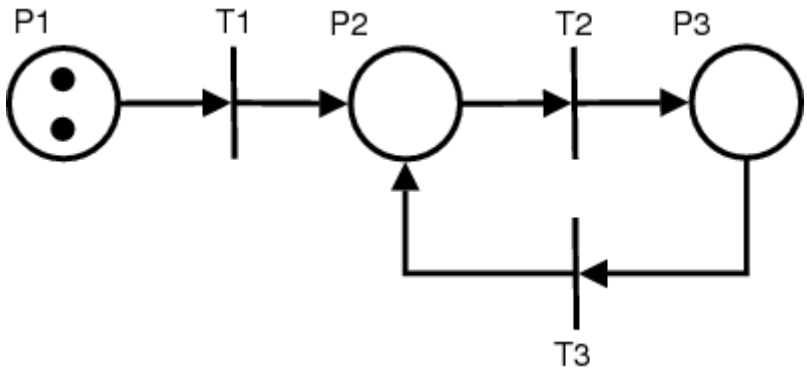
Professeur : Alexandre Hamez

Réseau de Petri



Soulon Hak

Ing 2 MTI 2009

Juin 2008

Sommaire

Introduction

- Programme = Code + exécution
    - Architecture statique + comportement dynamique
- Tendance : multi-processeurs, multi-cœurs, clusters, par à pait
    - Programmes concurrents (parallèles) et **répartis**
    - Complexité de la conception
    - Mauvaise conception = inter-blocages, famines, etc.

Modéliser des programmes concurrents

Comment modéliser cette politique d'accès ? Avec un diagramme de classes ?

Modéliser pour vérifier

Propriétés attendues du modèle
ex : equité, vivacité

Les propriétés sont elles vérifiées ?

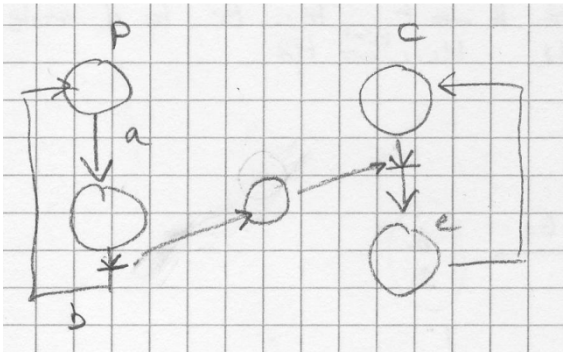
Génération du code correspondant au modèle.

Modéliser : Uitliser un formalisme de description

- Il existe plusieurs formalisme de description du comportement d'un programme

Pourquoi pas des automates ?

- Comment gérer le parallélisme avec les automates ? L'état d'une ressource (ex : un buffer) ?
- On pourrait représenter tous les états du processus de chaque processus par un automate différent, idem pour une ressource (donc bornée)
    - Ensuite on synchronise les automates
- L'automate ne peut que représenter l'état global du système : on ne peut pas modéliser des composants autonomes que l'on connecterait **simplement**
    - ex : un modèle avec **5 clients** et **2 serveurs** connectés de manière asynchrone : **111** états / **305** arcs



On aimerait des conditions pour avancer dans le processus
Y a-t-il un problème avec ça ?
Plutôt qu'avoir des conditions vrai/faux, on va passer au entier (réseau de Pétri).

Réseaux de Petri : présentation

- Graphe orienté biparti
- Places : représentées par un cercle
- Transitions : représentées par un rectangle
- Les places contiennent des jetons
    - On parle de marquages d'une place
- Les arcs portent des **valuations**
    - Ils ne peuvent pas relier deux sommets du même type.

Places et transitions

<P, T, Pré, Post>
P : ensemble de places
T : ensemble de transitions
Pré : $P \times T \rightarrow \mathbb{N}$
Post : $P \times T \rightarrow \mathbb{N}$

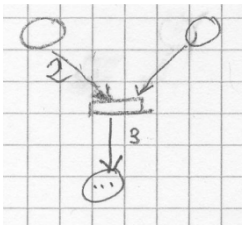
<R, M_0>
M_0 : marquage initial

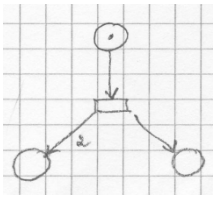
Places :

- Les états locaux d'un processus
- Une ressource partagée par plusieurs processus

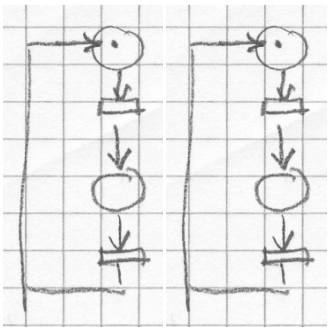
Franchissement dans un RdP (1)

- Une transition t est franchissable si pour toute place Pi telle qu'il existe un arc entrant <Pi, t> on a
- Les arcs entrants consomment des jetons. Chaque place Pi telle qu'il existe un arc <Pi, t> voit son marquage diminuer
- Les arcs sortants « produisent » des jetons. Chaque place Pj telle qu'il existe un arc <t, Pj> voit son marquage augmenter





Parallélisme



Contraintes sur les systèmes modélisés

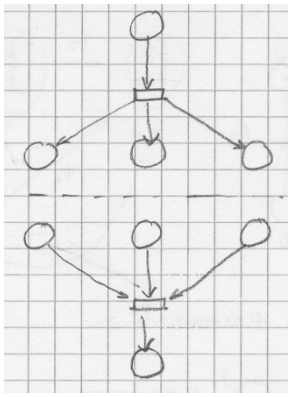
- Le système modélisé doit vérifier certaines hypothèses
  - Il existe une notion d'**état global**
  - Il existe des règles permettant de passer d'un état global à un autre (état suivant)
  - Les franchissements entre états doivent être **indivisibles**

Chercher et corriger l'erreur

Modèle de spécifications de base

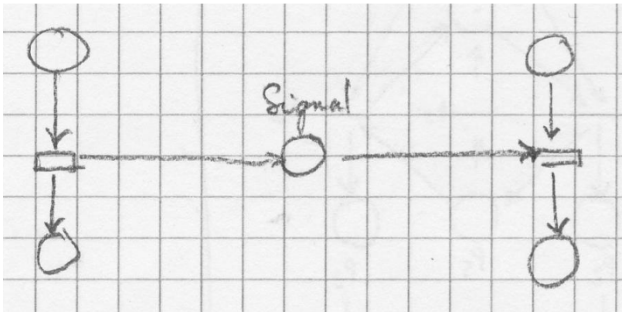
- Beaucoup de motifs se répètent dans la modélisation
  - Lancement parallèle et terminaison synchronisée
  - Synchronisation par signal
- Etc.

Lancement parallèle et terminaison synchronisée



Tous les processus sont créés en même temps. C'est seulement dans un point de vue de haut niveau, qu'on peut tirer un seul évènement à la fois. Il faut attendre que tous les processus soient présents pour qu'on puisse passer à l'étape suivant.
Dans le 2^{e} cas, il faut les 3 jetons pour en produire un seul.
Il faut penser en termes de « fork », « join».

Synchronisation par signal



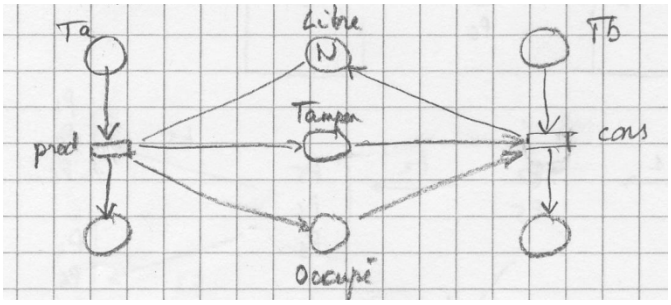
La version asynchrone parce qu'il faut représenter les états intermédiaires.

Exclusion mutuelle

Lecteur – écrivains

Le lecteur va autoriser n lectures_simultanées.
A chaque fois que l'écrivain a finit d'écrivain, il va donner n autorisations.

Tampon borné

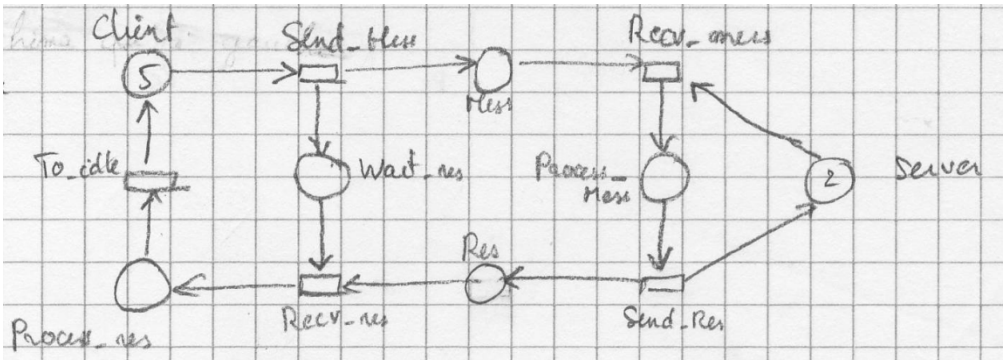


On a un thread A qui va produire, et un thread B qui va consommer.
La place libre, représente le nombre de place libre dans mon tampon.
On produit une place dans le buffer, une donnée dans la ressource.

Exemple : Client – serveur

- N Clients, M serveurs
- Communiquent par messages
    - Asynchrone
- Le client
    - Envoie une requête au serveur
    - Attend la réponse (il peut procéder à d'autres etc.)

Exemple : Client – serveur / Asynchrone

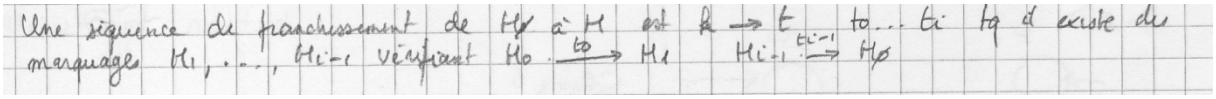


Peut-on avoir la famine, des processus en famine, cad qu'ils n'ont pas de ressources ?
Le problème c'est que ce n'est absolument pas FIFO, il n'y a pas de jetons.
On a que des entiers là-dedans, pour les taguer, ça sera difficile.
On n a pas d'interblocages, mais on peut avoir des processus qui finalement ne feront jamais rien.
Ici on a un problème de famine.

Asynchrone : veut dire qu'on n'a pas de notion de temps. On attend pas le résultat, on ne sait pas combien de temps, ça va prendre.

Automate du modèle Client-Serveur…

Graphe d'accessibilité d'un réseau <R, M0> -> <θ, Δ, λ, q0>
θ : l'ensemble des marquages accessibles depuis M0.
Δ : ensemble des arcs reliant 2 marquages accessibles
λ : étiquette les arcs par le nom de la transition de R qui a été franchie (q0 = M0)



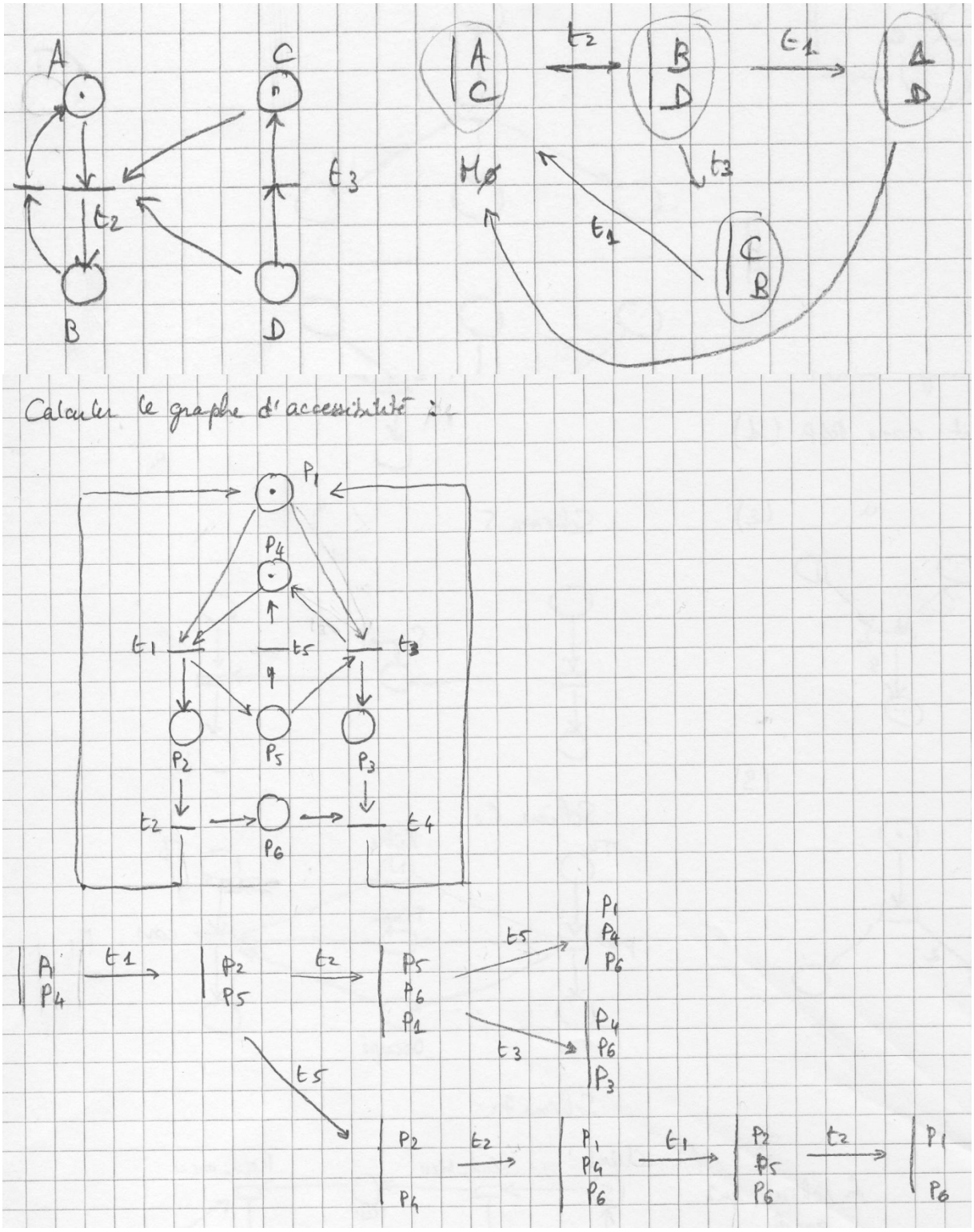
$$t \in T \; est \; franchissable \text{ à } partir \; de \; M \; ssi$$

$$\forall p \in P, M(p) \geq Pré(p,t)$$

$$\forall p \in P, M'(p) = M(p) - Pré(p,t) + Post(p,t)$$

BFS :

```
Set <- Q0
New <- Q0
tant que New ≠ 0
   New\{x}
   pour tout s succ(x)
      if s∉Set
         Set ∪ {s}
         New
```

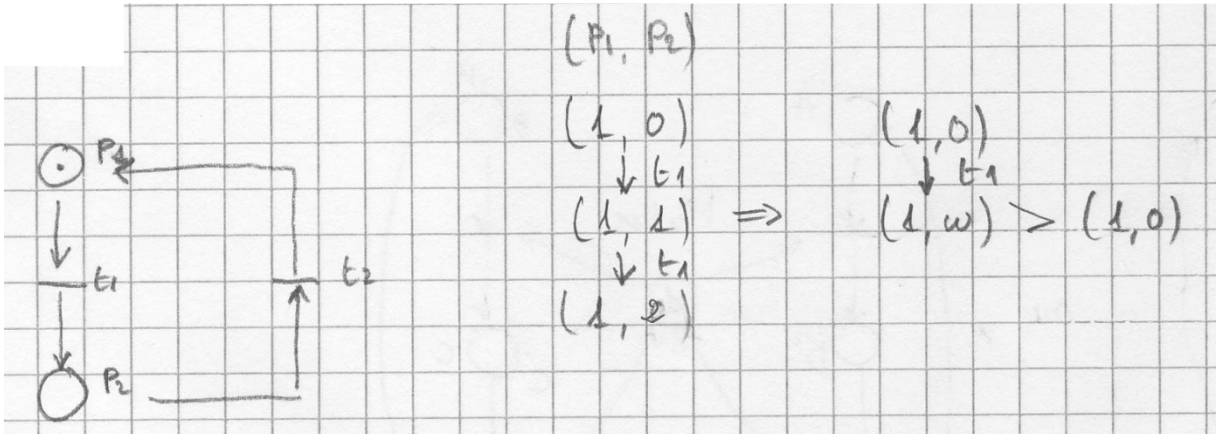


Graphe de couverture

Définition :

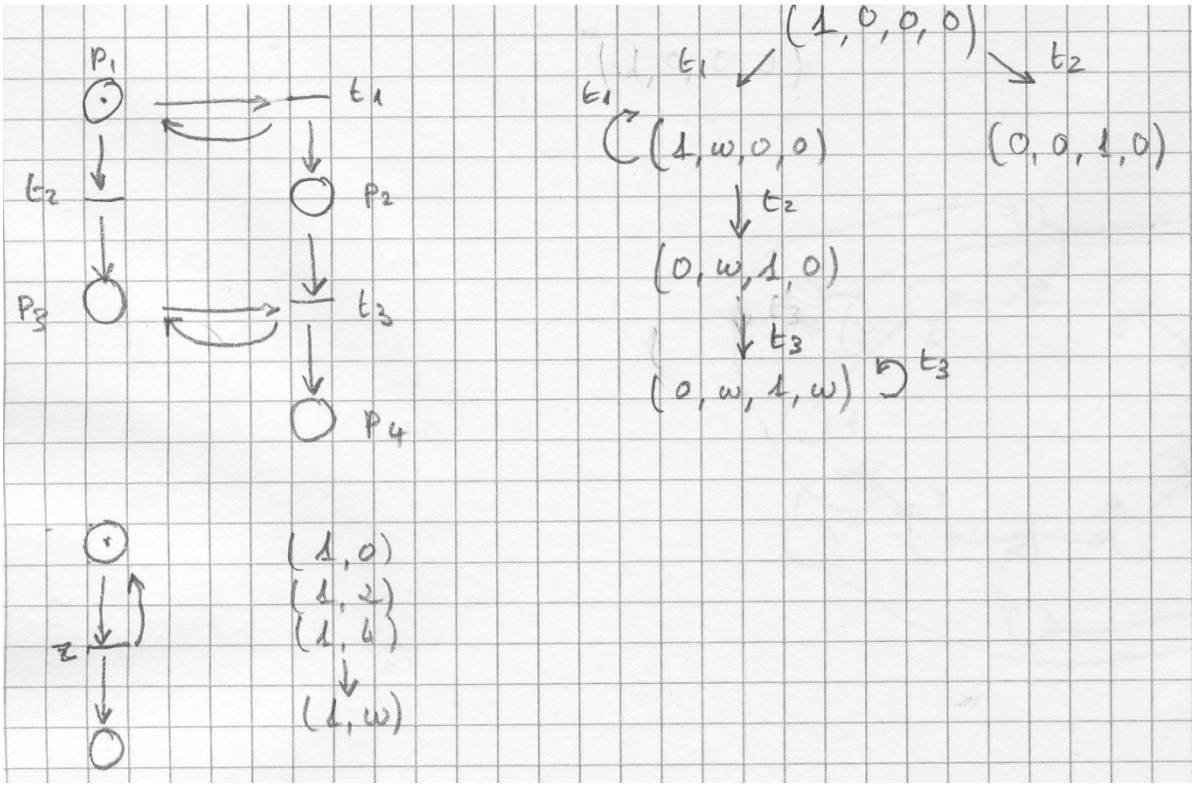
Pour un réseau <R, M0> on a <θ, Δ, λ, q0>

Q un ensemble de marquages sur $\mathbb{N} \cup \{w\}$ avec $\forall\, k \neq 0,\ \omega + k = \omega,\ \omega - b = \omega, \omega > k$

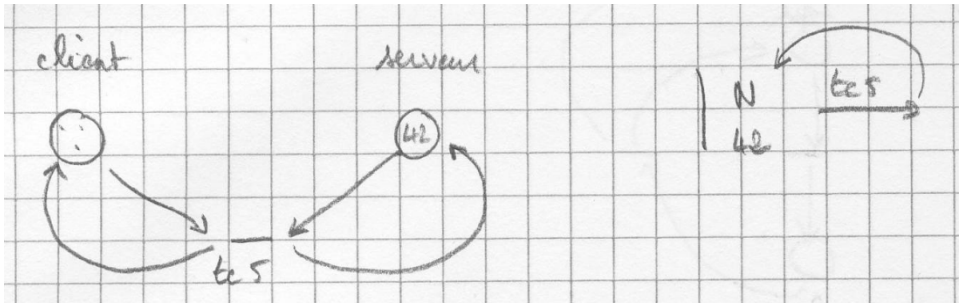


On aura des soucis d'accessibilité.

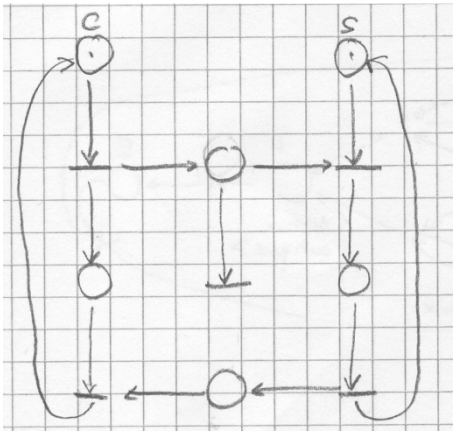
Le ω pourra couvrir des cas qui ne sont pas accessible.



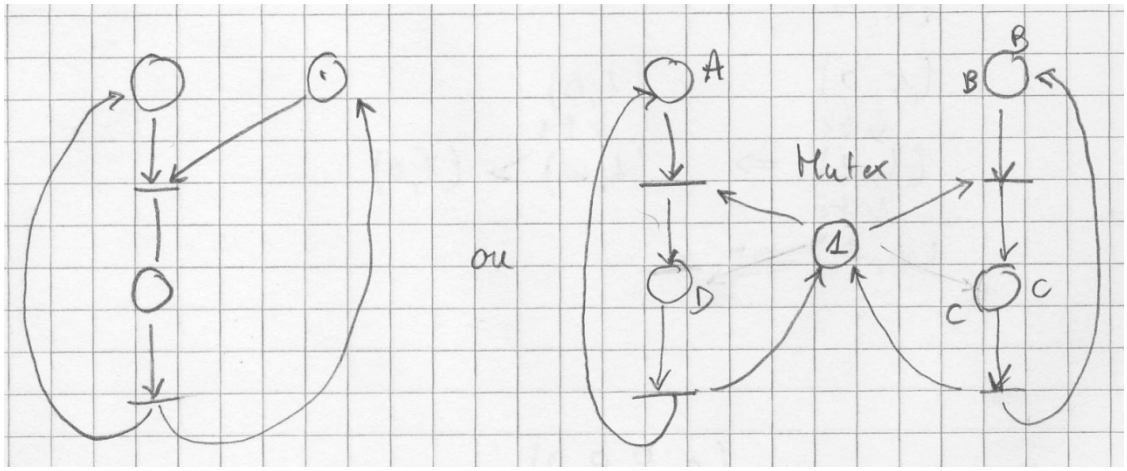
Modéliser un Client – Serveur / Synchrone



Modéliser une perte de messages

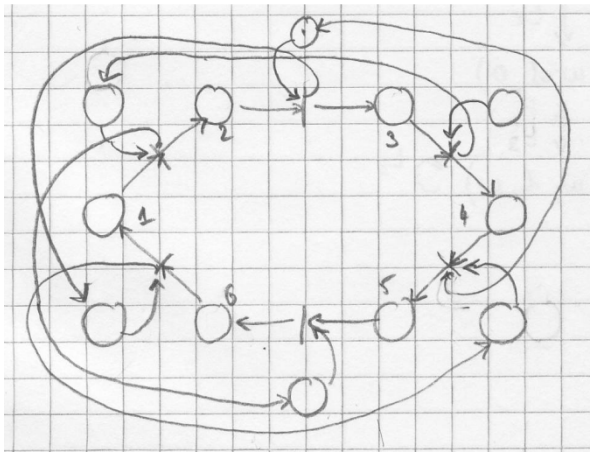


Modéliser un accès à une ressource critique



Trains sur section circulaire

- Soit un rail circulaire de 6 sections
  - 2 trains qui circulent dans le même sens.
- Initialement :
  - Train 1 sur section 1
  - Train 2 sur section 3
- Un train ne peut rentrer dans une section si
  - Ni cette section n
  - Ni la section n + 1 n'a de train

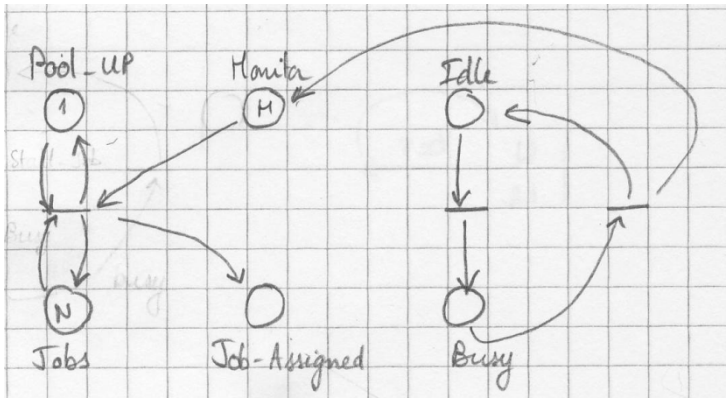


Se dire qu'on a là des sections critiques avec un mutex.

Modéliser une pile de thread.

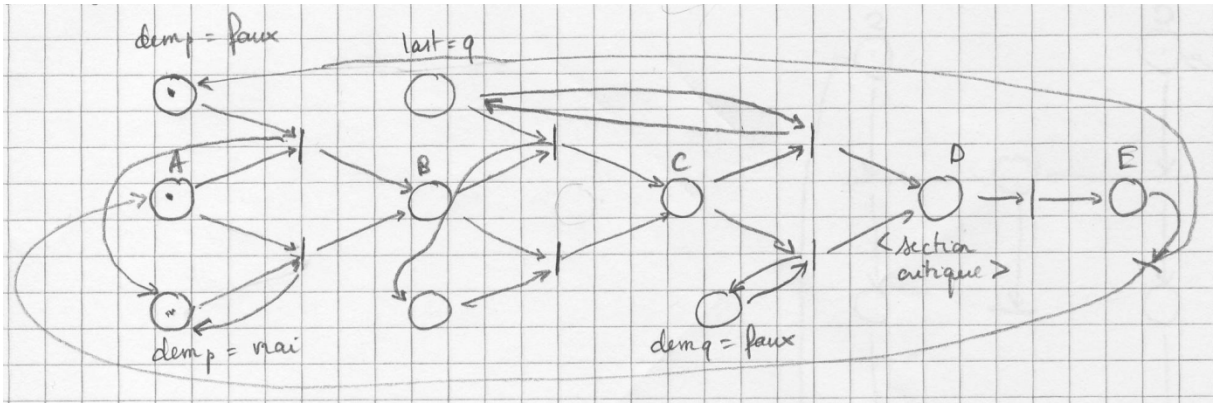
On a des threads qui prennent des jobs.
S'assurer qu'on a un contrôleur de thread.
Le thread n'a que 2 états :

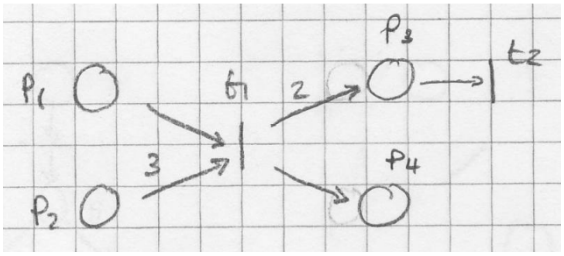
- Il ne fait rien
- Il travaille



Algorithme de Peterson

- dem_p
- $last = p$
- Attendre ($last == q \;||\; dem_q == faux$)
- <Section critique>
- $dem_p = faux \;;\; goto\ A$





Avant transition

Pré :

	t1	t2
P1	1	0
P2	3	0
P3	0	1
P4	0	0

Après transition

Post :

	t1	t2
P1	0	0
P2	0	0
P3	2	0
P4	1	0

Mains

Post-Pré :

Matrice d'incidence :

	t1	t2
P1	-1	0
P2	-3	0
P3	2	-1
P4	1	0

Matrice d'incidence C = Post – Pré

$M \xrightarrow{t} M'$

$M'(f) = M(f) + Post(f,t) - Pré(p,t) = M(f) + C(f,t)$

Vecteur caractéristique

On sait s : $t_a t_b$ une séquence de franchissement

$\underline{s} \quad T \to \mathbb{N}$

$\underline{s} \begin{bmatrix} ta & tb & tc \\ x & y & z \end{bmatrix}$

Equation caractéristique

$M \xrightarrow{s} M' \rightarrow M' = M + C.\underline{s}$

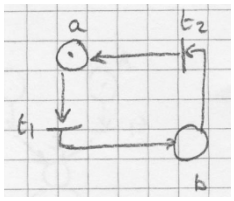
Séquence infinie

<R, M_0> admet une séquence infinie s où s est un mot infinie sur T, ssi pour s' préfixe fini de s.

s' est une séquence de franchissement

Si $s = t1.t2 \ldots tm \ldots$ alors $\forall i$, si $si = t1 \ldots ti$ on a $M0 \xrightarrow{si}$

Exemple :



Pseudo vivacité

$< R, M_0 >$ est pseudo vivant si pour $\forall M \in GMA(R, M_0)$ (graphe accessibilité), il existe un trans t tq :

$M \xrightarrow{t}$

(Franchir au moins un état, jamais bloqué)

Quasi-vivant

$< R, M_0 >$ est quasi-vivant si pour toute transition t, il existe $M \in GMA(R, M_0)$ tq :

$M \xrightarrow{t}$

(« Pas de code mort »)

Vivacité

$< R, M_0 >$ est quasi-vivant si pour tout $M \in GMA(R, M_0)$, <R, M> est quasi-vivant

(Pour tout les marquages, on regarde le réseau à partir du marquage)

Marquage d'accueil

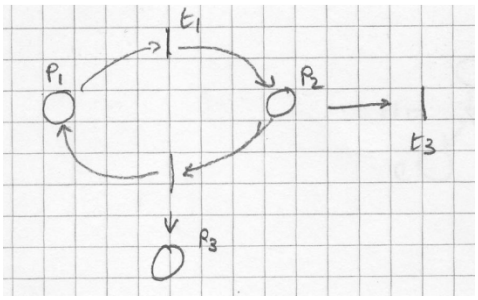
$< R, M_0 >$ admet un marquage d'accueil M_a si $\forall M \in GMA(R, M_0)$ il existe une séquence *s* tel que $M \xrightarrow{s} M_a$

Propriétés

- Si $< R, M_0 >$ est pseudo-vivant -> $< R, M_0 >$ admet une séquence infinie
- Si $< R, M_0 >$ est vivant -> {quasi, pseudo-vivant}
- Si $< R, M_0 >$ est quasi-vivant et admet M_0 comme marquage d'accueil : vivant

- °P : toutes transitions en entrée de P
- P° : toutes transitions en sortie de P
- °T
- T°

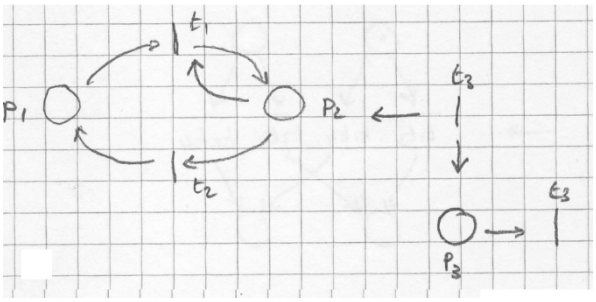
Verrou



Ensemble de place tel que :

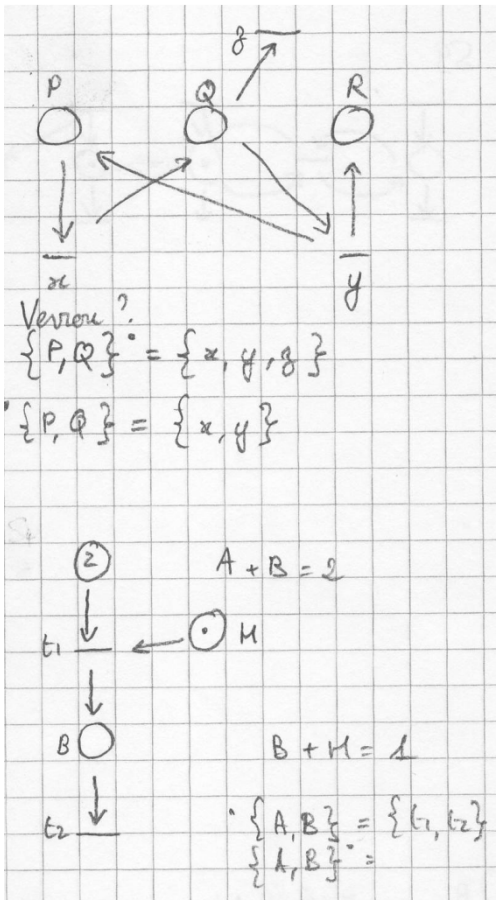
- $°V \leq V°$
- °{P1, P2, ~~P3~~} = {t1, t2}
- {P1, P2, ~~P3~~}° = {t1, t2, t3}
- <R, M0> vivant s'il ne contient pas de verrou

Trappe



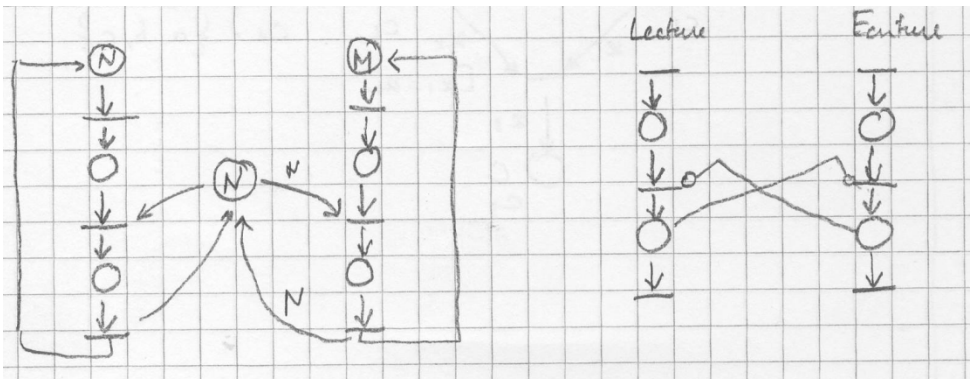
- $S° \leq °S$
- {P1, P2}° = {t1, t2}
- °{P1, P2} = {t1, t2, t3}

Exercice



- Les flots = invariants. On dit réseau de Petri safe.
- Flots : négatifs

Exercice



Arc inhibiteur : est ce que la place est libre
Comment faire pour savoir que je n'ai aucune lecture en cours, comment le tester ?

Le problème des philosophes

Le problème des philosophes et des spaghettis est un problème classique en informatique système. Il concerne l'ordonnancement des processus et l'allocation des ressources à ces derniers. Ce problème a été énoncé par Edsger Dijkstra[1].

La situation est la suivante :

- cinq philosophes (initialement mais il peut y en avoir beaucoup plus) se trouvent autour d'une table ;
- chacun des philosophes a devant lui un plat de spaghetti ;
- à gauche de chaque assiette se trouve une fourchette.

Un philosophe n'a que trois états possibles :

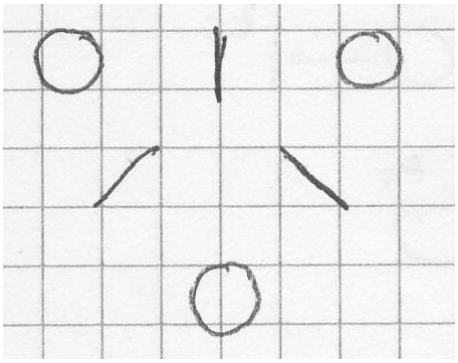
- penser pendant un temps indéterminé ;
- être affamé (pendant un temps déterminé et fini sinon il y a famine) ;
- manger pendant un temps déterminé et fini.

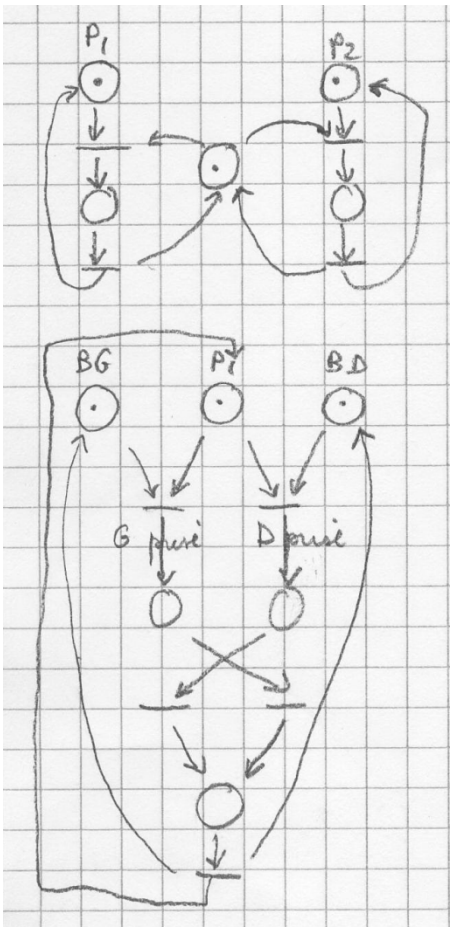
Des contraintes extérieures s'imposent à cette situation :

- quand un philosophe a faim, il va se mettre dans l'état « affamé » et attendre que les fourchettes soient libres ;
- pour manger, un philosophe a besoin de deux fourchettes : celle qui se trouve à gauche de sa propre assiette, et celle qui se trouve à gauche de celle de son voisin de droite (c'est-à-dire les deux fourchettes qui entourent sa propre assiette) ;
- si un philosophe n'arrive pas à s'emparer d'une fourchette, il reste affamé pendant un temps déterminé, en attendant de renouveler sa tentative.

Le problème consiste à trouver un ordonnancement des philosophes tel qu'ils puissent tous manger, chacun à leur tour. Cet ordre est imposé par la solution que l'on considère comme celle de Dijkstra avec sémaphores ou Courtois avec des compteurs.

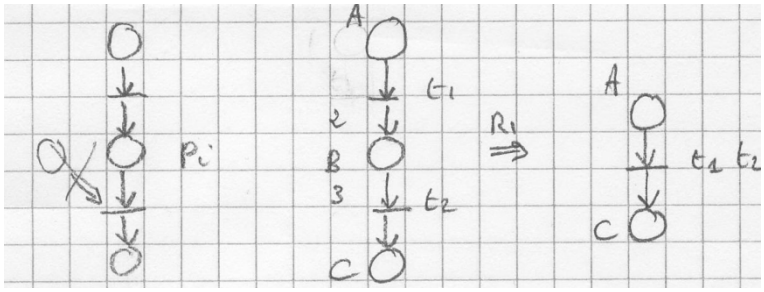
Source Wikipédia

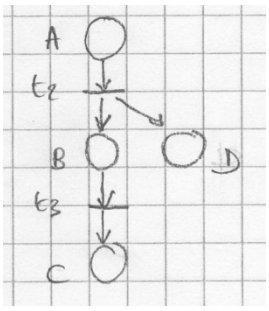


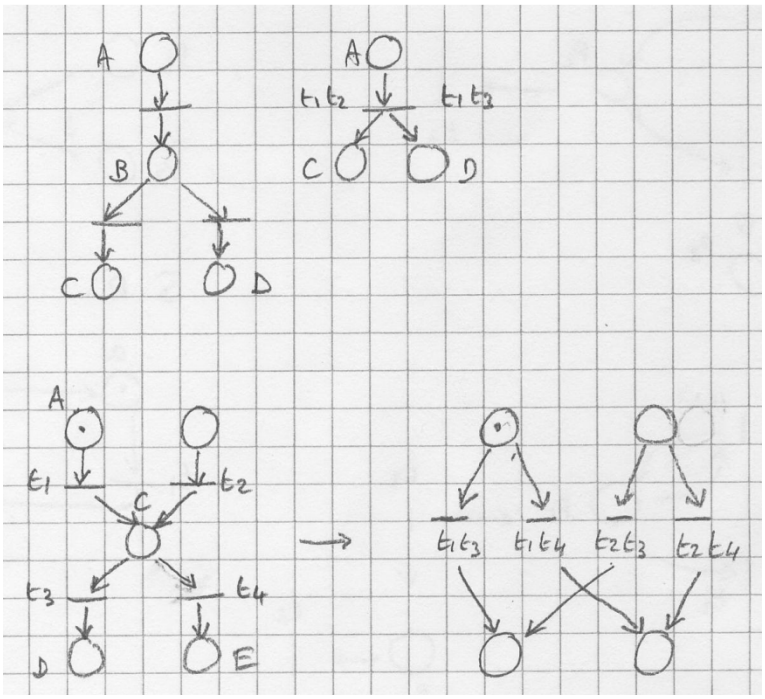


Réduction 1 : Place substituable (Pi)

1. $\forall T_j \in P_i^\circ\ ;\ {}^\circ T_j = \{P_i\}$
2. $P_i^\circ \cap {}^\circ P_i = \emptyset$
3. $\exists T_j \in P_i^\circ\ ; T_j \neq \emptyset$

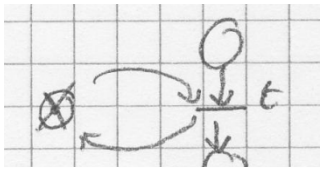






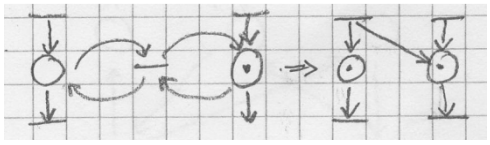
Réduction 2 : Place implicite (Pi)

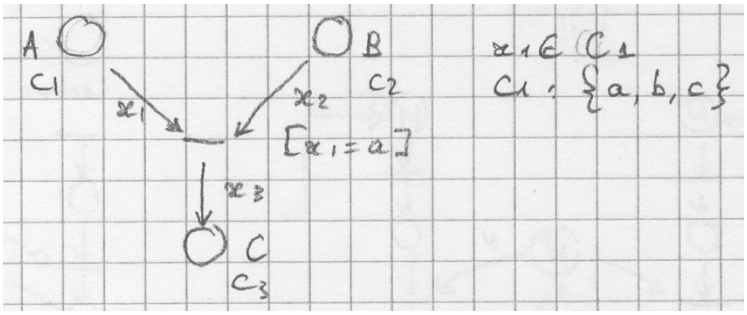
1. $\forall T_j \in P_i^\circ\ ;\ \forall P_b \in {}^\circ T_j\ \ \{Pi\}$
   $M(P_b) \geq Pre(P_k, T_j) \Rightarrow M(P_i) \geq Pre(P_i, T_j)$
2. Son marquage se déduit par : $M(P_i) = \sum_{k \neq i} \propto_k M(P_k) + \beta$

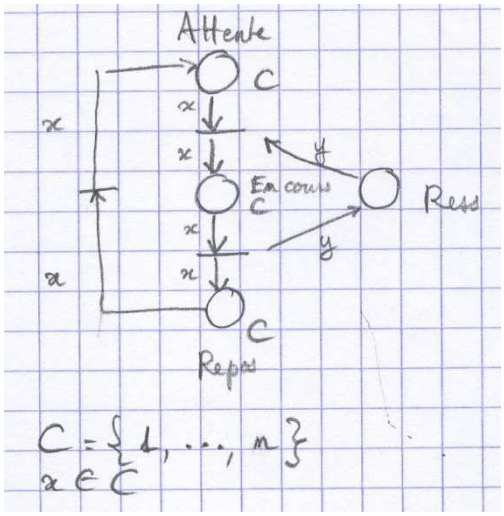


Réduction 3 : Transition neutre

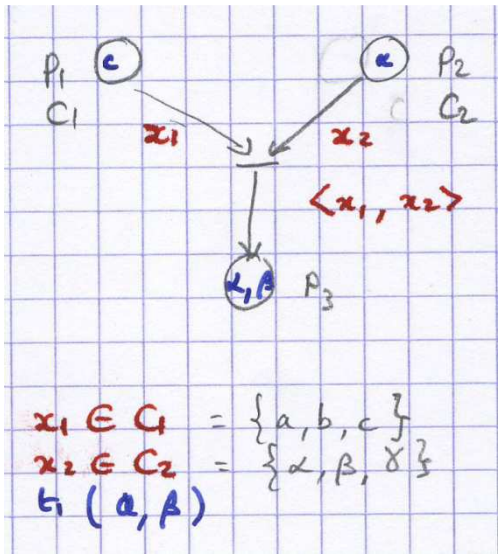
$^\circ T_i = T_i{}^\circ$

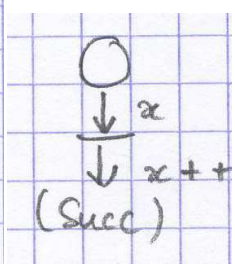






Successeur : « ++ »





Diffusion / Synchro