

Réseaux de Petri

Stéphane Louise

ENSTA

1 Introduction

Origine

Origine Carl Pétri mathématicien allemand contemporain. Les travaux sur les réseaux datent de 1960-1962.

Buts Modéliser des traitements parallèles. Modéliser les opérations de synchronisation.

Rechercher • Les blocages

- Les boucles
- Les propriétés

Outils à disposition • Représentation graphique.


- Calculs de type algèbre linéaire.





2 Représentation graphique

2.1 Convention graphique et utilisation

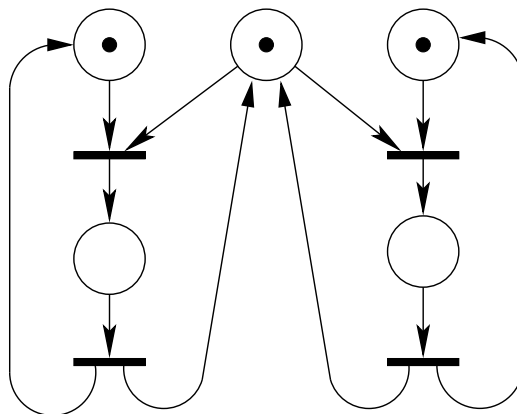
Représentation graphique

Définition formelle Un réseau de Petri est un 5-uplet: $PN = (P, T, F, Pre, Post) \cup M_0$ avec

- P ensemble de places 

- T ensemble de transitions 
- $F = Pre \cup Post$ ensemble d'arcs orientés avec $Pre \subseteq \{P \times T\}$ et $Post \subseteq \{T \times P\}$  
 - Pre est l'ensemble des arcs amont: relie les transitions aux places amont
 - $Post$ est l'ensemble des arcs aval: relie les transitions aux places aval
- M_0 vecteur de marquage initial. Correspond à un nombre de jetons (ou de marques) dans les places 

Exemple



Règles d'évolution

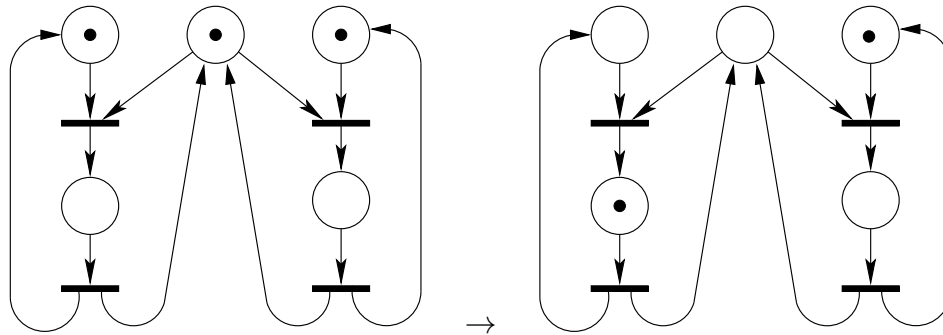
Définition 1 (Transition validée). On dit qu'une transition $t \in T$ est validée si et seulement si l'ensemble des places amont contient au moins un jeton

- On recherche toutes les transitions validées
- On choisit une transition t parmi celles qui sont validées
- Pour l'ensemble des places amont à la transition t : on retire un jeton
- Pour l'ensemble des places aval à la transition t : on ajoute un jeton

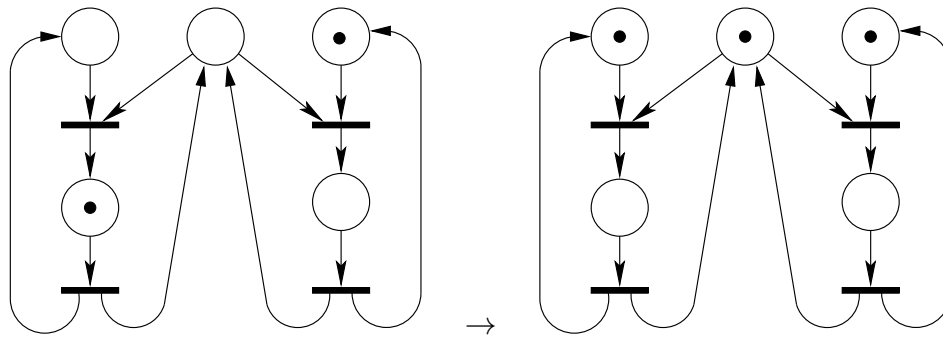
Il est interdit de tirer simultanément plusieurs transitions

Le tirage d'une transition est atomique Il n'existe pas d'état intermédiaire dans le tirage de la transition

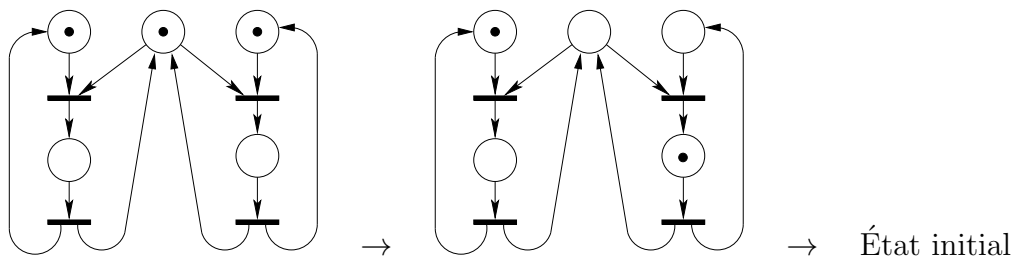
Exemple d'un tirage



Exemple d'un tirage, suite



Exemple d'un tirage, fin



Autres définitions

Definition 2 (Transition source). Une transition source est une transition sans place amont. Par construction, elle est toujours validée.

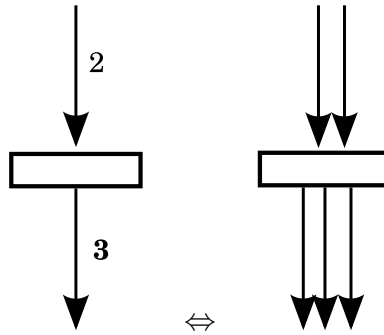
Definition 3 (Transition puits). Une transition puits est une transition sans place aval. Par construction, elle ne produit aucun jeton.

- Les transitions sont les actions qui font changer le système d'état
- Les places amont sont les conditions préalables à la réalisation de l'action

2.2 Variantes des RdP

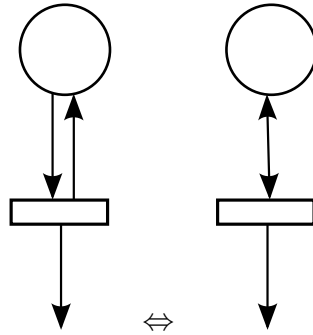
Variantes équivalentes

Definition 4 (Arcs pondérés). On ajoute un ensemble $W \subset \mathbb{N}^{*n}$ de poids sur les arcs: (a_i, ω_i) avec $a_i \in F$ et $\omega_i \in W$ de sorte que l'on consomme ω_i jetons dans la place amont si $a_i \in Pre$ ou que l'on ajoute ω_i jetons dans la place aval si $a_i \in Post$ N.B: si $\omega_i = 1$ on ne note pas le poids sur l'arc.



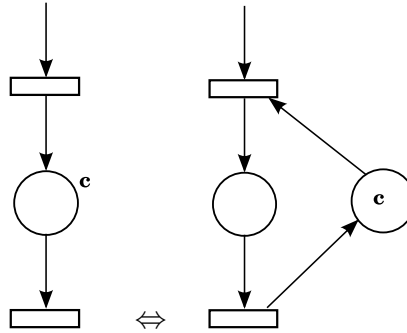
Test de présence de jeton

On peut faire une combinaison d'arcs $+1$ et -1 qui ne change pas le nombre de jeton dans une place par tirage, mais active une transition si et seulement si il y a au moins un jeton dans la place considérée.



Variantes équivalentes (suite)

Definition 5 (Réseau de Petri à capacité). On ajoute un ensemble $C \subset (\mathbb{N}^* \cup \{\infty\})^p$ de capacités sur les places: (p_i, c_i) avec $p_i \in P$ et $c_i \in C$ de sorte que les transitions amont soient invalidées si le nombre de jetons dans la place P_i noté $m_i = c_i$ N.B: si $c_i = \infty$ on ne note pas la capacité de la place



Variantes équivalentes (fin)

Definition 6 (Réseau de Petri coloré). • Des places et des transitions sont regroupées

- La couleur indique la place et la transition d'origine
- Toutes ces variantes sont simplement des notations plus compactes pour des RdP ordinaires
- On peut démontrer que les RdP ont une capacité d'expression supérieure aux machines d'états, mais inférieure aux machines de Turing

2.3 Interprétation et modélisation de systèmes

Interprétation des RdP

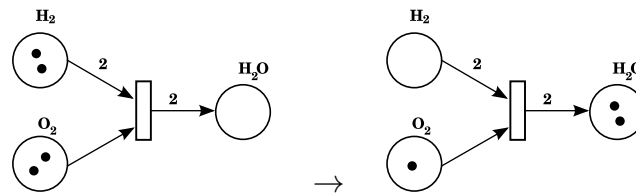
- **Places**

- Ressources
 - * nécessaires pour valider la transition
 - * produites (ou libérées) en franchissant la transition
- États prérequis

- **Transitions**

- Ensemble d'instructions
 - * à exécuter comme s'il n'y avait qu'un processus
 - * section critique ou point de synchronisation

Exemple:

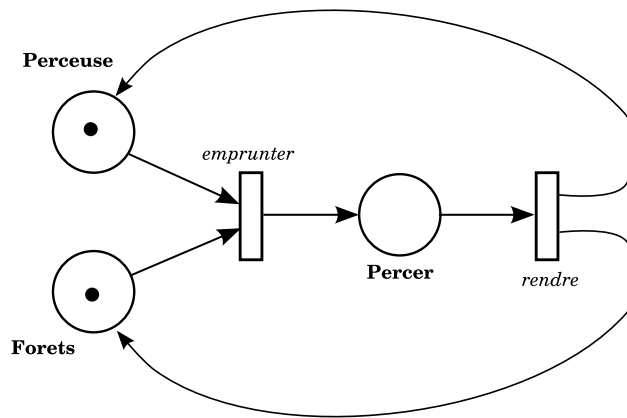


Un premier exemple élémentaire 1/4

Percer un trou nécessite d'aller au comité d'entreprise et d'emprunter:

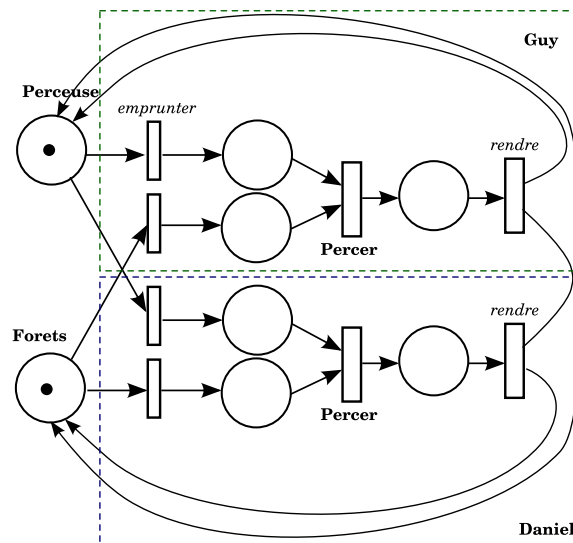
- Une perceuse
- Des forets

Ce sont des ressources nécessaires au perçage du trou. Après utilisation, les outils sont rendus au comité d'entreprise.



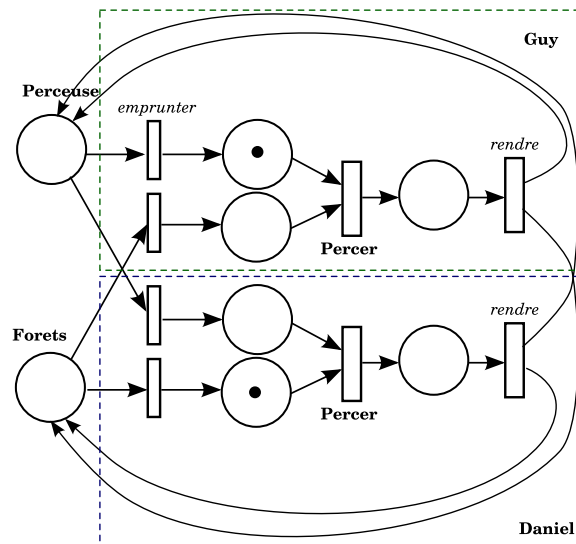
Un premier exemple élémentaire 2/4

On souhaite modéliser le cas de deux employés qui empruntent tous deux la perceuse et les forets au comité d'entreprise.



Un premier exemple élémentaire 3/4

Dans ce modèle que se passe-t-il si Guy emprunte la perceuse et Daniel emprunte les forets ?



Plus aucune transition n'est validée. Le système est bloqué.

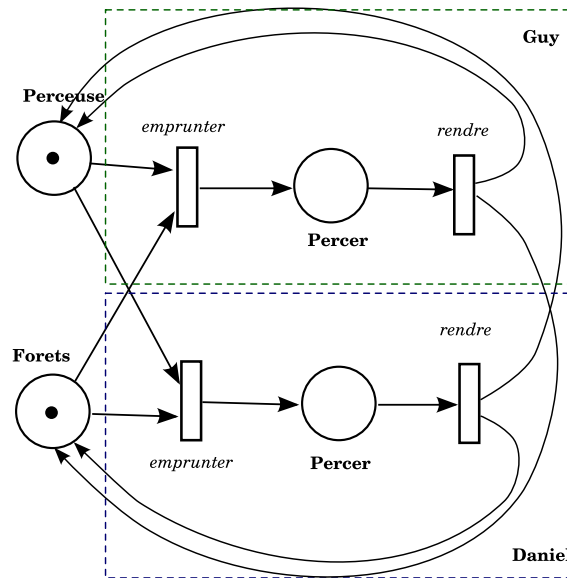
Conséquence

Definition 7 (Réseau mort, réseau bloqué). On dit qu'un réseau de Petri est mort ou bloqué si plus aucune transition n'est validée.

Un des intérêts des RdP est de pouvoir détecter l'existence de ces possibilités de comportements.

Un premier exemple élémentaire 4/4

Correction du modèle d'emprunt pour que la situation de blocage disparaisse



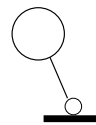
Une correction possible est que l'emprunt de la perceuse et des forets soit simultané

3 Extension des RdP

3.1 Arc inhibiteur et autres extensions

Arc Inhibiteur

Definition 8 (Arc inhibiteur).



- La transition n'est validée que lorsque la place est vide
- il sera dit aussi **test à zéro**.

En informatique le test à zéro est indispensable

On démontre que les réseaux de Petri étendus aux arcs inhibiteurs ont une capacité d'expression identique aux machines de Turing

Autres extensions

Definition 9 (Transitions ordonnées). • l'ensemble des transitions est muni d'une relation d'ordre

- si plusieurs transitions sont validées, c'est la plus petite qui est tirée

Definition 10 (Réseaux de Petri non autonomes). • à chaque transition est associé un événement ou le temps...

- la transition n'est tirable que si elle est validée **et** que si la condition est réalisée.

Ces extensions sont importantes pour leur capacité de simplification des modèles

3.2 Utilisation, preuve des sémaphores

Utilisation dans la preuve des sémaphores

Sémaphore

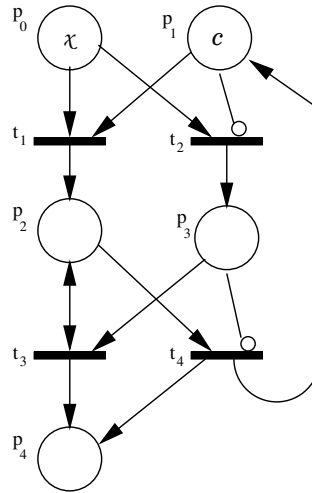
- **Objectifs**

- gérer un nombre de ressources disponibles
- gère des mises en attente et les réveils correspondants
- deux opérations :
 - * P : tenter d'occuper une ressource ou attendre
 - * V : libérer une ressource et réveiller un attendant

- **Contraintes**

- ne pas occuper plus de ressources qu'il n'y en a au départ
- n'attendre que s'il n'y a pas de ressources disponibles
- arriver à occuper au bout d'un temps fini

Réseau de Petri du sémaphore



Correspondances dans le modèle du réseau de Petri du sémaphore

- c le nombre de ressources
- les x processus sont en P_0 avant d'avoir occupé, en P_2 quand ils occupent, en P_4 après avoir occupé
- P_3 le nombre de processus en file d'attente
- la fonction P est représentée par les transitions t_1 et t_2
- la fonction V par les transitions t_3 et t_4 .

Programmation en C

- fonctions :

début_atomique()	fin_atomique()
bloquer_return()	debloquer_return()

```

static p1=c, p3=0;
void p() /* processus realisant P */
{
    /* acces au RdP */
    debut_atomique();

```

```

    if (p1 > 0) —p1; /* transition t1 */
    else /* transition t2 */ {
        ++p3; /*mettre le processus en file d'attente*/
        bloquer_return(); }
    fin_atomique();
}

```

Fonction V

```

/* processus faisant la fonction V */
void v()
{ /* accès au réseau */
    debut_atomique();
    if (p3 == 0) ++p1; /* transition t4 */
    else /*transition t3 */
    {
        --p3; /* prendre le processus en file d'attente */
        debloquer_return(); /* et le réactiver */
    }
    fin_atomique();
}

```

Remarque sur l'algorithme obtenu

Cet algorithme est identique à celui décrit dans les problèmes de parallélisme ; p_1 et p_3 sont remplacés par un compteur qui peut être négatif. La condition “*n’attendre que s’il n’y a pas de ressources disponibles*” est équivalente à

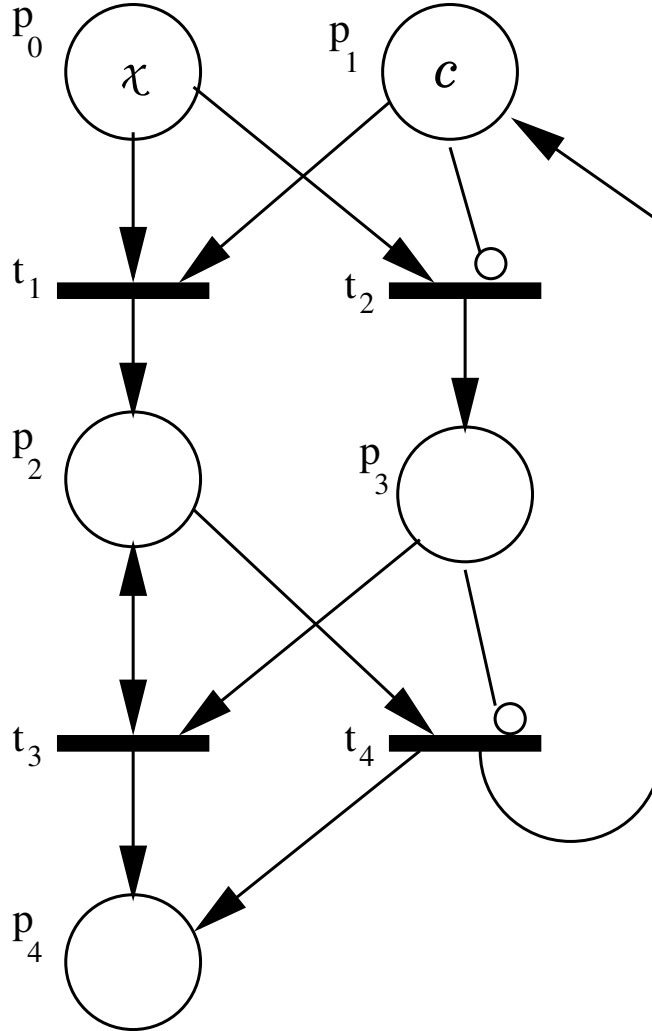
$$(p_3 = 0 \text{ ou } p_1 = 0)$$

c’est à dire

$$(p_3 > 0 \Rightarrow p_1 = 0)$$

3.3 Preuve du respect des contraintes

Preuve du respect des contraintes 1/4



Soit m_i le nombre de marques de la place P_i et n_i le nombre de fois que la transition t_i est franchie

Theorem 11. *Ne pas occuper plus de ressources qu'il n'y en a au départ*

Proof. • $m_1 + m_2 = c$

- vrai au départ, conservé par chaque transition, toujours vrai.

□

Preuve du respect des contraintes 2/4

Theorem 12. *N'attendre que s'il n'y a pas de ressources disponibles*

Proof. • $m_3 = 0 \vee m_1 = 0 \quad (m_3 > 0 \Rightarrow m_1 = 0)$

- si t_1 validée, on a $m_1 > 0$ donc $m_3 = 0$ avant et donc après
- si t_2 validée, on a $m_1 = 0$ avant et donc après
- si t_3 validée, on a $m_3 > 0$ donc $m_1 = 0$ avant et donc après
- si t_4 validée, on a $m_3 = 0$ avant et donc après.

□

Preuve du respect des contraintes 3/4

Theorem 13. *Occuper au bout d'un temps fini*

Proof. • Les processus sont dans l'une des places P_0 P_2 P_3 ou P_4 d'où $m_0 + m_2 + m_3 + m_4 = x$ vrai au départ, t_1 t_2 t_3 et t_4 laissent la somme inchangée donc reste vrai

- Avec le marquage initial, le réseau n'est pas vivant :
 - $n_1 + n_2 = x - m_0$ (donc $n_1 + n_2$ est majoré par x) : vrai au départ, t_1 t_2 t_3 et t_4 gardent la relation
 - $m_4 = n_3 + n_4$ (donc $n_3 + n_4$ est majoré par x) vrai au départ, t_1 t_2 t_3 et t_4 gardent la relation

□

Preuve du respect des contraintes 4/4

Theorem 14. *Le réseau est bloqué en au plus $2x$ tirages*

Proof. • Dans l'état bloqué $m_0 = m_2 = m_3 = 0$, $m_1 = c$, $m_4 = x$

- si $m_0 \neq 0$ alors t_1 ou t_2 est validée
- si $m_2 \neq 0$ alors t_3 ou t_4 est validée
- $(m_2 = 0 \wedge m_1 + m_2 = c) \Rightarrow m_1 = c$
- $((m_3 > 0 \Rightarrow m_1 = 0) \wedge m_1 = c) \Rightarrow m_3 = 0$
- $m_0 + m_1 + m_2 + m_3 + m_4 = x \Rightarrow m_4 = x$

□

En au plus $2x$ tirages, tous les processus ont occupé puis libéré une ressource

Remarques

- **Activités cycliques**

- p_4 est ramené sur p_0
- Le réseau devient vivant et toutes les transitions sont vivantes. Il manque une information pour réaliser la vivacité. Selon la file d'attente, celle-ci pourra ne pas être réalisée.

- **Propriétés en cours de fonctionnement**

$$- \begin{cases} m_0 + m_2 + m_3 + m_4 & = & x \\ n_1 + n_2 & = & x - m_0 \\ m_4 & = & n_3 + n_4 \\ n_3 + n_4 + m_2 + m_3 & & \end{cases} \Rightarrow n_1 + n_2 =$$

- *i.e.*: nombre de P égal nombre de V + nombre de processus en section critique ou en attente

- **En fin d'activité**

- $m_2 = m_3 = 0$ donc $n_1 + n_2 = n_3 + n_4$
- *i.e.*: nombre de P égal nombre de V

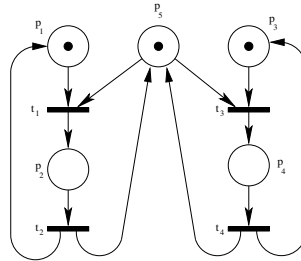
4 Étude des propriétés

4.1 Notations

Notations

On numérote les places de 1 à p , les transitions de 1 à T

Definition 15 (Marquage). Un marquage est un vecteur de \mathbb{N}^p dont les composantes donnent le nombre de marque dans chaque place correspondante



- $M_0 = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \end{pmatrix}$ est le *marquage initial*

Marquages accessibles

- t_1 et t_3 sont validées
- $M_0(t_1 \rightarrow M_1 = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \end{pmatrix}$
- $M_0(t_3 \rightarrow M_2 = \begin{pmatrix} 1 & 0 & 0 & 1 & 0 \end{pmatrix}$
- $M_1(t_2 \rightarrow M_0$
- $M_2(t_4 \rightarrow M_0$
- $M_0(t_1 t_2 \rightarrow M_0$
- $M_0(t_1 t_2 t_3 \rightarrow M_2$ séquence de tirage t_1 puis t_2 puis t_3
- si $S = t_1 t_2 t_3$ alors $M_0(S \rightarrow M_2$
- $*M_0 = \{M_0, M_1, M_2\}$ ensemble des marquages accessibles depuis M_0

Définitions complémentaires

Définition 16 (Transition vivante pour un marquage initial). Quelle que soit l'évolution du système, il existera toujours une suite de transitions qui la validera

Définition 17 (Réseau vivant pour un marquage initial). Aucune transition ne peut devenir infranchissable


Définition 18 (Réseau sans blocage pour un marquage initial). Quelle que soit l'évolution du système, il existera toujours une transition validée

Conflit

Définition 19 (Conflit). Une place est en amont de 2 transition validées (ou plus):



C'est un comportement indéterminé, en général non réaliste sur un ordinateur

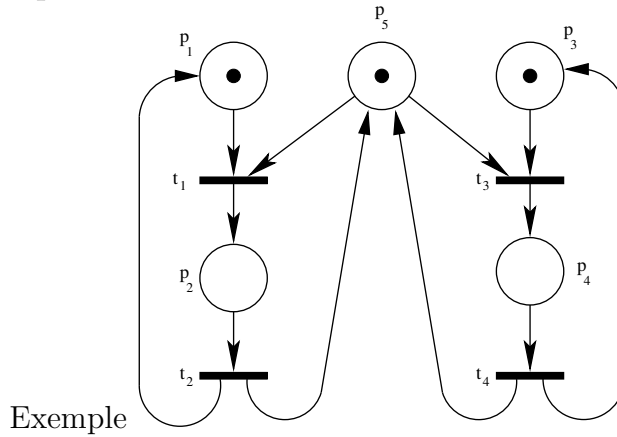
- conflit effectif avec persistance 

RdP persistant pour un marquage initial M_0

Definition 20 (RdP persistant pour un marquage initial M_0). Quelque soit M_i appartenant à $*M_0$, quelque soient t_j et t_k validées par M_i alors $t_j t_k$ est une séquence de franchissement à partir de M_i .

Le fait de faire un choix ne supprime pas l'alternative

Composantes conservatives



- p_5 représente les ressources disponibles
- p_2 et p_4 représente les ressources utilisées
- $M[5] + M[2] + M[4] = Cte(M_0)$

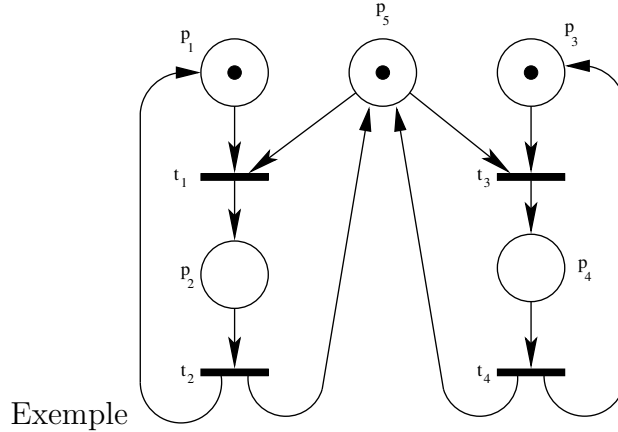
Composante conservative, définition

Definition 21 (Composante conservative). • Soit $P = \{p_1, p_2, \dots, p_n\}$ un sous-ensemble de places

- Soit $q = (q_1 \ q_2 \ \dots \ q_n)$ un vecteur de pondération
- On dit que $\sum_{i=1}^n q_i M[i]$ est une **composante conservative** ssi sa valeur ne dépend que de l'état initial.

n.b.: P et q dépendent du réseau

Séquence répétitive



- A partir de M_0 , t_1 puis t_2 ou t_3 puis t_4 ramènent sur M_0 $M_0(t_1t_2 \rightarrow M_0$
et $M_0(t_3t_4 \rightarrow M_0$
- C'est une propriété du réseau et du marquage initial (*ex.* devient faux
si $M_0[p_5] = 0$)

Séquence répétitive, définition

Definition 22 (Séquence répétitive). Une séquence est dite répétitive si elle ramène sur M_0 le marquage initial.

Definition 23 (Séquence répétitive complète). On dit qu'une séquence répétitive est complète si elle contient toutes les transitions

$$t_1t_2t_3t_4$$

4.2 Approche lexicale

Représentation lexicale

- soit un alphabet $T = \{T_i\} \ 1 \leq i < N$
- une expression régulière est une suite de lettres
- $T_1 + T_2$ représente la lettre T_1 **ou** la lettre T_2
- T_1T_2 représente la lettre T_1 suivie de la lettre T_2

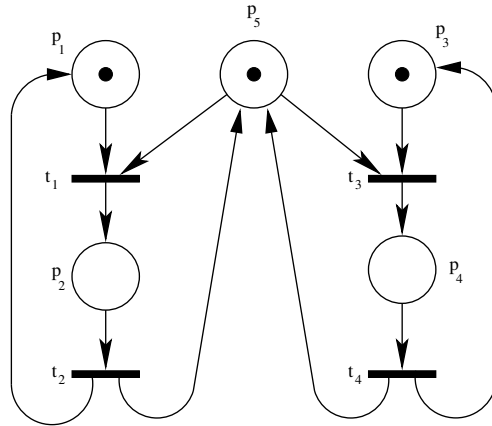
- $T_1T_2T_3$ est une séquence de longueur 3
- \dagger représente la séquence de longueur 0
- $T_1^* = \dagger + T_1 + T_1T_1 + T_1T_1T_1 + \dots$ représente l'itération de T_1 un nombre quelconque de fois

Quelques règles

- la notation S_i est donnée aux expressions régulières
 - $S_1 (S_2 + S_3) = S_1S_2 + S_1S_3$
 - $(S_1 + S_2) S_3 = S_1S_3 + S_2S_3$
- si $S_1 = S_2S_3$ on dira que S_2 est un préfixe de S_1
- $T^* = (\dagger + T_1 + T_2 + \dots + T_n)^*$ représente l'ensemble des séquences du monoïde construit sur l'alphabet T .

Pour un réseau de Pétri avec son marquage initial, l'ensemble L des séquences de franchissement à partir de M_0 est un sous-ensemble du monoïde T^* construit sur les transitions.

Exemple



- $S_1 = t_1t_2$
- $S_2 = t_3t_4$ sont répétitives

- $t_1t_2 + t_3t_4$ est la plus petite séquence complète répétitive
- $L = (t_1t_2 + t_3t_4)^* (\ddagger + t_1 + t_3)$ car ce sont les répétitions suivies des préfixes.

Séquence répétitive croissante

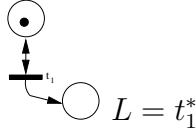
Definition 24 (Séquence répétitive croissante). • Soient 2 marquages M_1 et M_2 Si pour tout i , $M_1[p_i] > M_2[p_i]$ on dira que $M_1 > M_2$

- pour un RdP ordinaire (sans arc inhibiteur) $L_2 \subset L_1$

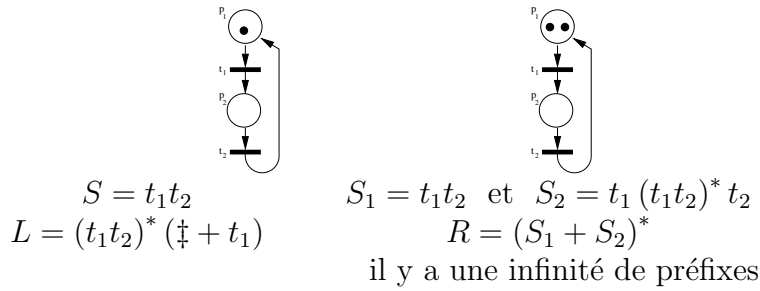
- Soit une séquence S telle que $M[M_o(S)] > M[M_0]$ alors S est dite **Séquence répétitive croissante**.

Remarques :

- si L est fini alors *M_0 est fini donc borné
- si L est infini alors *M_0 peut être fini ou infini
- si L est infini, il faut que certaines séquences contiennent des cycles



Avance synchronique



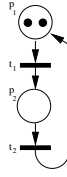
Les préfixes sont de la forme :

$$t_1 + t_1t_1 + t_1t_1t_2 + t_1t_1t_2t_1 + t_1t_1t_2t_1t_2 + \dots$$

Il est plus intéressant de compter le nombre de fois que figurent t_1 et t_2

Avance synchronique, définition

- Definition 25** (Avance synchrone). • Soit S_k une séquence de franchissement
- Soit $n_k(t_i)$ le nombre de fois que t_i apparaît dans S_k
 - $n_k(t_i) - n_k(t_j)$ s'appelle l'**avance synchronique de t_i sur t_j**



Dans ce cas, il est clair que

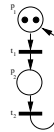
$$0 \leq n(t_1) - n(t_2) \leq 2$$

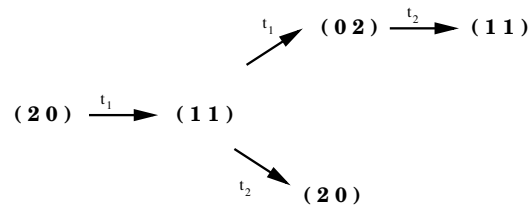
L'avance synchronique de t_1 sur t_2 est majorée par 2, et t_2 ne peut pas avoir d'avance synchronique sur t_1 .

4.3 Graphe de marquage

Graphe de marquage

- Definition 26** (Graphe de marquage). • Soit un RdP et son marquage initial.
- Les **noeuds** du graphe de marquage sont les marquages accessible : les **éléments de $*M_0$**
 - Les **arcs** sont les transitions qui permettent de **passer d'un marquage à un autre**





$$M_0 = \begin{pmatrix} 2 & 0 \end{pmatrix}$$

$$M_1 = \begin{pmatrix} 1 & 1 \end{pmatrix}$$

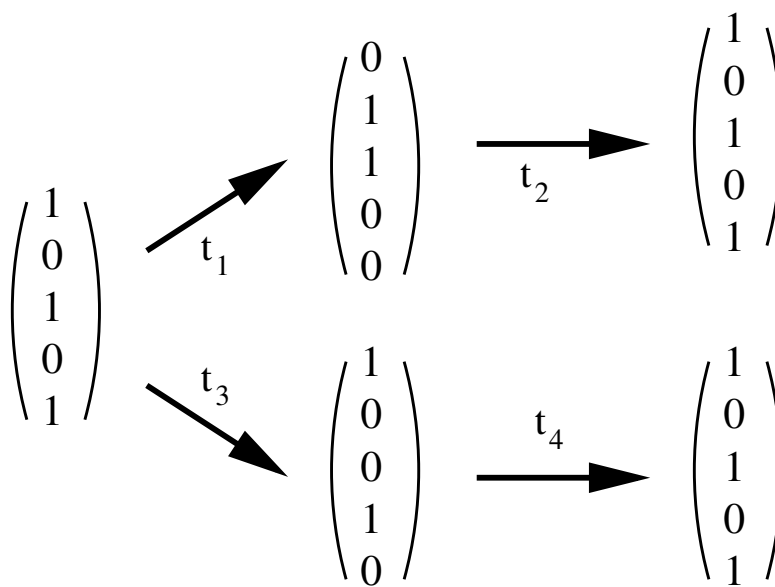
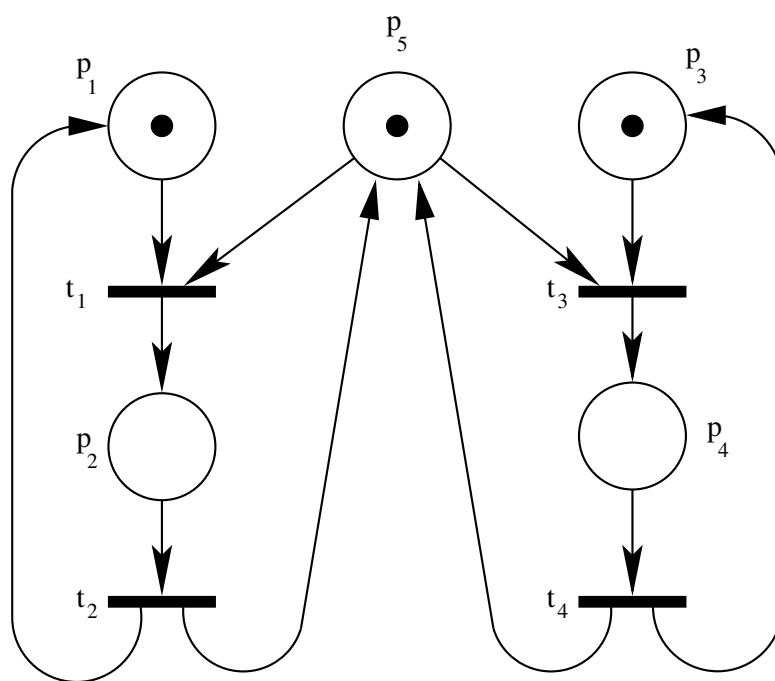
$$M_2 = \begin{pmatrix} 0 & 2 \end{pmatrix}$$

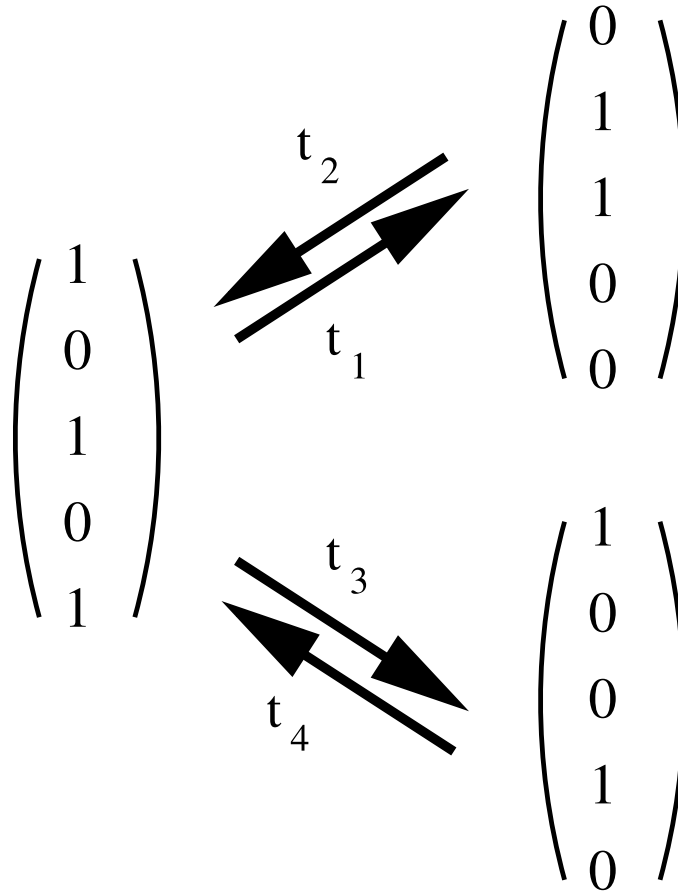
Construction des graphes de marquage

Construction

- **initialisation** porter le noeud M_0 , tirer les transitions validées et porter les arcs et les nœuds correspondants.
- **itération** soit M_i un noeud qui vient d'être obtenu.
 1. Si sur le chemin $M_0 \rightarrow M_i$ il existe un noeud M_j tel que $i \neq j$ et $M_i = M_j$, alors M_i n'a pas de successeur.
 2. Sinon tirer les transitions validées par M_i et on obtient ainsi les noeuds successeurs de M_i .

Exemple



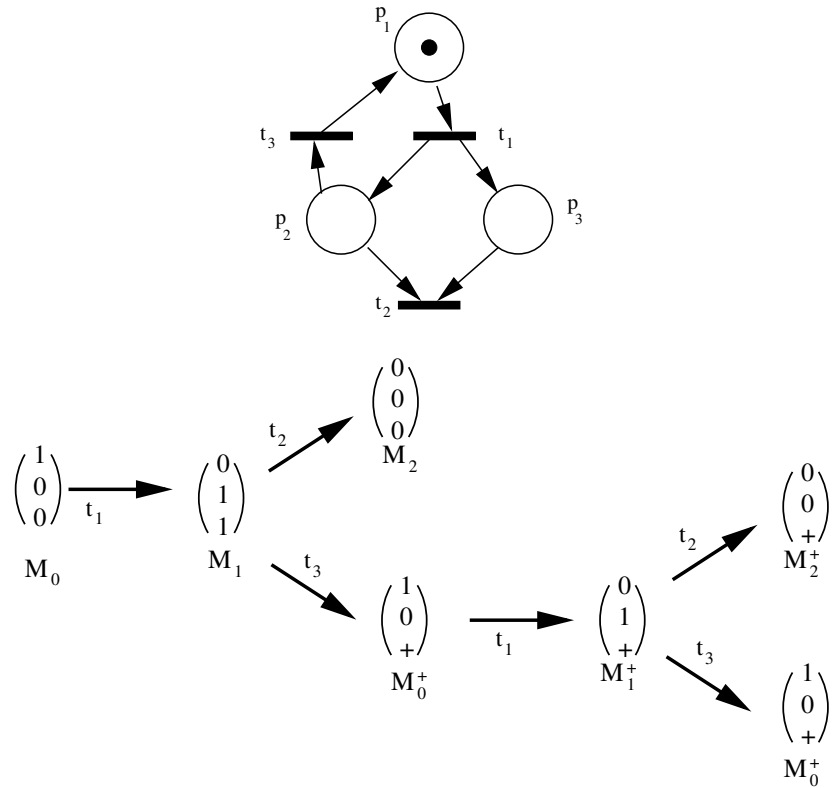


Cas de $*M_0$ infini

- initialisation porter le noeud M_0 , tirer les transitions validées vers les M_i et porter les arcs et les noeuds correspondants.
 - Si $M_i > M_0$ alors remplacer par + les composantes de M_i strictement plus grandes.
- itération soit M_i un noeud qui vient d'être obtenu.
 1. si sur le chemin $M_0 \rightarrow M_i$ il existe un noeud M_j tel que $j \neq i$ et $M_i = M_j$ alors M_i n'a pas de successeur.
 2. sinon tirer les transitions validées par M_i et on obtient ainsi les noeuds M_k successeurs de M_i :

- une composante + reste +
- sur le chemin $M_0 \rightarrow M_k$ il existe un noeud M_j tel que $j \neq k$ et $M_k > M_j$ alors remplacer par + les composantes de M_k strictement plus grandes.

Exemple

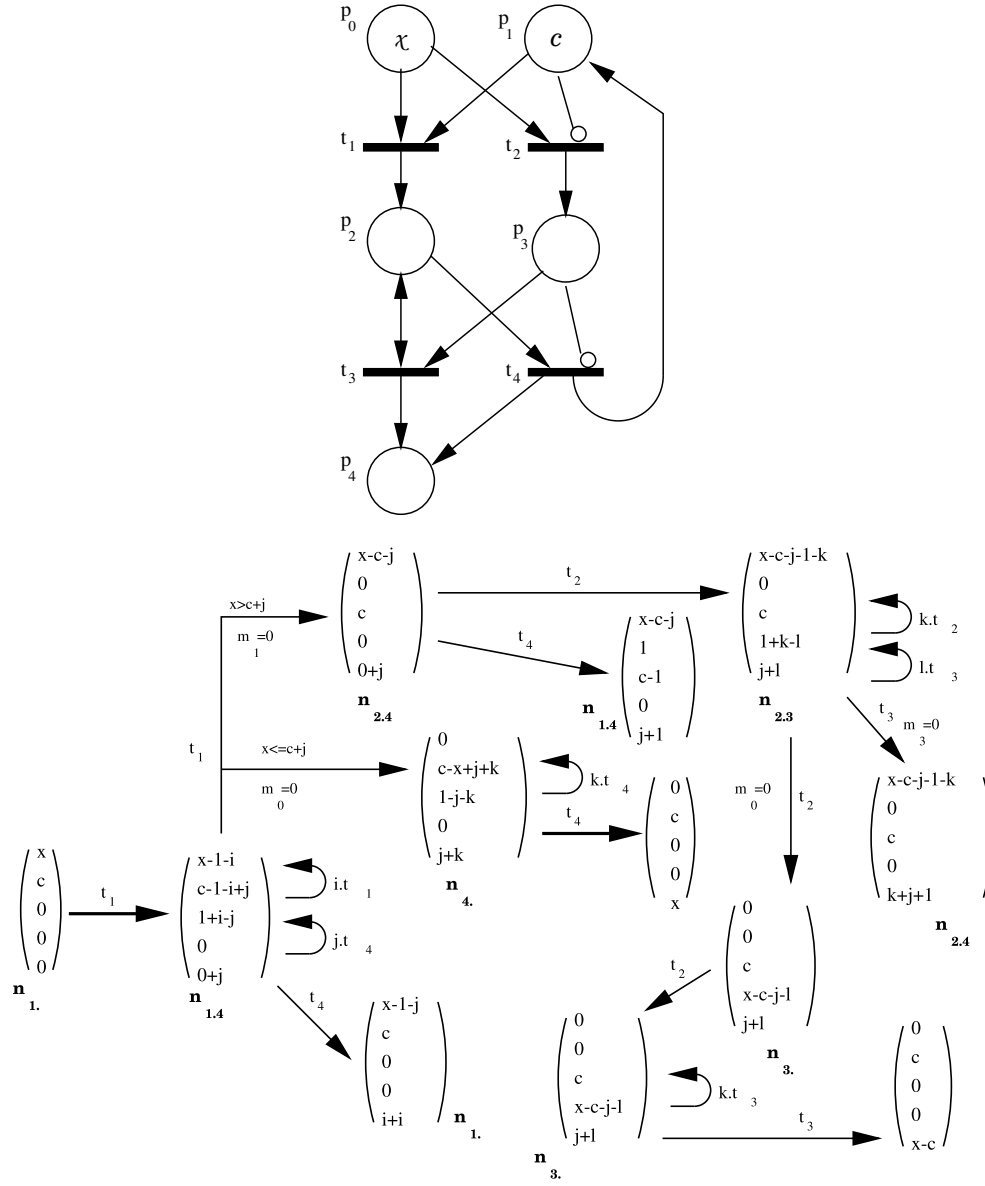


Places paramétrées

- Problème de représentation Le nombre d'éléments de $*M_0$ peut dépendre des paramètres
- **Représentation approchée**
 - les noeuds sont identifiés par les transitions qui sont validées $n_{1,3}$ ou n_2 ou $n_{3,5,9}$

- les arcs qui ne font pas changer de noeuds portent un indice d'itération
- si les nouvelles places portent un nom existant, elles n'ont pas de successeur.

Exemple



5 Algèbre linéaire

5.1 Définition

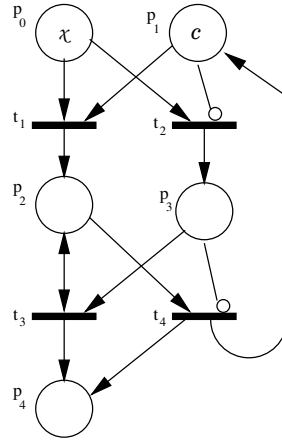
Définition

Définition 27 (Matrice d'incidence avant W^-). La matrice d'incidence avant W^- : t colonnes et p lignes: $W_{ij}^- = Pre(i, j)$ nombre de marques de la place i consommées par la transition j

Définition 28 (Matrice d'incidence après W^+). La matrice d'incidence après W^+ : t colonnes et p lignes: $W_{ij}^+ = Post(i, j)$ nombre de marques de la place i produites par la transition j

Définition 29 (Matrice d'incidence). Matrice d'incidence : $W = W^+ - W^-$
 W_{ij} est la variation de marques produites dans la place i par la transition j

Exemple



$$W^- = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad W^+ = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

$$W = \begin{pmatrix} -1 & -1 & 0 & 0 \\ -1 & 0 & 0 & 1 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix}$$

$$M_0 = \begin{pmatrix} x & c & 0 & 0 & 0 \end{pmatrix}$$

Équation fondamentale

- soit M_0 un marquage initial
- soit $S = t_i^{n_i} t_j^{n_j} \dots t_k^{n_k}$ une séquence de franchissement de *M_0
 - \underline{S} vecteur caractéristique de S
 - possède t composantes
 - $S[i] =$ nombre de fois que la transition t_i est franchie dans la séquence S
 - *n.b.: différentes séquences peuvent avoir la même caractéristique. toutes les caractéristiques ne sont pas possibles*
- **Équation** : si $M_0(S \rightarrow M_k)$ alors

$$M_k = M_0 + W.S$$

Exemple

exemple :

$$M_k = \begin{pmatrix} x \\ c \\ 0 \\ 0 \\ 0 \end{pmatrix}^{\dagger} + \begin{pmatrix} -1 & -1 & 0 & 0 \\ -1 & 0 & 0 & 1 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} n_1 \\ n_2 \\ n_3 \\ n_4 \end{pmatrix}$$

REMARQUES

- les arcs inhibiteurs,
- les extensions de conditionnement...

Jouent sur la validité d'une séquence S

Ne jouent pas sur l'équation de base : $M_k = M_0 + W.S$

5.2 Calcul des invariants

Composantes conservatives

$F \in L(P, N)$ soit $F = (q_1 \ q_2 \ \dots \ q_p)$ de

$$M_k = M_0 + W.S$$

en multipliant par F à gauche

$$F.M_k = F.M_0 + F.W.S = Cte$$

quelque soit M_k appartenant à $*M_0$

Theorem 30. *F est une composante conservative pour tout marquage initial à la condition suffisante que $F.W = 0$*

Exemple

$$F.W = 0 \implies \begin{cases} (1) & -q_0 - q_1 + q_2 = 0 \\ (2) & -q_0 + q_3 = 0 \\ (3) & -q_3 + q_4 = 0 \\ (4) & q_1 - q_2 + q_4 = 0 \end{cases}$$

- (1) $\Rightarrow q_2 = q_0 + q_1$
- (2) $\Rightarrow q_3 = q_0$
- (3) $\Rightarrow q_4 = q_0$ et (4) est vérifiée

donc $F = (q_0 \ q_1 \ q_0 + q_1 \ q_0 \ q_0)$

- $(q_0 \ q_1) = (1 \ 0) \implies m_0 + m_2 + m_3 + m_4 = x$ conservation des processus
- $(q_0 \ q_1) = (0 \ 1) \implies m_1 + m_2 = c$ conservation des ressources

Remarques

$$F.M_k = F.M_0 + F.W.S$$

pour certains réseaux et certains marquages initiaux, on peut avoir $F.W.S = 0$ avec $F.W \neq 0$

- si quelques transitions sont mortes
- s'il n'y a pas de cycles

Linéarité

Si F_1 et F_2 sont deux composantes conservatives, alors $aF_1 + bF_2$ est une composante conservative, *sous réserve que les marquages correspondants soient atteignables*.

Séquences répétitives

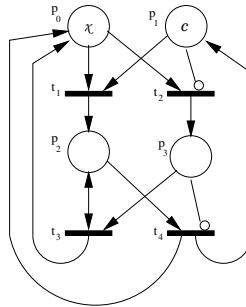
- $M_0(S \rightarrow M_k$ alors $M_k = M_0 + W.S$
- pour une séquence répétitive, $M_k = M_0$

Theorem 31. *Toute séquence répétitive vérifie $W.S = 0$*

Remarques

Les solutions de $W.S = 0$ peuvent ne pas avoir de séquence atteignable à partir de M_0 .

Exemple



$$W = \begin{pmatrix} -1 & -1 & 1 & 1 \\ -1 & 0 & 0 & 1 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \end{pmatrix}$$

$$\bullet \begin{cases} (1) & -n_1 - n_2 + n_3 + n_4 = 0 \\ (2) & -n_1 + n_4 = 0 \\ (3) & n_1 - n_4 = 0 \\ (4) & n_2 - n_3 = 0 \end{cases}$$

- (2) $\Rightarrow n_4 = n_1$
 - (4) $\Rightarrow n_3 = n_2$
 - (1) et (3) sont alors vérifiées

$$S = \begin{pmatrix} n_1 & n_2 & n_2 & n_1 \end{pmatrix}$$

Remarques

- $n_4 = n_1$ on doit rendre toutes les ressources qu'on a occupées
- $n_3 = n_2$ on doit sortir de file d'attente les processus qu'on y a mis
- $S = \begin{pmatrix} 0 & 1 & 1 & 0 \end{pmatrix}$ ne correspond à aucune séquence accessible

Interprétation de la méthode du pivot pour résoudre les équations

Les lignes (places) touchées par une transition sont remplacées par des combinaisons invariantes pour la transition (coefficient nul).

- les petits systèmes se résolvent à la main
- les grands systèmes se traitent par ordinateur

6 Utilisation

6.1 Programmation POSIX

Position du problème

- Mémorisation du réseau

- Accès atomique au réseau pour tirer une transition
- Comment transposer le réseau dans les outils du système
 - association processus du RdP et tâche du système
 - comment exécuter le code associé aux places et aux transitions
 - comment une tâche se met-elle en attente
 - comment réveiller une tâche

Analyse des outils

1. Comment créer des tâches
2. Liste de fonctions permettant à une tâche
 - de se mettre en attente
 - de réveiller une tâche
3. Fonctionnalités permettant de partager des données
4. Interface avec l'environnement
 - problème des philosophes
 - calculateur SUN ou Linux (POSIX) comme outil

POSIX de base

1. deux possibilités
 - fonction **fork()**
 - lancement d'un programme
2. trois possibilités pour chaque
 - (a) mise en attente
 - **sleep**(delai)
 - **pause()** : attente d'un signal

- les opérations d'entrée/sortie, notamment **read()** et **write()**

(b) réveil

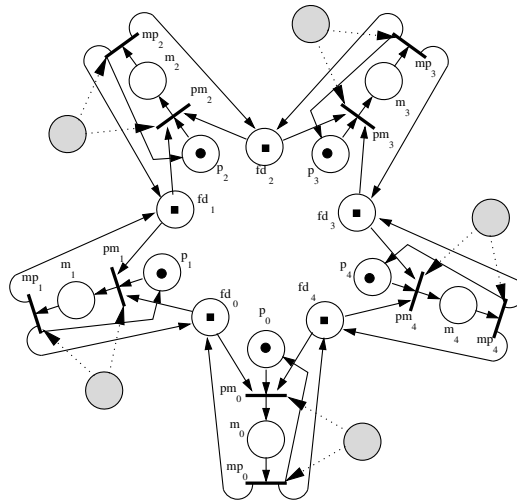
- délai écoulé
- envoi d'un signal avec la fonction **kill**(sig, pid)
- la fin d'entrée/sortie (par un *write* ou un *read* pour les tubes)

3. partage de données avec le système de fichiers

- permanent sur le disque
- temporairement avec le tampon d'un tube hérité ou nommé (FIFO)

4. avec l'opérateur par le terminal

Modélisation du problème des 5 philosophes



Programmation: déclarations

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#define MPHILO 5
```

```

typedef struct {
    char plc;
    char penser;
    char manger;
    char fourchd; } PHILO;
struct { PHILO table[MPHILO];
    int chaise;
    char plibre, poccm1, pvalid; } petri;

```

Programmation: déclarations 2

```

int nuph= -1, fpr, fpw ,fdp[MPHILO], frep; /* pour l'initialisation */
char buf[ ]="fifo?";
#define MODE 0010666 /* FIFO et RW-RW-RW- */

    /* abréviations pour désigner les places */
#define pp(i) petri.table[(i+MPHILO)%MPHILO].penser
#define pm(i) petri.table[(i+MPHILO)%MPHILO].manger
#define pf(i) petri.table[(i+MPHILO)%MPHILO].fourchd
#define pc(i) petri.table[(i+MPHILO)%MPHILO].plc
    /* descripteur de fichier désignant un philosophe */
#define fd(i) fdp[(i+MPHILO)%MPHILO]
    /* texte pour le dialogue */
char tamp[80]; char baniere[ ]= ".";
char *mes[2]={ "Vous pensez\n", "Vous mangez\n"};

```

Programmation: gestion parallèle

```

petriph()
{
    int i, j, flag;
    read(fpr, &petri, sizeof(petri)); /*prendre le RdP */
    ++pc(nuph); /* mettre la marque de demande du philosophe */
    do {
        flag= 0;
        for (j= 0, i= nuph; j < MPHILO; i++, j++) {
            if (pc(i) AND pf(i-1) AND pp(i) AND pf(i)) {
                ++flag; /* une transition est trouvée */
                --pc(i); --pf(i-1); --pp(i); --pf(i);
                ++pm(i);
                write(fd(i), "m", 1); /* réveil d'un philosophe */ }
            if (pc(i) AND pm(i)) {
                ++flag;
                --pc(i); --pm(i);
                ++pf(i-1); ++pp(i); ++pf(i);
                write(fd(i), "p", 1); /* réveil d'un philosophe */ } }
        while (flag); /* tourner tant qu'on trouve une transition */
    write(fpw, &petri, sizeof(petri)); /* rendre le RdP */
    /* attendre que la marque de commande soit consommée */
    read(frep, tamp, 1);
}

```

}

Problème de la gestion des processus

INITIALISATION

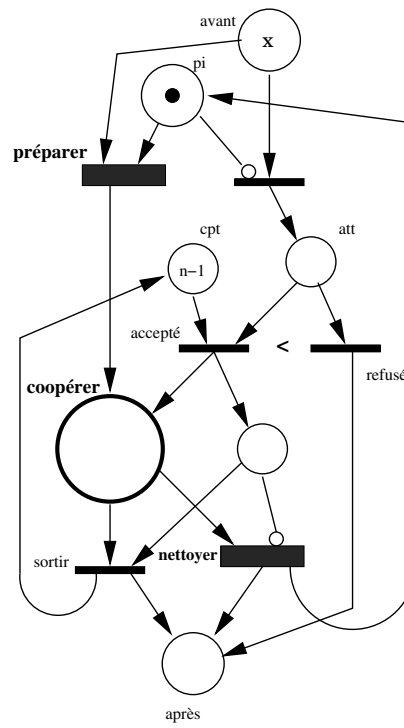
- le calcul du numéro de chaise
- les ouvertures des différentes FIFO
- la création des FIFO dans le système de fichier

FIN DU PROGRAMME

- fermeture des descripteurs de fichier du philosophe qui part
- libération de la chaise
- suppression des FIFO au départ du dernier philosophe

REMARQUES Dans POSIX, il est possible de supprimer le nom d'un objet (fichier généralisé) pendant qu'il est employé. L'objet n'est plus "ouvrable" mais reste utilisable. Il est supprimé quand il n'est plus employé.

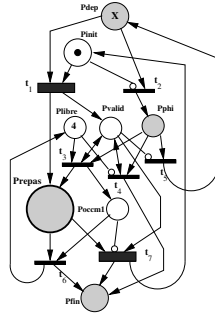
Modèle de processus



Principe de réalisation

- La place Pinit est associée à l'existence de fifop sur le système de fichier
- t_1 et t_2 correspondent à des voies différentes dans le début du programme
- Pphi et Prepas sont des positions dans le code d'initialisation
- le choix du tirage de $(t_3, t_4$ ou $t_5)$ ou $(t_6$ ou $t_7)$ est réalisé par l'accès au réseau et la position dans le code

Elargissement du RdP



- Pdep et Pfin sont les places correspondant au démarrage et à la fin du code. Il peut être lancé autant de fois que désiré.
- noter Plibre de poids 4
 - $t_4 \rightarrow Pfin$ quand la table est pleine
 - $t_5 \rightarrow Pdep$ quand il y a eu la suite t_2, t_7

Programme Main

```
main()
{
    int r, i;
    Pdep:
    T1T2: /* mknod fait —Pinit ou teste Pinit */
    if((r= mknod("fifop",MODE, 0)) > 0) goto T1;
    else if(errno == EEXIST) goto Pphi; /* fin de T2 */
    printf("erreur en creation de fifop, errno=%d\n", errno);
    exit(-2);
    T1:
    for(i= 0; i < MPHILO; ++i) {
        buf[4]= '0'+i;
        mknod(buf, MODE, 0); }
    fpr= open("fifop", O_NDELAY|O_RDONLY, 0);
    fpw= open("fifop", O_WRONLY, 0);
    fcntl(fpr, F_SETFL, O_RDONLY);
    nuph= 0; petri.chaise= ((1<<MPHILO)-1)&~(1<<nuph);
    petri.plibre= MPHILO-1; petri.pocm1= 0; petri.pvalid= 1;
    for(i= 0; i < MPHILO; ++i)
        {pc(i)= 0; pp(i)= 1; pm(i)= 0; pf(i)= 1;}
    write(fpw, &petri, sizeof(petri)); /* libere le RdP */
    goto Prepas; /* fin de T1 */
    Pphi:
    if((fpr= open("fifop", O_NDELAY|O_RDONLY, 0)) < 0) goto Pdep;
    if((fpw= open("fifop", O_WRONLY, 0)) < 0) goto Pdep; }

    T3T4T5:
    read(fpr, &petri, sizeof(petri));
    T5:
    if(petri.pvalid == 0)
        {r= 5; goto J3_4_5;}
    T4:
    if(petri.plibre == 0 AND petri.pvalid > 0)
        {r= 4; goto J3_4_5;}
    T3:
    if(petri.plibre > 0 AND petri.pvalid > 0) {
```

```

        for(i= petri.chaise, nuph= 0; (i&1) == 0; i= i>>1)
            {++nuph;}
        petri.chaise &= ~(1<<nuph);
        --petri.plibre; ++petri.poccm1; r= 3; goto J3-4-5; }
J3-4-5:
    write(fpw, &petri, sizeof(petri)); /* libere le RdP */
    if(r == 3) goto Prepas;
    if(r == 4) {printf("table complete\n"); goto Pfin;}
    if(r == 5) {close(fpr); close(fpw); goto Pdep;}

Prepas:
    printf("numero de chaise=%d\n", nuph);
    for(i= 0; i < MPHLO; ++i) {
        buf[4]= '0'+i;
        frep= open(buf, O_NDELAY|O_RDONLY, 0);
        fdp[i]= open(buf, O_WRONLY, 0); close(frep); }
    buf[4]= '0'+nuph; frep= open(buf, O_RDONLY, 0);
    repas();

T6T7:
    read(fpr, &petri, sizeof(petri));
    petri.chaise |= 1<<nuph;
T7:
    if(petri.poccm1 == 0 AND petri.pvalid > 0) {
        --petri.pvalid; unlink("fifo"); /* ++Pinit */
        for(i= 0; i < MPHLO; ++i)
            buf[4]= '0'+i; unlink(buf);}
        goto J6-7; }

T6:
    if(petri.poccm1 > 0)
        --petri.poccm1; ++petri.plibre; goto J6-7;}
J6-7:
    write(fpw, &petri, sizeof(petri)); /* libere le RdP */
    goto Pfin;
Pfin:
    return(nuph);
}

```

6.2 Preuve de programme

Que doit-on prouver ?

- problèmes de calcul séquentiel
- la programmation est-elle conforme au modèle ?
- le modèle est-il correct ?

Assertions

- Sur le modèle

– quand on est premier, on met la table :

a1- t_1 validée \Rightarrow $Prepas = 0$, $Pvalid = 0$, $Plibre = 4$, $Poccm1 = 0$

– quand on est le dernier à partir, on débarrasse

- a2-** t_7 validée $\Rightarrow P_{repas} = 1$
- la table ne déborde pas
- a3-** $P_{repas} \leq 5$
- tout le monde quittera la table
- a4-** le réseau converge vers $P_{fin} = X$, $P_{init} = 1$, $P_{valid} = 0$, $P_{libre} = 4$

• Sur la programmation

- certaines places sont des positions dans le code et les transitions en aval des voies dans le programme

a5- $P_{dep} \geq 0 \Rightarrow t_1 \text{ ou } t_2$

a6- $P_{phi} \geq 0 \Rightarrow t_3 \text{ ou } t_4 \text{ ou } t_5$

a7- $P_{repas} \geq 0 \Rightarrow t_6 \text{ ou } t_7$

			n_1	n_2	n_3	n_4	n_5	n_6	n_7
			t_1	t_2	t_3	t_4	t_5	t_6	t_7
Matrice d'incidence	q_1	P_{dep}	-1	-1	0	0	1	0	0
	q_2	P_{init}	-1	0	0	0	0	0	1
	q_3	P_{libre}	0	0	-1	0	0	-1	0
	q_4	P_{valid}	1	0	0	0	0	0	-1
	q_5	P_{phi}	0	1	-1	-1	-1	0	0
	q_6	P_{repas}	1	0	1	0	0	-1	-1
	q_7	P_{occm1}	0	0	1	0	0	-1	0
	q_8	P_{fin}	0	0	0	1	0	1	1

Invariants

• Séquences répétitives

$$WS = 0 \Leftrightarrow \begin{cases} -n_1 - n_2 + n_5 = 0 \\ \vdots \\ n_4 + n_6 + n_7 = 0 \end{cases}$$

• Invariants de marquage

$$QW = 0 \Leftrightarrow \begin{cases} -q_1 - q_2 + q_4 + q_6 = 0 \\ \vdots \\ q_2 - q_4 - q_6 + q_8 = 0 \end{cases}$$

Résolution

Theorem 32. • *Séquences répétitives*

$$\mathbf{r1} \quad n_1 = 0, n_3 = 0, n_4 = 0, n_5 = n_2, n_6 = 0, n_7 = 0$$

• *Invariants de marquage*

$$q_5 = q_1, q_6 = q_1 + q_2 - q_4, q_7 = -q_2 + q_3 + q_4, q_8 = q_1 \Rightarrow$$

$$\mathbf{i1} \quad P_{dep} + P_{phi} + P_{repas} + P_{fin} = X$$

$$\mathbf{i2} \quad P_{init} + P_{repas} - P_{occm1} = 1$$

$$\mathbf{i3} \quad P_{libre} + P_{occm1} = 4$$

$$\mathbf{i4} \quad P_{valid} - P_{repas} + P_{occm1} = 0$$

Autres relations

$$\mathbf{i5} \quad P_{init} + P_{valid} = 1 \Leftrightarrow ((i1) + (i4)) \text{ i.e' } P_{init} = 1 \text{ ou } P_{valid} = 1$$

$$\mathbf{r2} \quad P_{init} + n_1 - n_7 = 1 \quad (\text{matrice d'incidence})$$

$$\mathbf{p1} \quad P_{repas} > 0 \Rightarrow P_{valid} = 1$$

- vrai pour l'état initial
- conservé par t_1, t_2, t_3, t_4, t_5 et t_6
- t_7 validée $\Rightarrow P_{occm1} = 0$ ($i4$) $\Rightarrow P_{repas} = P_{valid}$ donc $P_{repas} = 1$ après tirage de t_7 , $P_{repas} = 0$ donc la relation est vérifiée

$$\mathbf{p2} \quad P_{repas} > 0 \Rightarrow t_6 \text{ ou } t_7 \text{ validée } P_{occm1} = 0 \text{ et } (p1) \Rightarrow t_7 \text{ validée}$$

Démonstrations

Proof. **Démonstration de (a2)** t_7 validée $\Rightarrow P_{repas} > 0$ et $P_{occm1} = 0$
 $P_{occm1} = 0 \wedge (i4) \Rightarrow P_{repas} = P_{valid} \quad P_{repas} = P_{valid} \wedge (i5) \Rightarrow P_{repas} \leq 1$

$$P_{repas} = 1$$

□

Proof. **Démonstration de (a3)**

- ($i3$) $\Rightarrow P_{occm1} \leq 4$ et ($i5$) $\Rightarrow P_{valid} \leq 1$
- avec ($i4$) $\Rightarrow P_{repas} \leq 5$ QED

□

Démonstration de (a1)

- t_1 validée $\Leftrightarrow P_{dep} > 0$ et $P_{init} > 0$
 - (i5) $\Rightarrow P_{init} = 1$
 - (r2) $\Rightarrow n_7 = n_1$
 - si $n_1 = 0$ c'est l'état initial
 - si $n_1 > 0$ alors t_7 a été tirée
 - * à la dernière validation de t_7 on avait (a2): $P_{repas} = 1, P_{occm1} = 0, P_{valid} = 1$
 - * (i3) et (i5) $\Rightarrow P_{libre} = 4$ et $P_{init} = 0$
 - * après le dernier tirage de t_7 : $P_{repas} = 0, P_{libre} = 4, P_{init} = 1, P_{occm1} = 0, P_{valid} = 0$ i.e. (a1)
 - dans ces conditions seules t_5 et t_1 peuvent être validées et t_5 ne modifie que P_{phi} et P_{dep}
- Si t_5 a été tiré, la condition (a1) est **restée validée**

Démonstration de (a4): le réseau se bloque

- (i1), (i2), (i3) \Rightarrow le marquage reste borné, donc sans séquence répétitive, le RdP se bloque.
- (r1) une séquence répétitive ne contient que t_2 et t_5 en nombre égal
 - t_2 validée $\Rightarrow P_{init} = 0$ avec (i5) on déduit qu'après le tirage de t_2 , t_5 n'est plus validée
 - t_5 validée $\Rightarrow P_{valid} = 0$ avec (i5) on déduit qu'après le tirage de t_5 , t_2 n'est plus validée
- Il n'existe pas de marquage validant une séquence répétitive
 - état bloqué = $\{P_{fin} = X, P_{init} = 1, P_{libre} = 4\}$ t_1 et t_2 invalidées $\Rightarrow P_{dep} = 0$ t_3, t_4 et t_5 invalidées $\Rightarrow P_{phi} = 0$ t_6, t_7 invalidées $\Rightarrow P_{repas} = 0$ à cause de (p2)
 - avec (i4) on obtient $P_{fin} = X$

$$P_{dep} = 0, P_{phi} = 0, P_{repas} = 0 \text{ et } P_{fin} = X$$

- t_1, t_2, t_3 et t_5 font croître l'une de ces places donc la dernière transition tirée est soit t_4 , soit t_6 soit t_7 .
- t_4 validée $\Leftrightarrow P_{libre} = 0$ et $P_{valid} = 1$ et $P_{phi} > 0$ avec $(i3) \Rightarrow P_{occm1} = 4$ et avec $(i4) \Rightarrow P_{repas} = 5$ avec $(p2)$ on sait que t_6 ou t_7 sont validées donc t_4 **n'est pas la dernière**.
- t_6 ou t_7 dernier tirage
 - au dernier tirage $P_{repas} = 1$ donc avec $(i4)$ et $(p1)$ on a $P_{occm1} = 0$ et c'est t_7 qui est **validée**.
 - après tirage de t_7 : $P_{valid} = 0$ ($i5$) , $P_{occm1} = 0$ donc $P_{fin} = X$, $P_{init} = 1$, $P_{libre} = 4$

Démonstration de (a5), (a6) et (a7)

- Ces assertions se déduisent des conditions de validation des transitions.
Par exemple pour a6 :
 - $P_{phi} > 0 \Rightarrow t_3$ ou t_4 ou t_5 validées. Sachant $P_{phi} > 0$, on peut écrire la table de vérité

P_{valid}	P_{libre}	t_3	t_4	t_5
0	0	<i>nv</i>	<i>nv</i>	<i>v</i>
0	> 0	<i>nv</i>	<i>nv</i>	<i>v</i>
> 0	0	<i>nv</i>	<i>v</i>	<i>nv</i>
> 0	> 0	<i>v</i>	<i>nv</i>	<i>nv</i>

et c'est un ou exclusif

v=Vrai, nv=Non Vrai

Remarques

- Comme $P_{init} = 0$ ou 1 il peut être représenté par l'inexistence du fichier **fifop**.

- Après le tirage de t_1 , les places P_{libre} , P_{valid} , P_{occm1} ont toujours la même valeur. Il est donc valable de remplacer l'incrément des valeurs par leur forçage
- L'accès atomique au RdP est réalisé par les fonctions **mknod**, **read**, **write**, **unlink**.
- La validité du calcul séquentiel peut être appréciée par les méthodes habituelles

7 Conclusion

7.1 Utilisation d'équivalences

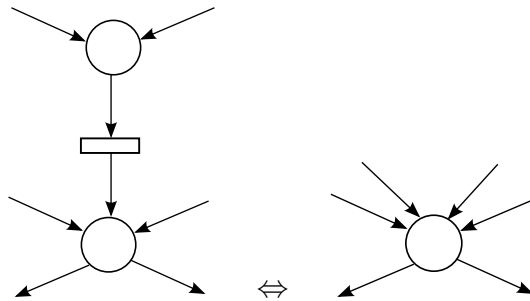
Propriétés nécessaires des équivalences

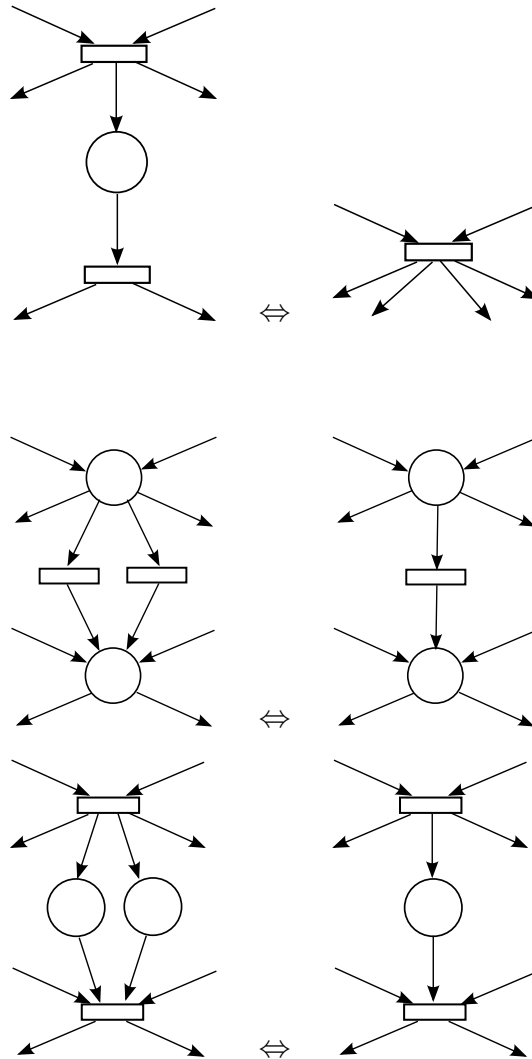
Toute équivalence doit préserver :

- Vivacité du RdP
- Sureté
- Dénombrabilité des états

De par l'existence d'équivalences, il n'existe jamais un seul unique modèle valide pour un problème. D'autre part il est souvent avantageux de résoudre de gros RdP par partie lorsque c'est possible (démontrer certaines propriétés locales sur une partie du RdP complet puis remplacer la partie vérifiée par un modèle plus simple).

Exemples





8 Modelisation Temps-Réel : un exemple concrêt (simplifié)

Parallélisme, réseaux de Petri et temps-réel

Pourquoi étudier les réseaux de Petri pour des problèmes de temps-réel ?

- Les systèmes temps-réels sont naturellement des systèmes parallèles:
 - Plusieurs échelles de temps

- Partage de ressources complexe (temps, mémoire, capteurs, actuateurs, ...)
- Parfois événements sporadiques, *etc.*
- 2 problèmes difficiles en temps-réel:
 1. Parallélisme (possibles fautes dans la gestion du parallélisme)
 2. Ordonancement
- S'il y a une faute de parallélisme, l'ordonancement ne sert à rien

8.1 L'hélicoptère martien Ingenuity

Un modèle simplifié de l'hélicoptère martien Ingenuity

En entrée (capteurs):

1. Camera (25Hz)

- 1 image produite toute les $\frac{1}{25}$ s
- une *main frame* de façon sporadique (les autres sont *inter-frame*)
- transition T_{cam}

2. Centrale inertielle (accélération 6 axes)

- transition T_{inert} fréquence 125Hz

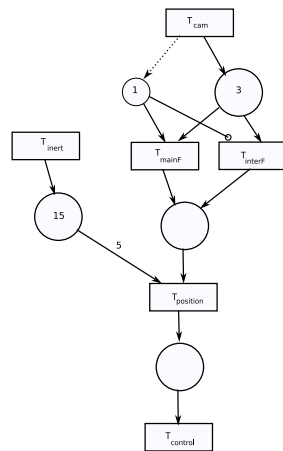
En sortie (actuateur):

1. Système de controle moteurs (1 axe, 2 rotors)

- transition $T_{control}$

Calculs/traitements intermédiaires:

- transition T_{mainF} : position et vitesse sur main frame
- transition T_{interF} : vitesse (2 axes 2D)
- transition $T_{position}$: position, vitesse et attitude



Analyse du modèle: algèbre linéaire



Matrice d'incidence (modèle simplifié):

$$W = \begin{pmatrix} 1 & 0 & -1 & 0 & 0 \\ 0 & 1 & 0 & -5 & 0 \\ 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 1 & -1 \end{pmatrix} \quad (1)$$

N.B.: pas de composante conservative

Séquence répétitive: $W.S = 0$

On trouve $S_r = {}^t (1 \ 5 \ 1 \ 1 \ 1)$

- On retrouve l'état initial après 1 exécution de T_{cam} , 5 exécutions de T_{inert} , 1 exécution de T_{frameP} et 1 exécution de T_{pos} et T_{ctrl}
- si la fréquence de la caméra est 25Hz, la séquence répétitive est consistante avec une fréquence de $5 \times 25 = 125Hz$
- *C'est toujours le cas pour un système périodique*: la séquence répétitive fournit une base pour l'ordonnancement (\neq ordonnancement)

Grphe de marquage (réseau non simplifié)

à faire en cours

Programmation conforme, commentaires, conclusion

à faire en cours

8.2 Utilisation scientifique et industrielle des RdP

Utilisation pratique

- Évaluations de performance (en particulier sur les bases de RdP non autonomes)
- Conception, Modélisation et vérification de protocoles réseaux
- Systèmes de fichiers et bases de données
- Systèmes de contrôle de fabrication ou industriels

- Systèmes à événements discrets
- Protocoles pour la mémoire des multiprocesseurs
- Systèmes tolérants aux fautes
- Calculs en flots de données
- Langages, compilateurs et systèmes d'exploitation
- *etc.*

Les RdP sont des outils de modélisation importants des processus asynchrones et parallèles

Quelques références intéressantes

- C. A. Petri, “Kommunikation mit Automaten.” Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr.3, 1962. Traduction anglaise: “Communication with Automata” New York: Griffiss Air Force base. Tech.rep. RADCTR-65-377, Vol. 1, 1966
- C.A. Petri, “Fundamentals of a theory of asynchronous information flow”, *in* Proc. IFIP Congress 62, pp. 386–390, 1963
- J. L. Peterson, “Petri Net Theory and the Modeling of Systems”, Englewood Cliffs, NJ: Prentice-Hall, 1981
- T. Murata, “Petri Nets: Properties, Analysis and Applications”, *Proceeding of the IEEE*, Vol. 77, N. 4, 1989
- G. Vidal-Naquet, “Réseaux de Petri et Systèmes parallèles”, Armand-Colin Editeur, ISBN 978-2-200-21197-4, 1992