# Modèle Linda

communication par espace commun de travail



## Introduction

Motivation : le couplage des processus participant à la synchronisation est fort dans toutes les primitives étudiées jusqu' à présent : les processus doivent exister pour se synchroniser.

Modèle Linda

- **espace de travail** (*tuple space*) : structure globale de données, où les processus déposent, lisent ou effacent les données.
- **notes** (*tuple*) : séquence typée de données ; le premier élément de la séquence est un littéral qui indique le contenu de la note.

## Modèle Linda (cont.)

Opérations atomiques sur l'espace de travail :

- **postnote(e1,...,en)** : création d'une note à partir des valeurs et son dépôt dans l'espace de travail. Ceci peut débloquer un processus (arbitraire) en attente d'un note de ce type.
- **removenote(x1,...,xn)** : efface une note qui correspond aux types de variables formels de l'opération ; si aucune note ne correspond, le processus est bloqué. Si plusieurs notes correspondent, l'une est choisie arbitrairement.
- **readnote(x1,...,xn)** : comme ci-dessus mais sans effacement de la note.



## Exemple :

```
int m, n; boolean b;
    postnote(‘a‘, 10, 20);
    postnote(‘a‘, 30);
    postnote(‘a‘, false, 40);
    postnote(‘a‘, true, 50);
    removenote(‘b‘, m, n);
    removenote(‘a‘, m, n);
    readnote(‘a‘, m);
    removenote(‘a‘, n);
    removenote(‘a‘, m, n);
    removenote(‘a‘, b, n);
    removenote(‘a‘, b, m);
    postnote(‘b‘, 60, 70);
```

## Expressivité du modèle Linda

### Modélisation des sémaphores

- Initialisation du sémaphore :

```
do :: /* k times */ -> postnote('s') od
```

- Attente sur le sémaphore :

```
removenote('s');
```

- Signalisation du sémaphore :

```
postnote('s');
```

**Modélisation des moniteurs :** le langage qui utilise le modèle Linda doit assurer l'encapsulation des données. Par exemple : JAVASPACES.



**Exemple 1: Client-serveur.** Un serveur propose un service dont le type est s à un ou plusieurs clients. Le client envoie une requête de service s (note R) avec des paramètres corrects et attend la réponse (note S).

```
active proctype Client (int me) {
  serviceType service;
  dataType param, result;
  service = /* service demande */;
  param = /* intialise les parametres */;
  postnote('S',me,service,param); /* non-Promela */
  removenote('R',me,result) /* non_Promela */
}
active proctype Server (serviceType s) {
  int client;
  dataType p, r;
  removenote('S',client,s=,p); /* parametres formel s */
  r = do(s,p);
  postnote('R',client,r) /* non-Promela */
}
```

## Exemple 2 : canal FIFO non borné

```
active proctype Producer () {
  int count = 0;
  dataType d;
  do :: true -> d = /* produce */;
                postnote('B',count,d);
                count = count + 1
  od
}
active proctype Consumer () {
  int count = 0;
  dataType d;
  do :: true -> removenote('B',count=,d); /* parametre formel */;
                consume(d);
                count = count + 1
  od
}
```

Questions :

- Quelles sont les limitations de cette implementation ?
- Comment obtenir un canal FIFO borné en Linda ?



## Paradigme maître-travailleur

Motivation : utilisation des algorithmes parallèles avec des canaux synchrones impose la présence et la même vitesse de travail pour tous les processus.

Le paradigme maître-travailleur assure la flexibilité dans la réalisation d'une tâche. Le maître détermine quelles sont les sub-tâches à réaliser et les travailleurs, quand ils sont disponibles, effectuent ces sub-tâches.

La granularité du paradigme est déterminée par le nombre de sub-tâches envoyés par le maître.

Exemples :

- Multiplication de deux matrices : sub-tâche = calcul d'un élément de la matrice produit.
- Tri de nombres : sub-tâche = tri d'une partie du tableau.

## Multiplication de matrices (1)

Les matrices opérandes du produit sont disponibles par lignes et colonnes.

Avec granularité = 1 élément du produit par travailleur.

```
active proctype Master() { /* n = dimension des matrices */
  int i, j, result, r, c;
  for i = 0 to n-1 do
    for j = 0 to n-1 do
      postnote('T',i,j)
  od od
  for i = 0 to n^2-1 do
      removenote('R',r,c,result);
      print r,c,result
  od od
}
active proctype Worker() {
  int r,c,result;
  int[n] vec1, vec2;
  do :: true -> removenote('T',r,c);
    readnote('A',r=,vec1);
    readnote('B',c=,vec2);
    result = vec1 * vec2; /* multiplication de vecteurs */
    postnote ('R',r,c,result)
  od
}
```



## Multiplication de matrices (2)

Avec granularité : 1 group de k lignes par travailleur.

Exercice !

## Simulation du modèle Linda avec Promela

- espace = canal
- note = message
- opérations sur l'espace :
  - poster une note : envoi sur le canal

    ```
    ch ! 'T',i,j,0,0
    ```

  - effacer une note d'un certain type : lecture avec sélection :

    ```
    ch ?? 'T',r,c,_,_
    ```

  - lire une note d'un certain type :

    ```
    ch ?? <'A',eval(r),vec11,vec12,vec13>
    ```

Donc limitations, mais facile à utiliser.