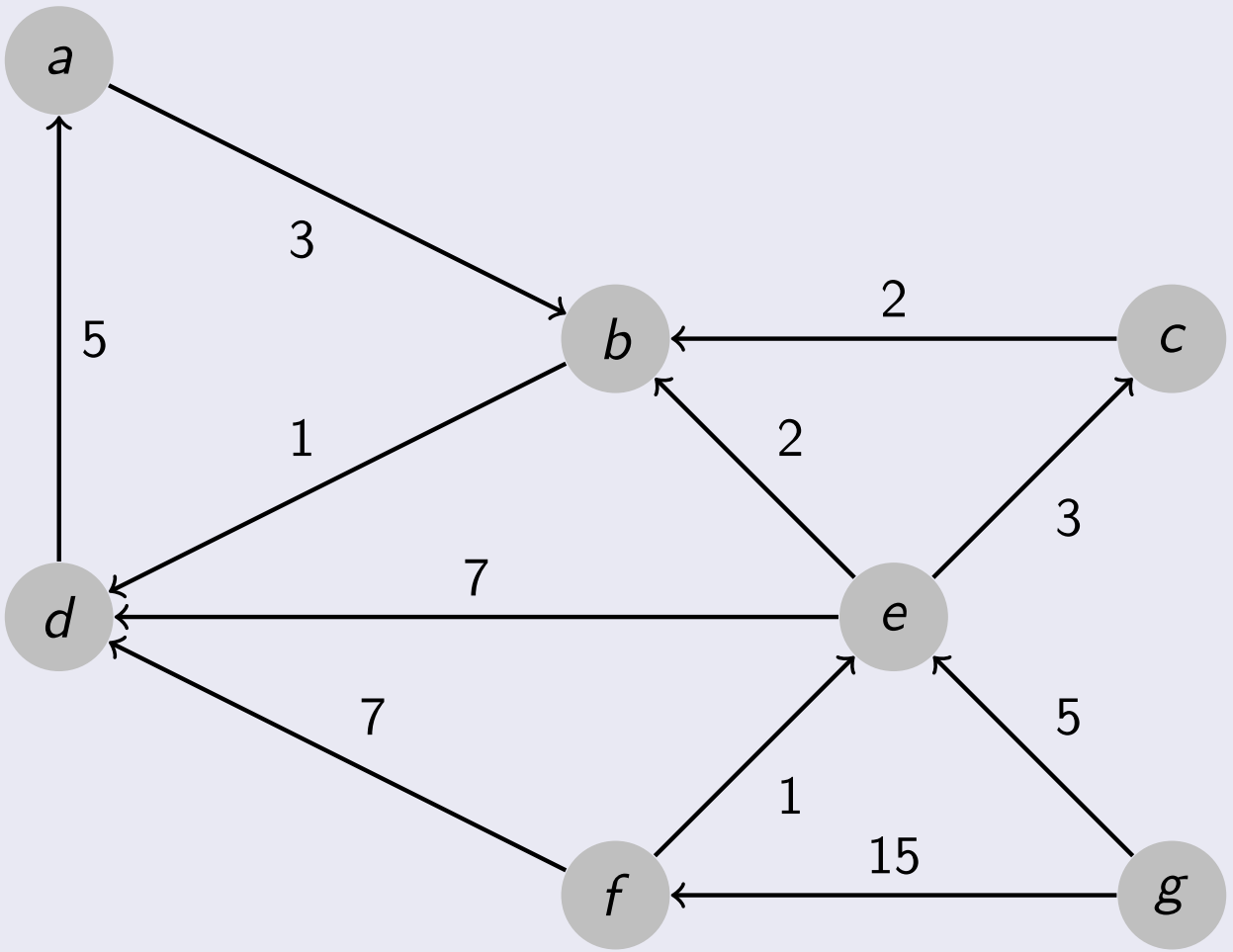
Valuation?

- Une généralisation bien utile!
- Chaque arc possède une valuation (valeur entière ou réelle typiquement); on passe ainsi d'une matrice d'adjacence à coefficients dans $\{0, 1\}$ à des matrices générales à coefficients dans $\mathbb{N}$ ou $\mathbb{R}$!

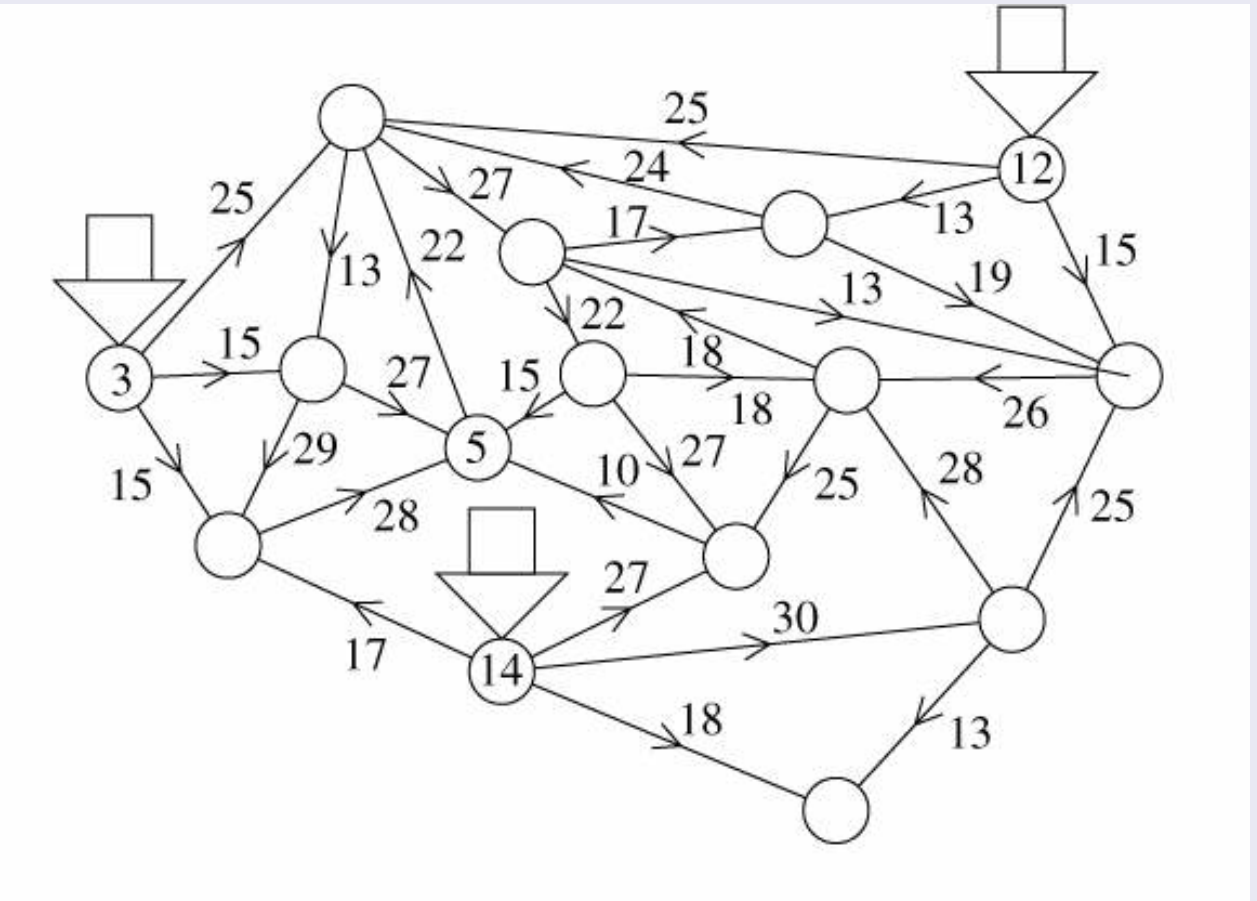
Implémentation Java

Toujours la même!

Exemple



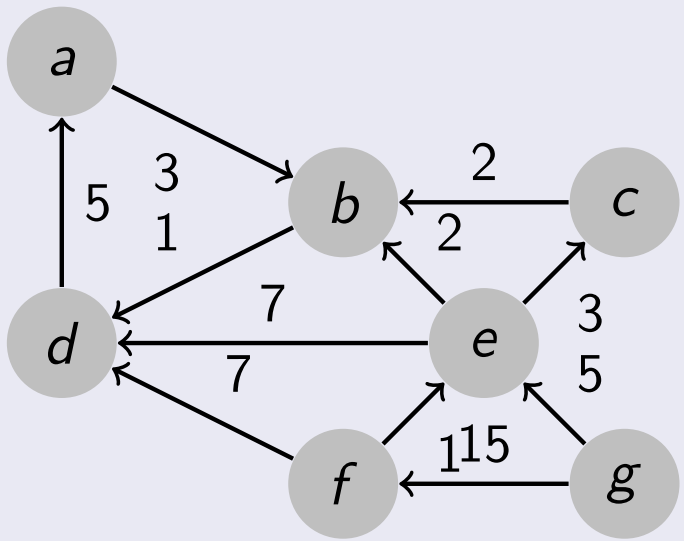
Capacité de transport électrique



Rapport avec l'algèbre linéaire...

- Très profond... début de la théorie des matroïdes (Whitney, 1935)

Notre exemple



$$M = \begin{pmatrix} 0 & 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 3 & 7 & 0 & 0 & 0 \\ 0 & 0 & 0 & 7 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 15 & 0 \end{pmatrix}$$

Rapport avec l'algèbre linéaire...

- Une théorie profonde: Perron-Frobenius, "parle" des matrices à coefficients positifs, de leur décomposition spectrale etc.
- Utilisé pour le classement des pages web google!:
  - Graphe G (monstrueux!) où les noeuds sont les pages web accessibles
  - Où on a un arc d'un noeud vers un autre si la première page web pointe sur la seconde...
  - On peut associer une valuation (nombre d'accès à la deuxième page en passant par la première)
- Le rayon spectral de M donne une bonne indication des pages les plus référencées

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

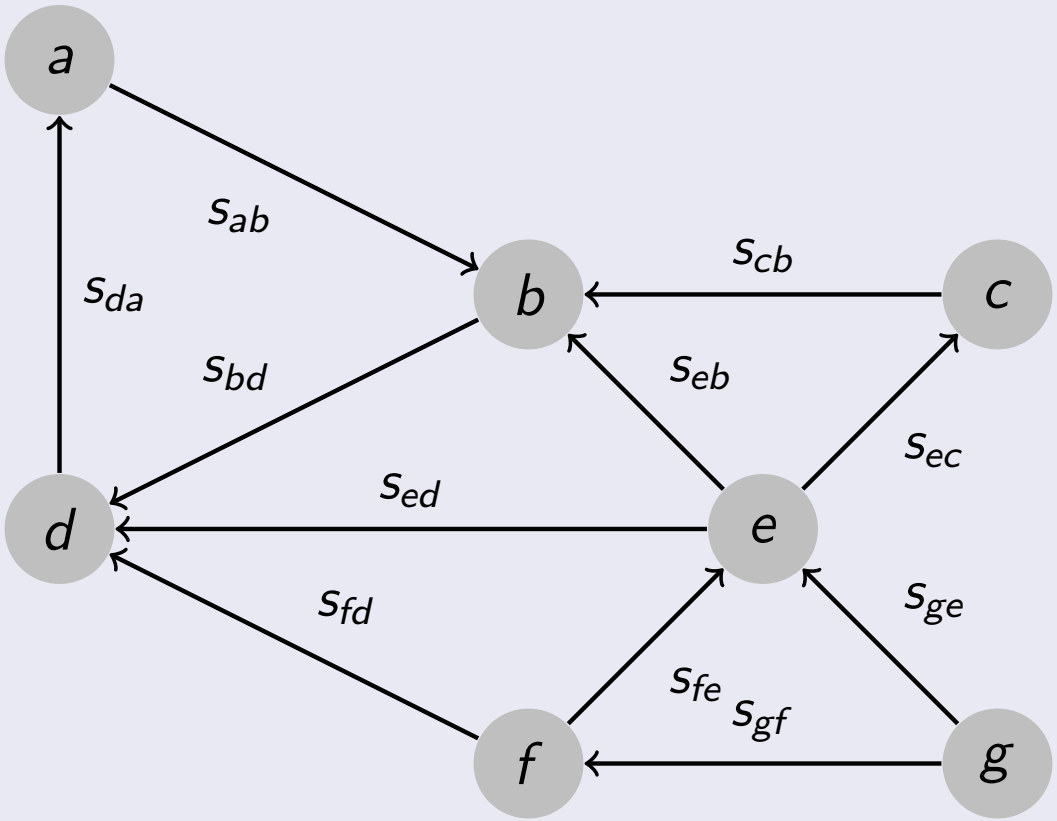
- Versions académique LUSTRE, industrielle SCADE (Esterel Technologies)
- Nombreux outils, de preuve, test etc.
- Belle sémantique et utilisé couramment dans le contrôle commande (Airbus, Siemens etc.)!

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 7 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 7)
- On se contente d'imposer pour $f : \mathcal{S}^n \to \mathcal{S}^m$ 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 $\mathcal{S}$,

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

(comme $f(x^j)$ n'est pas nécessairement croissante, il faut supposer qu'il existe un *sup* de cette suite, égale au terme de gauche, dans cette définition)

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
$\bigcup_{i \in \mathbb{N}} g(y^i) = \bigcup_{i \in \mathbb{N}} (0, \ldots) = (0, \ldots)$!

Continuité et croissance

A l'origine, Kahn demandait juste cette préservation des bornes supérieures. Quid de la croissance (=causalité!)?

Cela implique la croissance!

- Supposons que l'on ait une fonction $f : \mathcal{S} \to \mathcal{S}$ telle que pour toute ω-chaîne $x^0 \leq x^1 \leq \ldots \leq x^j \leq \ldots$,

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

- Soit, pour $x \leq y$, la suite $X = (x^0 = x, x^1 = y, \ldots, x^n = y, \ldots) \in \mathcal{S}$, alors $f\left(\bigcup_{j\in\mathbb{N}} x^j\right) = f(y)$ mais $\bigcup_{j\in\mathbb{N}} f(x^j) = Z$ est tel que $f(x) \leq Z = f(y)$ par hypothèse; CQFD

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:

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

  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!

Maintenant on va voir en pratique à quoi cela ressemble, dans LUSTRE...

Réseau de Kahn synchrone!

- 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
- 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 $(x_0 + y_0, x_1 + y_1, \ldots, x_n + y_n, \ldots)$

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 $(x_0, y_1, \ldots, y_n, \ldots)$

Remarque: les flots sont typés

`bool` est le type des flots de booléens etc.

Expressions arithmétiques et booléennes

Les mêmes que d'habitude, mais étendues point à point aux flots

Affectations, conditionnelles, séquence...

- Affectation: `let ... = ... tel`
- Conditionnelles: `if ... then ... else`
- Séquence:

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

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

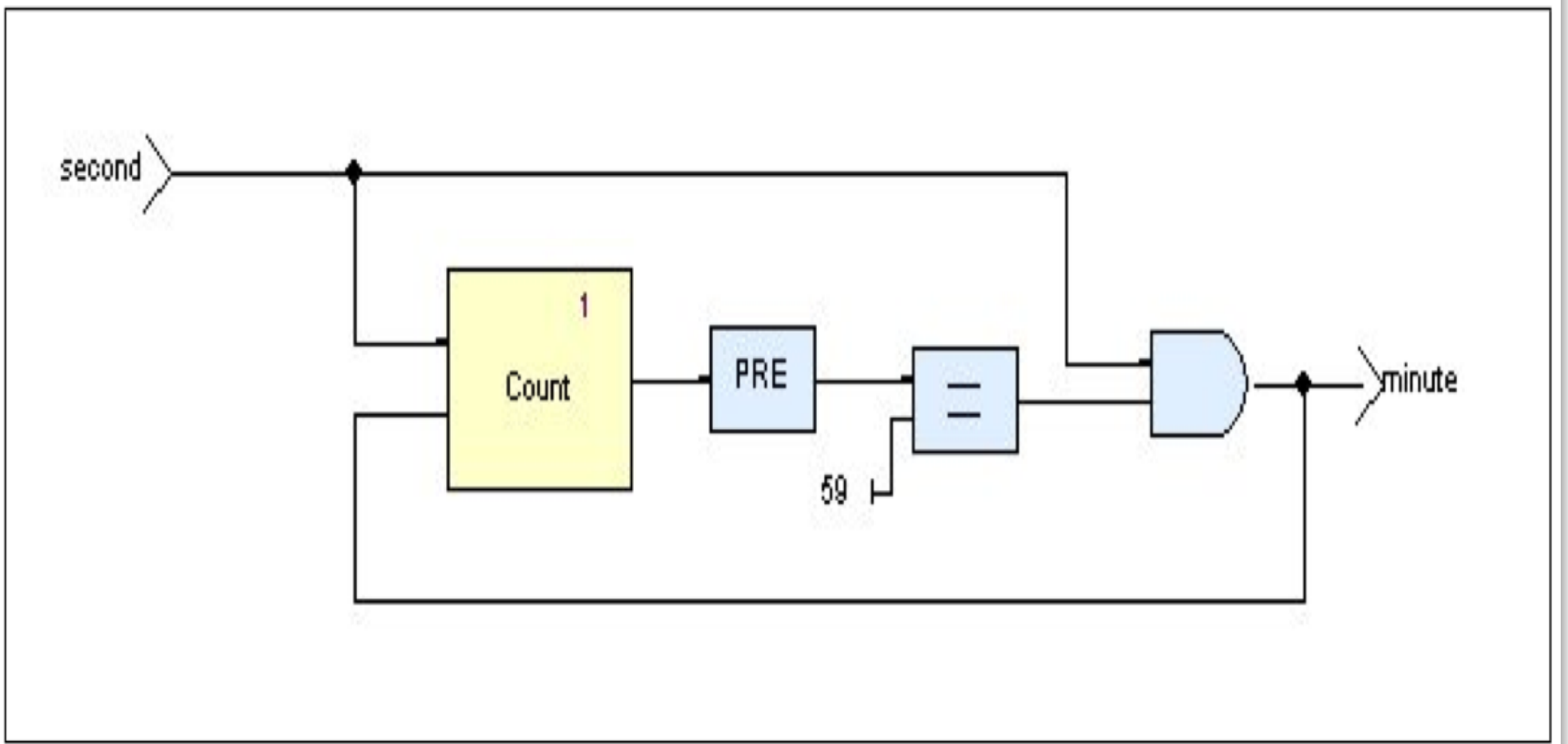
Compter les minutes et secondes

```
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

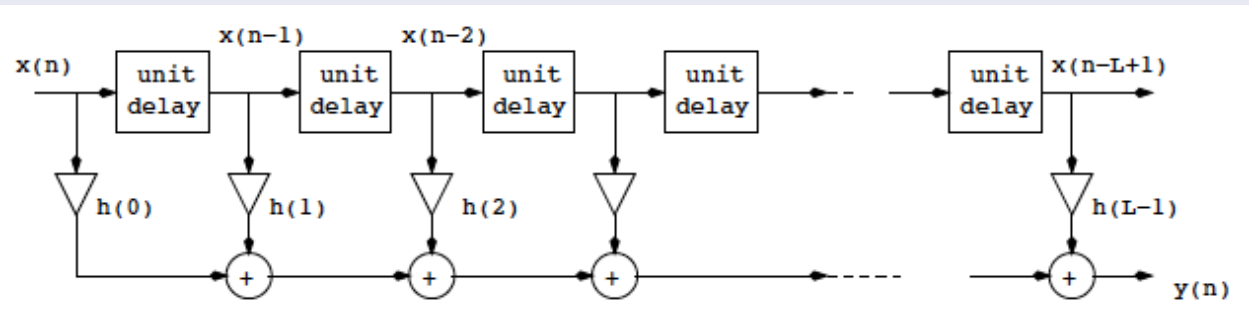


Filtres linéaires à réponse finie

- 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



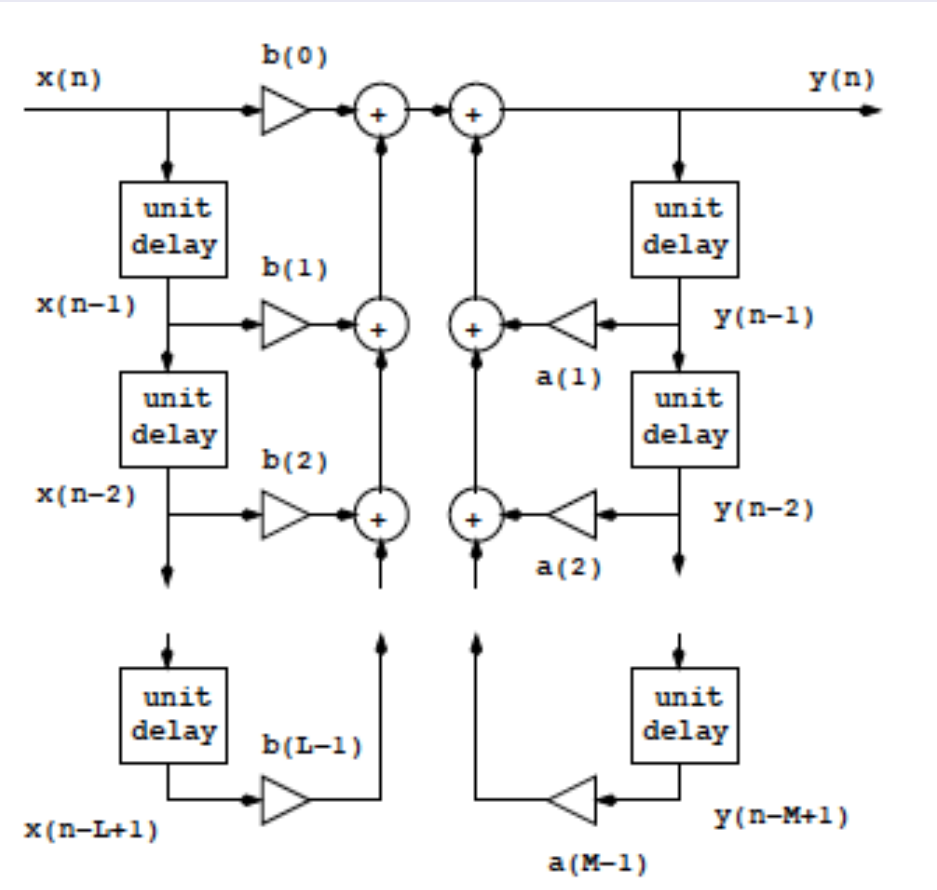
Autre exemple: IIR

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;
```

"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!)

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;
```

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" - model-checking etc.)

Lucid synchrone

Mariage du paradigme fonctionnel, et réactif...

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!!! (12 juillet)

Bon TD!