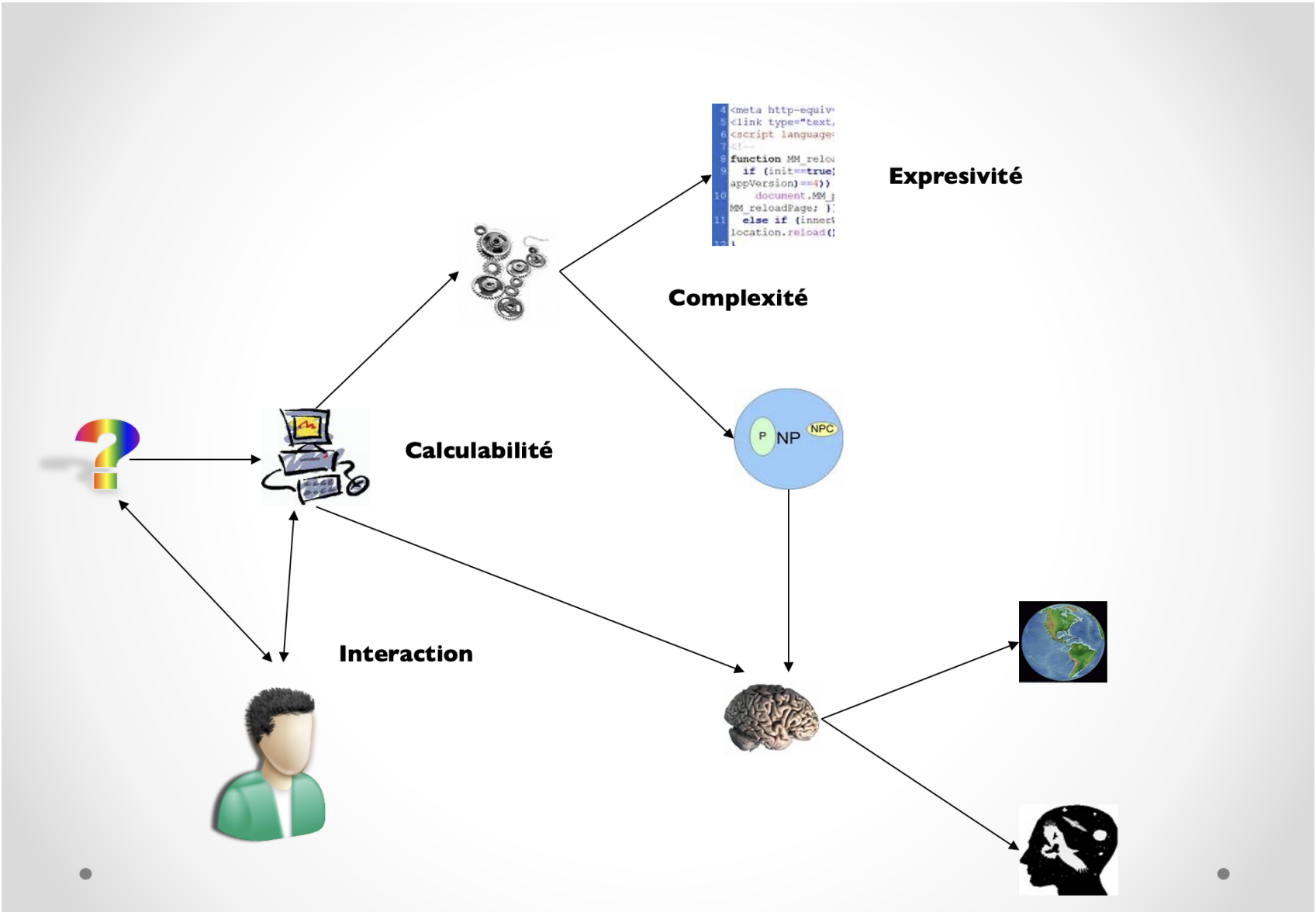
Master ATIAM

Cours 5
Interaction

Carlos Agon (agonc@ircam.fr)

22 septembre 2023

Interaction

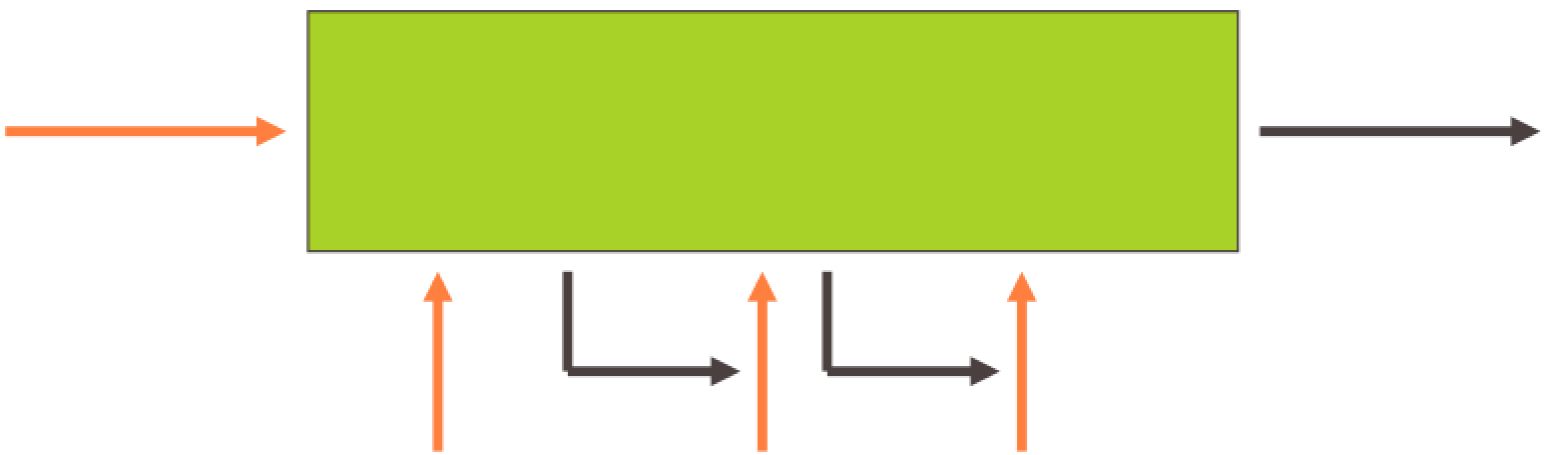


Interaction

Un système interactif est une application informatique qui prend en compte, au cours de son exécution, d'informations communiquées par le ou les utilisateurs du système, et qui produit, au cours de son exécution, une représentation perceptible de son état interne.

Interaction

Un système interactif est une application informatique qui prend en compte, au cours de son exécution, d'informations communiquées par le ou les utilisateurs du système, et qui produit, au cours de son exécution, une représentation perceptible de son état interne.

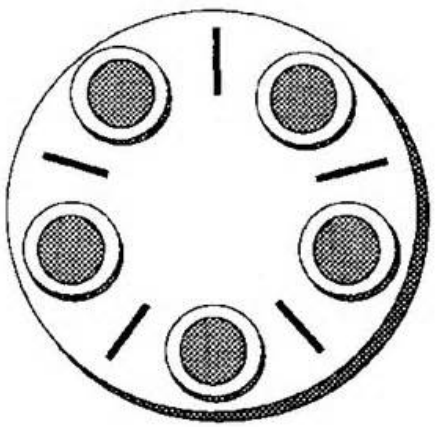


Historique

E. W. Dijkstra. Solution of a problem in concurrent programming control.

Communications of the ACM, 8(9) :569, September 1965.

Les dîner des philosofes



Interblocage ou Famine

Quelque problèmes classiques

- **The dining savages problem** : *The Little Book of Semaphores, Allen B. Downey*
  A tribe of savages eats communal dinners from a large pot that can hold m servings of stewed missionary. When a savage wants to eat, he helps himself from the pot, unless it is empty. If the pot is empty, the savage wakes up the cook and then waits until the cook has refilled the pot.
- **The barbershop problem** : *Operating Systems Concepts, Silberschatz and Galvin*
  A barbershop consists of a waiting room with n chairs, and the barber room containing the barber chair. If there are no customers to be served, the barber goes to sleep. If a customer enters the barbershop and all chairs are occupied, then the customer leaves the shop. If the barber is busy, but chairs are available, then the customer sits in one of the free chairs. If the barber is asleep, the customer wakes up the barber. Write a program to coordinate the barber and the customers.

Comment décrire le comportement d'un système ? avec quel langage

"The only mathematics we need to describe computations are sets, functions, and simple predicate logic." (L. Lamprot Teaching Concurrency 2009)

"It is more useful to think about states than sequences of steps because what a computing device does next depends on its current state, not on what steps it took in the past. Computer engineers should learn to think about a computation in terms of states rather than verbs."

C'est quoi la programmation concurrente ?

- Modeliser ou Calculer ?
- Resultat ou invariance ?
- Specifier et verifier
- Determinisme vs non determinisme
- Enrichir le séquentiel pour faire du parallèle ou partir de primitives parallèles ?

Questions de base

- C'est quoi un processus ?
- Comment communiquent les processus ?
  - *Shared variables*
  - *Message passing*
  - *Handshaking*
- Comment se synchronisent les processus ?
  - Synchrone
  - Asynchrone
  - GALS

Outils de simulation et vérifaction

Besoin d'outils pour suivre le fonctionnement du système et vérifier certaines propriétés :

- L'activité : si le fonctionnement d'une partie ou de tout le système évolue.
- La répétitivité : s'il y a des séquences qui se répètent.
- La bornitude : quand une partie ou tout le système n'évolue plus, une fois qu'un état spécifique est atteint.
- La vivacité : Vérifie qu'un état du système puisse être atteignable, quel que soit l'état dans lequel il se trouve.

L'un de modèles le plus connu sont les réseaux de Petri.

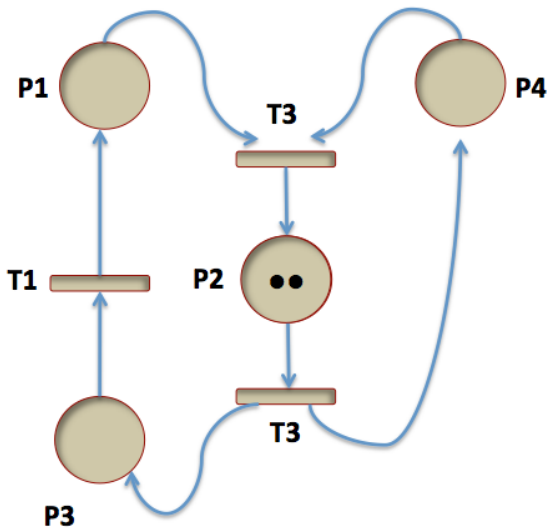
Réseaux de Petri

Proposés par Carl Adam Petri (1962)
Utilisés pour la modélisation de systèmes interactifs.

Un réseau de Petri (rdP) est un graphe biparti orienté :

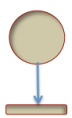
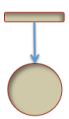
- **Places** (variable d'état du système)
- **Transitions** (événement ou action)
- **Jetons** (valeur de la variable)
- **Arcs**
  - la valeur **p** influence l'occurrence de l'action associée à **t**
  - l'occurrence de l'action associée à **t** influence la valeur de **p**

Réseaux de Petri

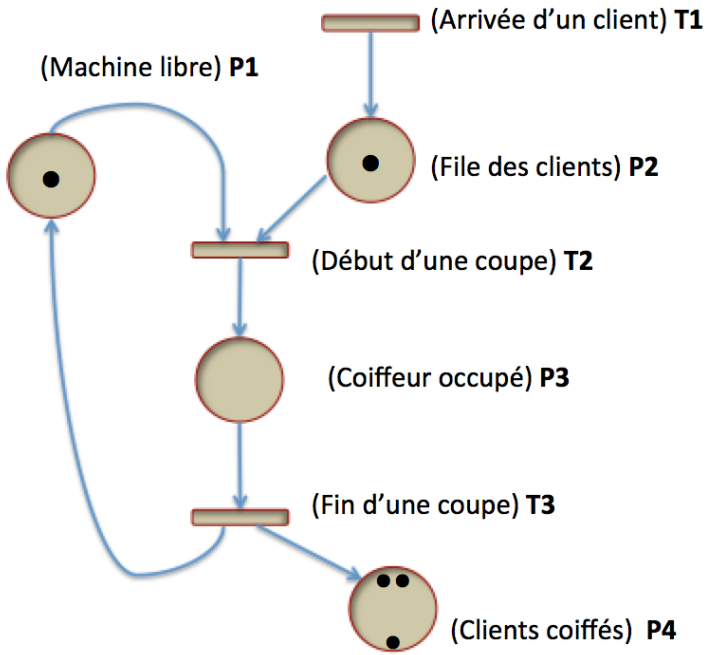


Un arc relie une place à une transition ou une transition à une place, mais jamais une place à une place ou une transition à une transition

Cas particuliers

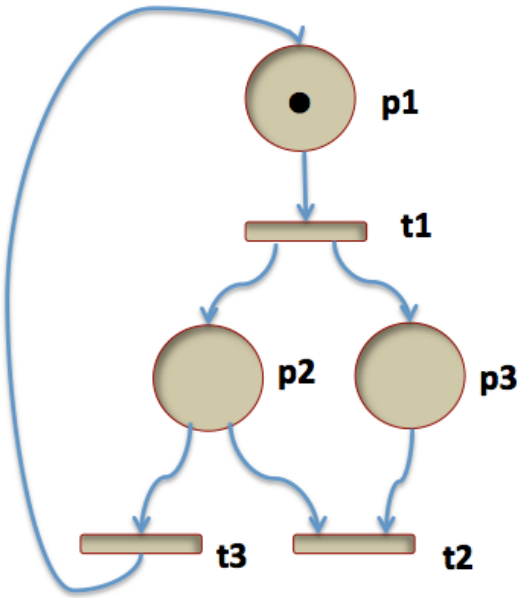
-  **transition puits** (pas de **p** en sortie de **t**)
-  **transition source** (pas de **p** en entrée de **t**)

Exemple informel

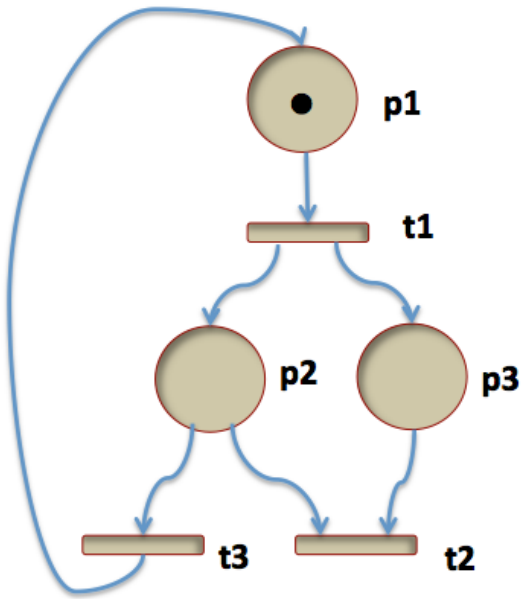


- Les places sont les variables d'état du système
- Les transitions sont des actions qui font évoluer l'état du système
- Les jetons sont indiscernables et interprétables par rapport à la place

Franchissement

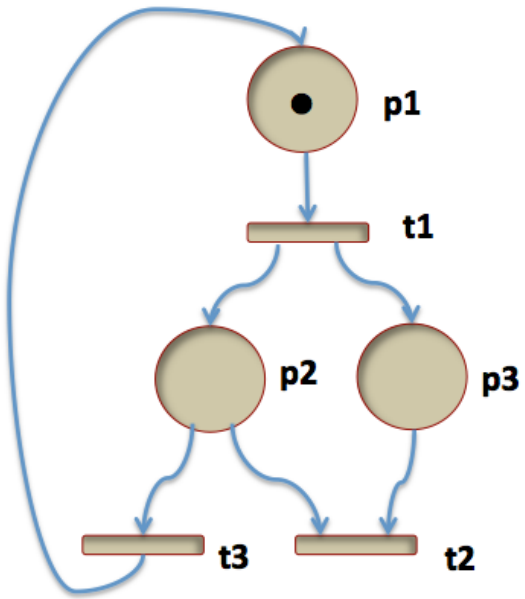


Franchissement



- $t_1 \rightarrow t_2$

Franchissement

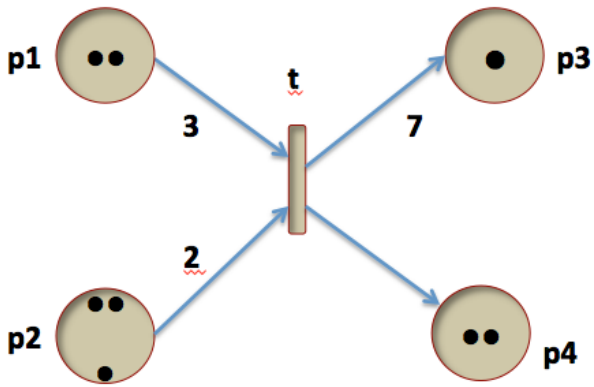


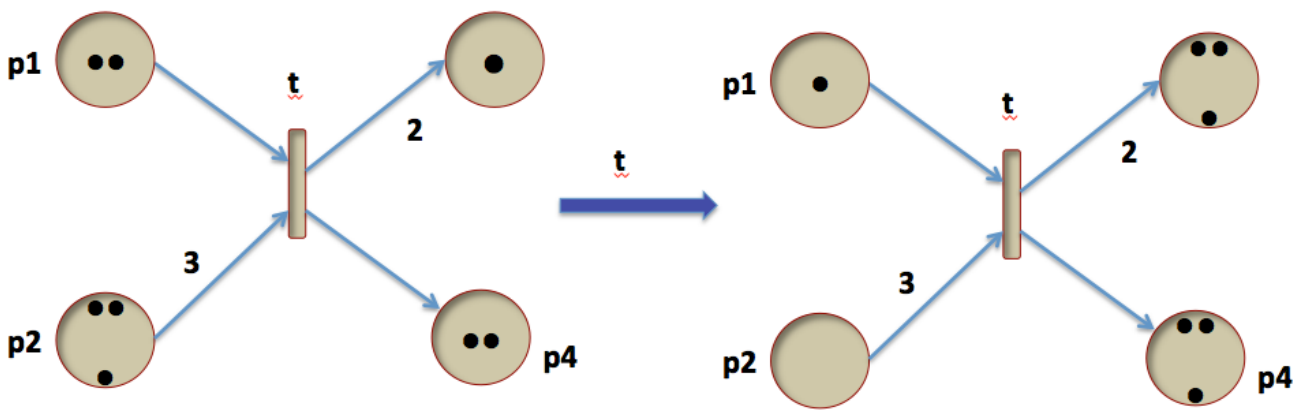
- $t_1 \rightarrow t_2$

ou

- $t_1 \rightarrow t_3 \rightarrow t_1 \rightarrow t_2$

Exemple





Définition Réseau de Petri généralisé

définition

Un rdP est un graphe biparti défini par (P,T, Pre, Post), avec :

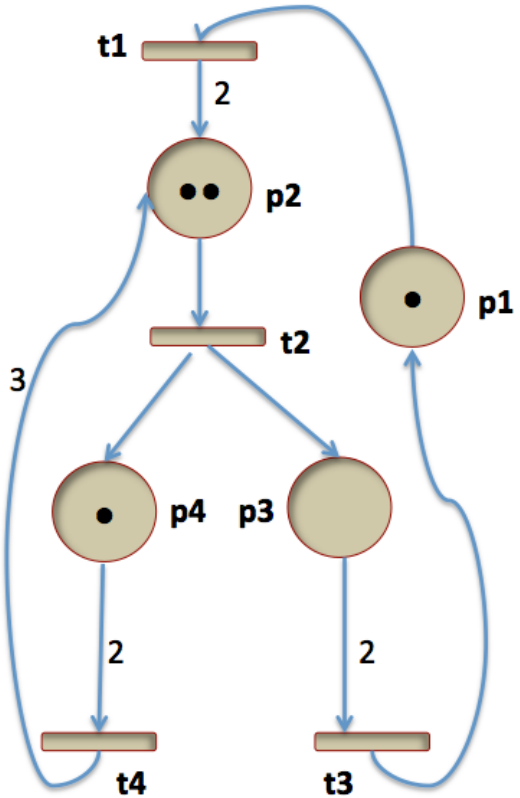
- $P = \{p_1, p_2, ..., p_n\}$
- $T = \{t_1, t_2, ..., t_m\}$
- Pre : $P \times T \rightarrow \mathbb{N}$ (de P vers T)
- Post : $P \times T \rightarrow \mathbb{N}$ (de T vers P)

Un marquage est un vecteur avec le nombre de jetons pour chaque place à un instant donné.
M_0 dénote le marquage initial

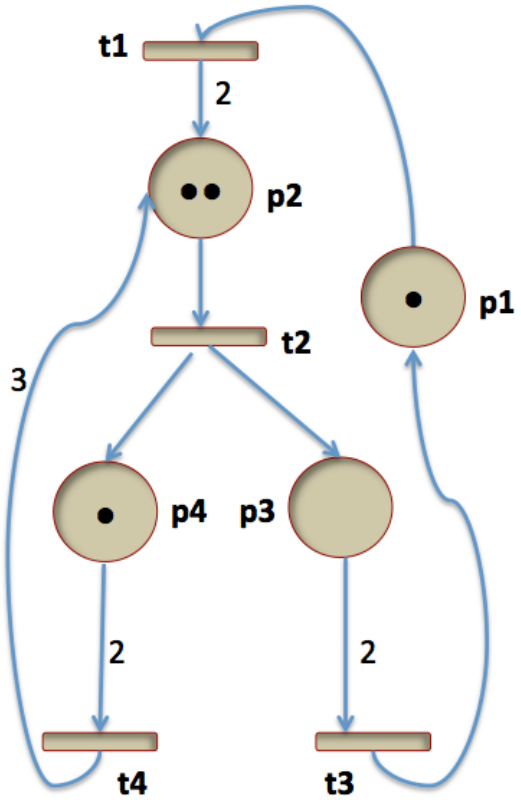
définition

Un rdP marqué = (P,T,Pre, Post, M_0)

Exemple



Exemple

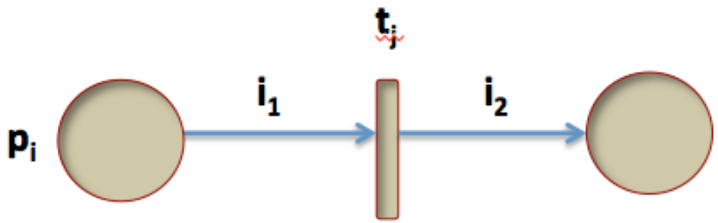


$$Pre = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix} \quad Post = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 3 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \quad M_0 = \begin{pmatrix} 1 \\ 2 \\ 0 \\ 1 \end{pmatrix}$$

Evolution du marquage

- L'évolution d'un rdP correspond à l'évolution du M_0
- Les jetons peuvent passer d'une place à l'autre par franchissement d'une transition
- Une transition est franchissable (valide), si les places d'entrée possèdent un nombre de jetons $\geqslant$ au poids des arcs entre chaque place et la transition
- Franchir une transition consiste à enlever à chaque place d'entrée le nombre de jetons égal au poids de l'arc d'entrée et à déposer dans la place de sortie le nombre de jetons de son arc d'entrée.
- Lorsqu'une transition est franchissable, cela n'implique pas qu'elle sera franchie tout de suite. Le réseau évolue en franchissant une seule transition à la fois parmi l'ensemble de transitions valides.

Matrice d'incidence



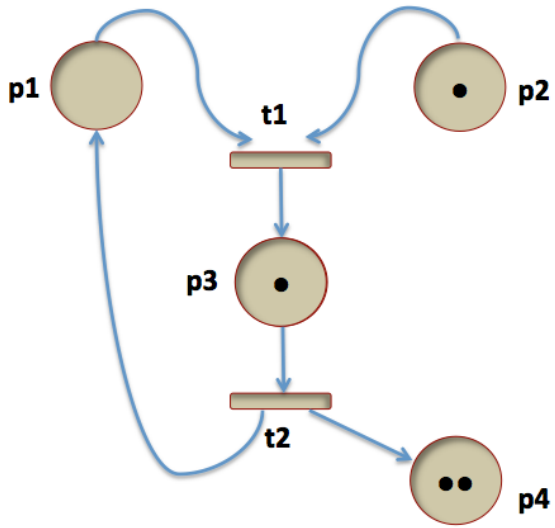
- Condition nécessaire : $M(p_i) \geqslant \text{Pre}(p_i, t_j)$ où M est le marquage courant
- Franchir t_j produit un nouveau marquage M' tq. $\forall p_i \in P, M'(p_i) = M(p_i) - \text{Pre}(p_i, t_j) + \text{Post}(p_i, t_j)$

Définition

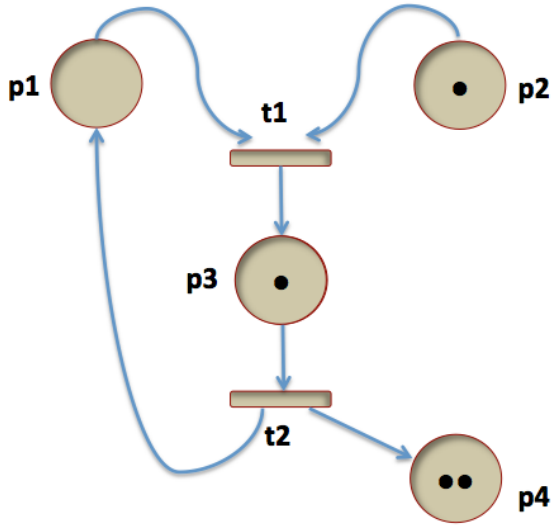
On défini la matrice d'incidence W par :
$W(p_i, t_j) = \text{Post}\ (p_i, t_j) - \text{Pre}(p_i, t_j) \qquad \forall p_i \in P \forall t_i \in T$

Un nouveau marquage M' à partir du Marquage M par franchissement de la transition t_i est obtenu par : $\text{M'} = \text{M} + \text{W} \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} \leftarrow t_j$ (notation $\text{M}[t_j > \text{M'}$)

Algebre

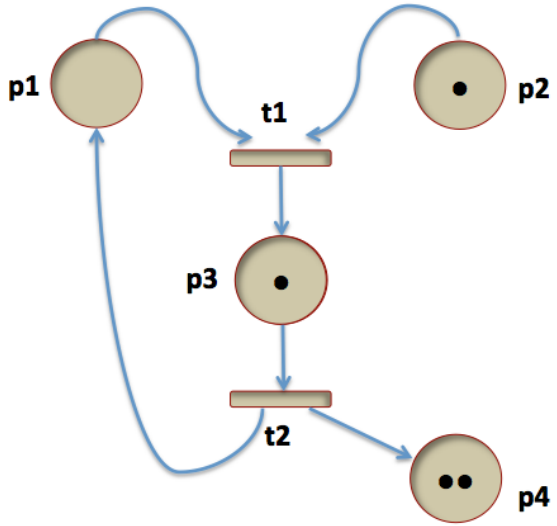


Algebre



$$M_0 = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 2 \end{bmatrix} \quad Pre = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \quad Post = \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \quad W = Post - Pre = \begin{pmatrix} -1 & 1 \\ -1 & 0 \\ 1 & -1 \\ 0 & 1 \end{pmatrix}$$

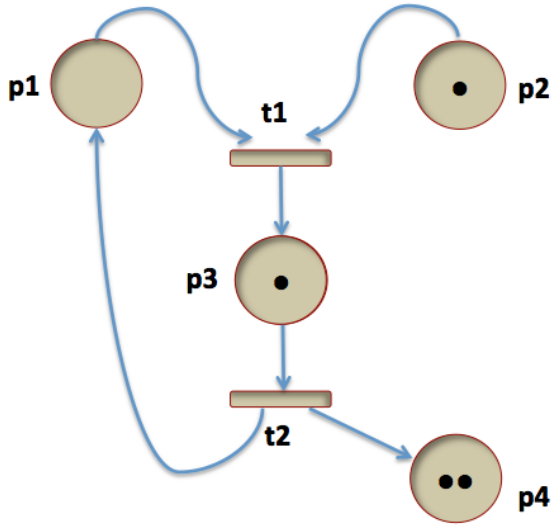
Algebre



$$M_0 = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 2 \end{bmatrix} \quad Pre = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \quad Post = \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \quad W = Post - Pre = \begin{pmatrix} -1 & 1 \\ -1 & 0 \\ 1 & -1 \\ 0 & 1 \end{pmatrix}$$

$$M' = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 2 \end{bmatrix} + \begin{bmatrix} -1 & 1 \\ -1 & 0 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 3 \end{bmatrix}$$

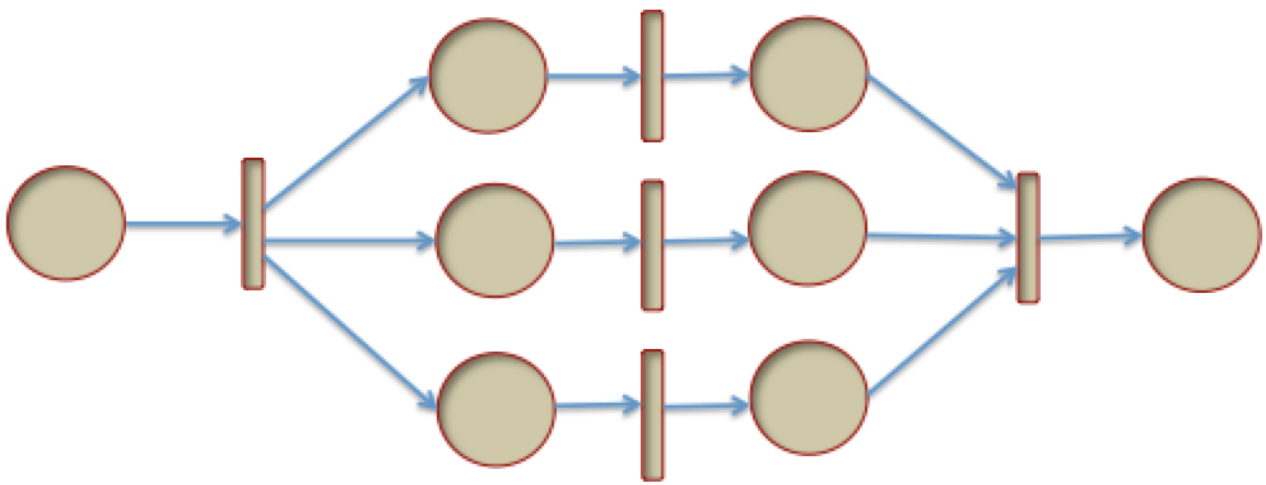
Algebre



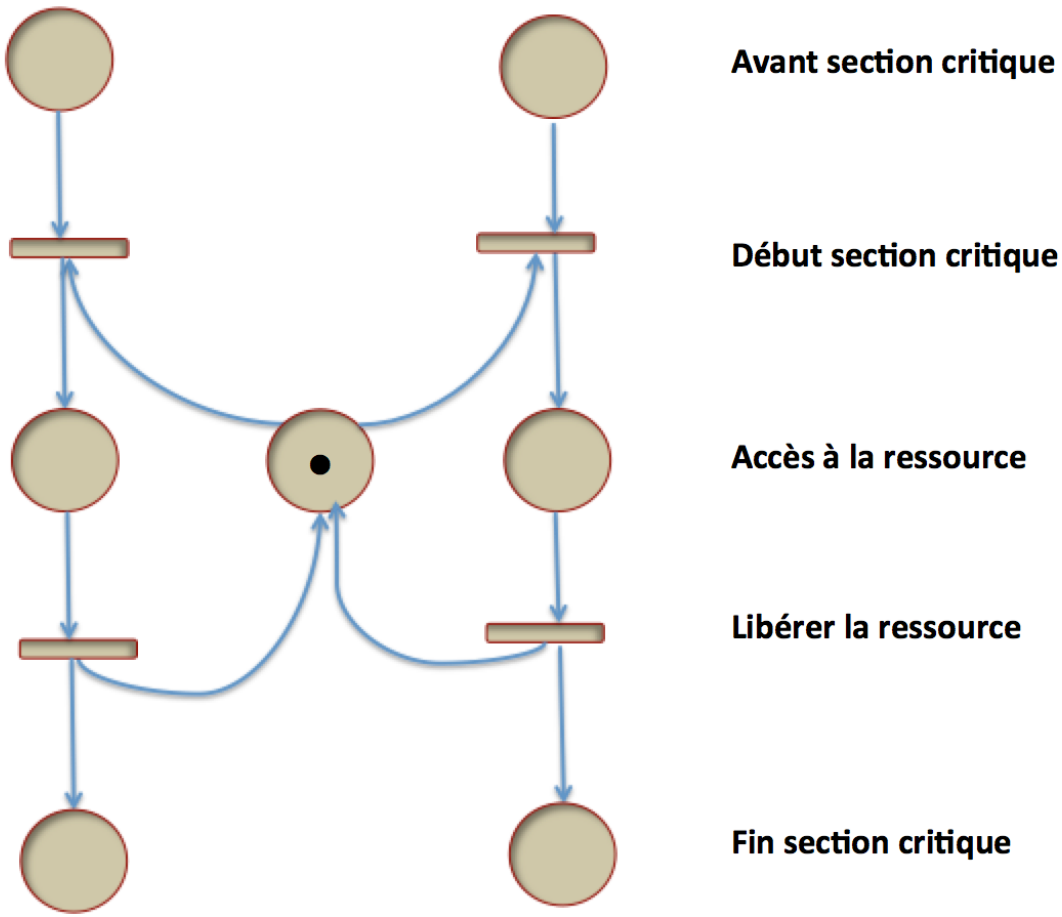
$$M_0 = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 2 \end{bmatrix} \quad Pre = \begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \quad Post = \begin{bmatrix} 0 & 1 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \quad W = Post - Pre = \begin{pmatrix} -1 & 1 \\ -1 & 0 \\ 1 & -1 \\ 0 & 1 \end{pmatrix}$$

$$M' = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 2 \end{bmatrix} + \begin{bmatrix} -1 & 1 \\ -1 & 0 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 3 \end{bmatrix} \quad M'' = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 3 \end{bmatrix} + \begin{bmatrix} -1 & 1 \\ -1 & 0 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 3 \end{bmatrix}$$

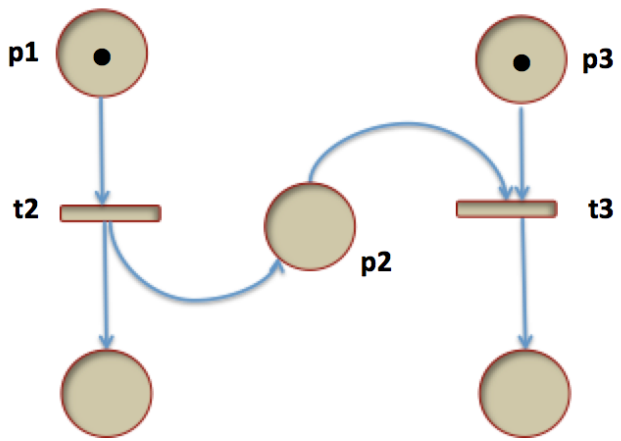
Parallèlisme et synchronisation



Partage de ressources



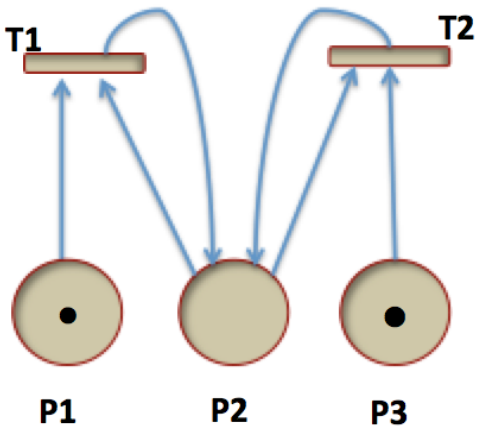
Communication asynchrone



Blocage

Definition : blocage

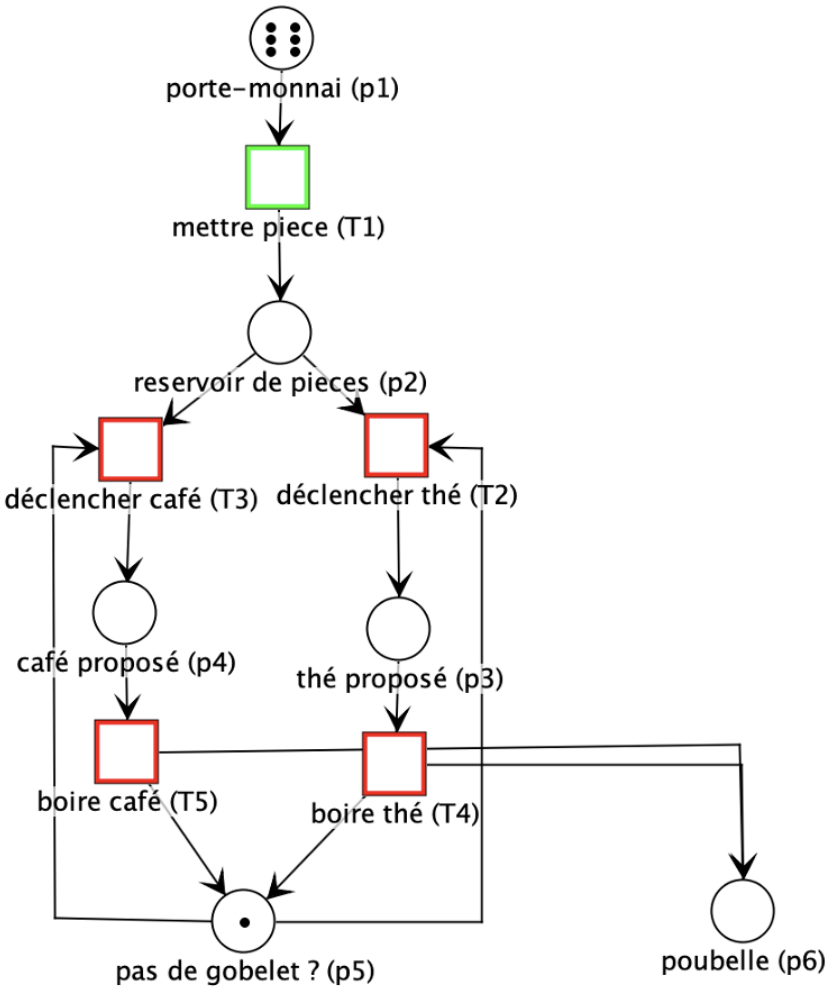
Un blocage est un marquage pour lequel il n'y a pas de transition valide.
Un rdP est dit sans blocage pour M_0 si il n'y a pas de marquage dans $*M_0$ qui soit un blocage.



$$M_0 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

Une machine à café

Une machine à café



Références

Petri Nets

- Slides de Vincent Augusto ESM Saint-Etienne
- Les Réseaux de Petri. Notes de cours par N. Bennis
- Plein de pages sur le net, en particulier `www.informatik.uni-hamburg.de/TGI/PetriNets/`

Programmation DataFlow : systèmes réactifs

- Systèmes informatiques qui réagissent à l'environnement.
- La vitesse des événements est imposée par l'environnement.
- Ils sont concurrents par nature.
- Contraints temporellement (i/o).
- Généralement déterministes.
- La fiabilité est importante.
- Généralement faits de software et hardware.

Dataflow

- Pour faire du calcul sur des flux de données .
- Les flux de données traversent une structure d'acteurs (le programme) qui vont les transformer.
- En théorie, ce modèle de computation ignore la notion de temps.
- Une exécution *dataflow* consiste en une sequence de *firings*, *triggers*, *bangs*, tirs.
- Chaque *fire* arrive comme une réaction à la disponibilité d'une donnée.
- Un tir est une unité de calcul (petite, en général) qui consomme une entrée et produit une sortie.

Un exemple puredata

Dataflow

Les acteurs commencent leur exécution (sont déclenchés) quand leurs entrées sont prêtes.
C'est quoi être prêt ?
Deux familles dataflow :

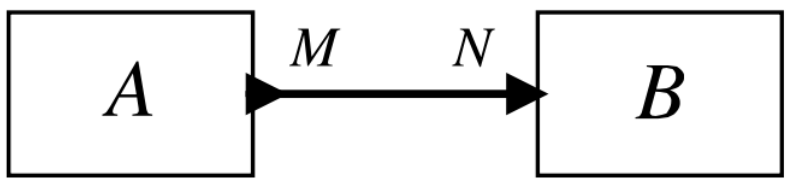
- *static dataflow (SDF synchronous dataflow)*
  -l'ordre d'exécution des acteurs ne dépend pas des données
- *dynamic dataflow (DDF)*
  - on peut avoir des conditionnels par rapport à la valeur d'une entrée

Static Dataflow

Definition (itération)

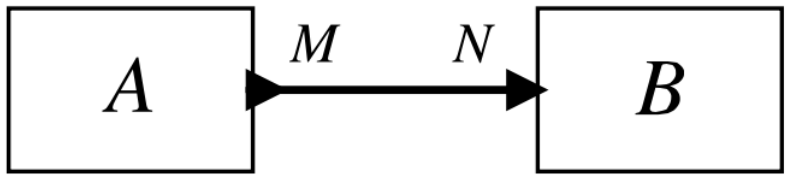
Une itération consiste en UN fire de tous les acteurs possibles.
Si chaque acteur produit et consomme un token, le programme est dit homogène

Les acteurs peuvent consommer et produire de multiples tokens à chaque déclenchement.



Exemple FFT

Les équations de balance



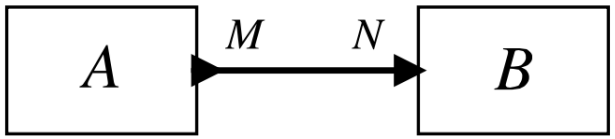
Quand A tire, on produit M *tokens* et quand B tire, on consomme N *tokens*.

Si A tire q_A fois et B tire q_B fois alors les tirs de A sont consommés par B si l'équation suivante est satisfaite :
$q_A M = q_B N$

On trouve les plus petites q_A et q_B qui satisfont l'équation et on construit un scheduler qui tire A q_A fois et B q_B fois. Le system est balancé : A produit le même nombre de tokens que B consomme.

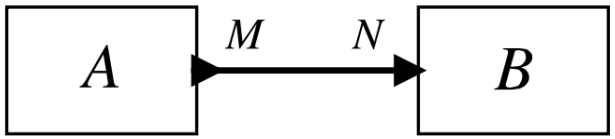
Puisqu'on peut répéter cette opération autant de fois que l'on veut (avec une quantité de mémoire fixe), on dit qu'on réalise une **unbounded execution** avec **bounded buffers**.

Exemple



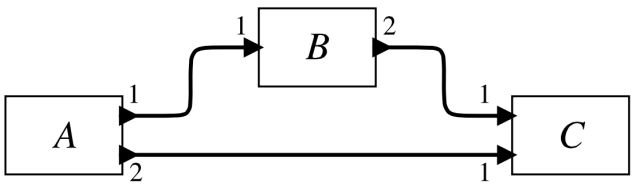
- Pour $M = 2$ et $N = 3$, une solution de l'équation de balance est donnée par : $q_A = 3$ et $q_B = 2$.
- On trouve les plus petites q_A et q_B qui satisfont l'équation et on construit un scheduler qui tire A q_A fois et B q_B fois.
  Alors, on peut tirer la sequence $AAABB$

Exemple



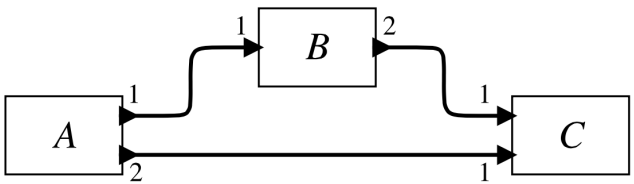
- Pour $M = 2$ et $N = 3$, une solution de l'équation de balance est donnée par : $q_A = 3$ et $q_B = 2$.
- On trouve les plus petites q_A et q_B qui satisfont l'équation et on construit un scheduler qui tire A q_A fois et B q_B fois.
  Alors, on peut tirer la sequence $AAABB$
- Cependant, la séquence $AABAB$ a besoin de moins de mémoire pour garder les *tokens* intermédiaires.
- Des solutions plus grandes (e.g. $q_A = 3$ et $q_B = 2$) compliquent l'optimisation de la mémoire.
  La solution $q_A = 0$ et $q_B = 0$ n'est pas utile, ainsi que les solutions négatives.

Exemple



Les équations de balance sont :

Exemple



Les équations de balance sont :

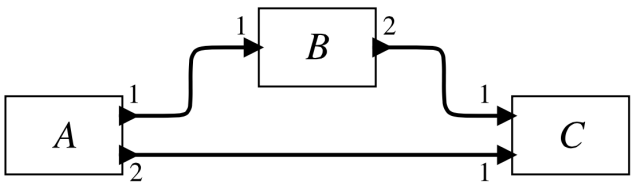
$q_A = q_B$

$2q_B = q_C$

$2q_A = q_C$

La solution la plus petite est :

Exemple



Les équations de balance sont :

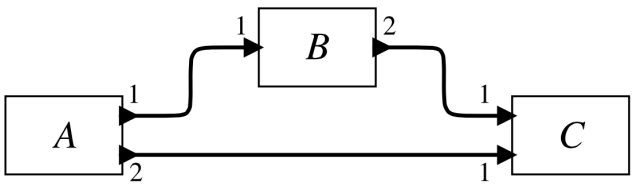
$q_A = q_B$
$2q_B = q_C$
$2q_A = q_C$

La solution la plus petite est :

$q_A = 1$, $q_B = 1$ et $q_C = 2$

La séquence optimale en mémoire est :

Exemple



Les équations de balance sont :

$q_A = q_B$

$2q_B = q_C$

$2q_A = q_C$

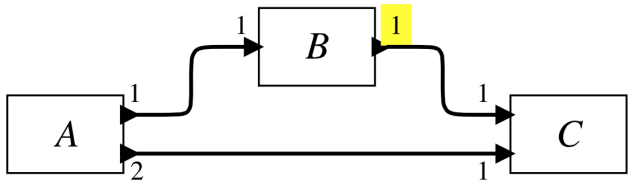
La solution la plus petite est :

$q_A = 1$, $q_B = 1$ et $q_C = 2$

La séquence optimale en mémoire est :

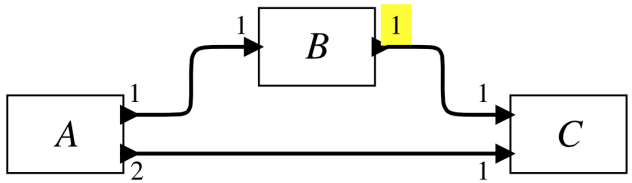
A, B, C, C

Exemple



Les équations de balance sont :

Exemple



Les équations de balance sont :

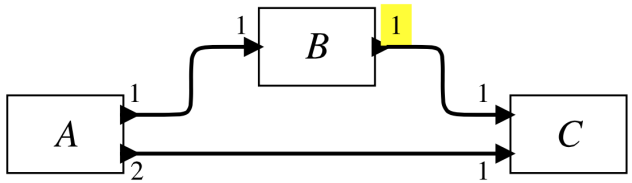
$q_A = q_B$

$q_B = q_C$

$2q_A = q_C$

La solution la plus petite est :

Exemple



Les équations de balance sont :

$q_A = q_B$

$q_B = q_C$

$2q_A = q_C$

La solution la plus petite est :

$q_A = 0$, $q_B = 0$ et $q_C = 0$

Le modèle est inconsistant. Il ne possède pas une **unbounded execution** avec **bounded buffers**.

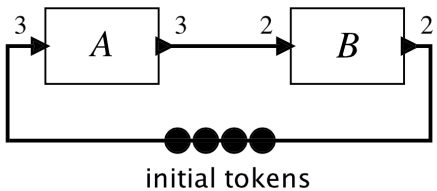
Exemple **unbounded execution** avec **bounded buffers**

Feedback loops

Exemple feedback loops

Feedback loops

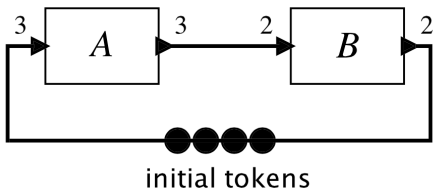
Exemple feedback loops



Les équations de balance sont :

Feedback loops

Exemple feedback loops

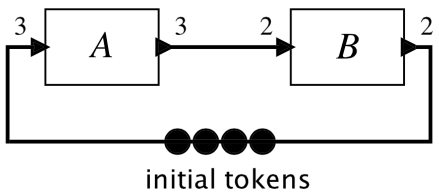


Les équations de balance sont :
$3q_A = 2q_B$
$2q_B = 3q_A$

La solution la plus petite est :

Feedback loops

Exemple feedback loops



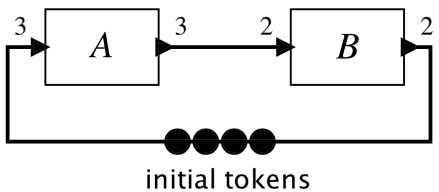
Les équations de balance sont :
$3q_A = 2q_B$
$2q_B = 3q_A$

La solution la plus petite est :
$q_A = 2$ et $q_B = 3$

Une séquence possible est donnée par :

Feedback loops

Exemple feedback loops



Les équations de balance sont :
$3q_A = 2q_B$
$2q_B = 3q_A$

La solution la plus petite est :
$q_A = 2$ et $q_B = 3$

Une séquence possible est donnée par :
A, B, A, B, B
La consistance est suffisante pour assurer **bounded buffer** mais pas **unbound execution**, car le modèle peut *deadlock* même s'il est consistant.
Heureusement, on peut trouver, s'il est possible, la séquence qui assure **unbound execution**. On peut aussi savoir si une telle séquence n'existe pas.

Extensions

- fire,
- iteration,
- complete iteration.

Beaucoup d'extensions, e.g. dynamically varying rates
i.e. on peut faire un static data flow avec variable rate (M et N peuvent changer) mais uniquement après une iteration complete.

Dynamic dataflow

- Le problème avec SDF est le pouvoir d'expression. En particulier, on n'a pas de tir conditionnel, c-à-d qu'un acteur tire pour une valeur particulière.
- Dans DDF, chaque acteur a une règle de tir (une condition qui doit être satisfaite avant que l'acteur soit tiré).
- SDF est un cas particulier de DDF avec pour unique condition les *tokens* nécessaires en entrée.
- Une autre différence est qu'avec DDF, le nombre de tokens de sortie pour un acteur peut varier.
- Même la règle d'un acteur est dynamique, elle peut changer après un tir.

Select et *Switch*

- `BoolenSelect` mélange deux flux dans un seul par rapport à un troisième flux de type boolean.
- `GeneralSelect` mélange n flux dans un seul par rapport à un troisième flux de type integer.
- `BoolenSwitch` sépare un flux en deux par rapport à un troisième flux de type boolean.
- `GeneralSwitch` sépare un flux en plusieurs autres par rapport à un troisième flux de type integer.

Exemple DDF et **no bounded buffer**

Exercice

Prendre en flux les entiers 0,1,2,... et à chaque fois diviser par 2 jusqu'à que la valeur soit <= 0.5

Deux modes d'exécution logicielle classiques

- Système échantillonné :

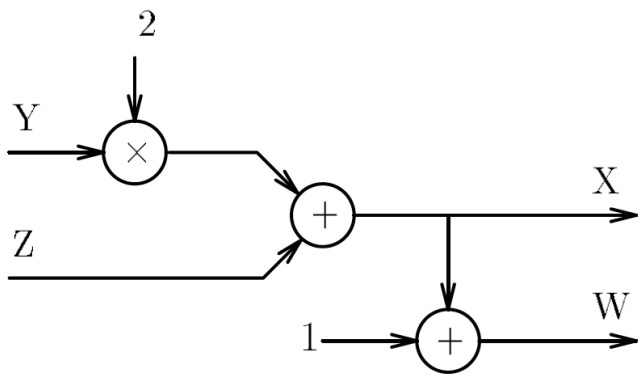
```
Initialize Memory
for each clock tick do
   - Read Inputs
    -Compute Outputs
    -Update Memory
end
```

- Système événementiel :

```
Initialize Memory
for each input event do
         -Compute Outputs
         -Update Memory
end
```

S'assurer que le temps de calcul du corps de la boucle (réeaction) est inférieur à la période de l'horloge temps-réel ou du délai minimal entre deux événements successifs

Programmes comme equations



$\forall t,\ X(t) = 2Y(t) + Z(t)$ et $W(t) = X(t) + 1$