TD3 : systèmes de tâches et parallélisme maximal

Systèmes d'exploitation

sergiu.ivanov@univ-evry.fr

1 Contexte

Ce TD porte sur la maximisation du parallélisme dans les systèmes de tâches abstraites. Nous discuterons de la composition séquentielle et parallèle, de la traduction des programme parallèles en graphe de précédence et inversement, des traces d'exécution, du déterminisme des systèmes de tâches et, enfin, nous nous exercerons à l'algorithme de construction de systèmes de parallélisme maximal.

2 Programmes parallèles et graphes de précédence

La plupart des librairies de programmation parallèle fournissent des primitives réalisant la *composition parallèle* : la mise en parallèle de plusieurs tâches. Le paradigme de programmation impérative fournit à son tour la *composition séquentielle* : l'exécution de plusieurs tâches l'une à la suite de l'autre.

La composition parallèle est souvent écrite en utilisant le symbole $\|$, alors que la composition séquentielle est notée en juxtaposant les tâches. Par exemple, $(T_1 \parallel T_2)\, T_3$ exprime la composition parallèle des tâches T_1 et T_2, et ensuite la composition séquentielle du résultat avec la tâche T_3.

Cependant, lorsqu'on souhaite décrire dans un pseudocode le contenu de tâches composées, il est plus pratique d'utiliser des constructions `parbegin` / `parend` pour la composition parallèle et `début` / `fin` pour la composition séquentielle. L'exemple précédent s'exprimerait avec ces constructions comme ceci :

```
début
  parbegin T1; T2 parend;
  T3
fin
```

2.1 Conversion entre les deux formats d'écriture

Écrivez les expressions suivantes avec les constructions `parbegin` / `parend` et `début` / `fin` :

1. $T_1\,((T_2\,T_3) \parallel T_4)\,T_5\,T_6$
2. $T_1 \parallel (T_2\,(T_3 \parallel T_4)\,T_5) \parallel T_6$
3. $(T_1\,T_2) \parallel (T_3\,T_4\,((T_5 \parallel T_6)\,T_7))$

2.2 Programme → graphes de précédence

Construisez les graphes de précédence correspondant aux programmes et aux expressions de l'exercice précédent. Ensuite, construisez le graphe de précédence pour le programme suivant :

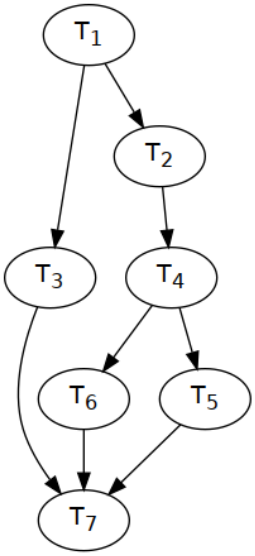
```
début
  parbegin lire(a); lire(b) parend;
  parbegin
    c := a × a;
    début
      d := b × b;
      parbegin f := a + b; g := a × b parend
    fin
  parend;
```

```
  e := c + d - (g + f)
fin
```

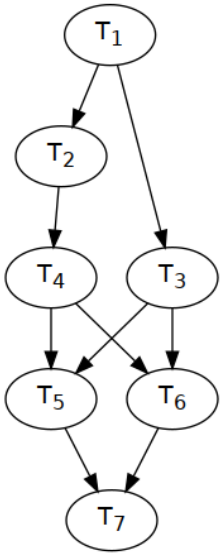
2.3 Graphes de précédence → programme

Exprimez les graphes de précédence suivants sous forme de programmes utilisant les constructions `parbegin`, `parend`, `début` et `fin`, puis sous forme d'expressions utilisant l'opérateur ‖ et la juxtaposition des tâches :

Graphe 1 :



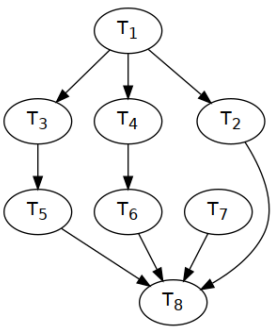
Graphe 2 :



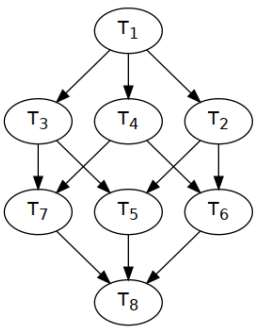
2.4 Limites des compositions parallèle et séquentielle

Tous les graphes de précédence ne peuvent pas être présentés avec des compositions parallèle et séquentielle. Traduisez les graphes de précédence suivants en programmes en utilisant les constructions `parbegin`, `parend`, `début` et `fin`, puis en expressions en utilisant l'opérateur ‖ et la juxtaposition des tâches. Argumentez rigoureusement les cas d'impossibilité de traduction éventuels.

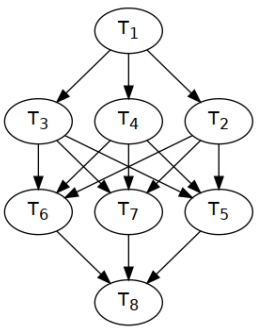
Graphe 1



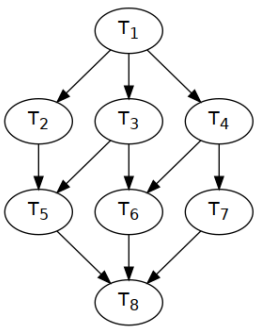
Graphe 2



Graphe 3



Graphe 4



Les graphes de précédence qui ne sont pas exprimables en termes d'opérations de composition parallèle et séquentielle restent pourtant des graphes valables que l'on peut rencontrer dans des situations réelles. Proposez une idée de solution qui pourrait être utilisée afin d'assurer les contraintes de précédence imposées par un graphe arbitraire.

3 Comportements des systèmes de tâches

Pour décrire le comportement d'un système de tâches, on associe des *interprétations* aux tâches : des fonctions décrivant les valeurs des sorties pour des valeurs d'entrée données. Ensuite, on trace les valeurs que le système peut écrire dans ses cases mémoire (variables) lors des exécutions permises par le graphe de précédence.

Prenons, par exemple une tâche T_1 avec le domaine de lecture $L_1 = \{M_1, M_2\}$, le domaine d'écriture $E_1 = \{M_1, M_3\}$ et l'interprétation $F_1(M_1, M_2) = (M_1 + M_2, M_1 \times M_2)$. Cette tâche lit les valeurs des cases mémoire M_1 et M_2, et met la somme de ces valeurs dans M_1 et le produit dans M_3.

3.1 Suites de valeurs écrites dans des cases mémoire

Considérons le système de tâches $S = (\{T_1, T_2\}, \emptyset)$, avec $L_1 = L_2 = E_1 = E_2 = \{M_1, M_2\}$ et les interprétations $F_1(M_1, M_2) = (M_1 + M_2, M_2)$ et $F_2(M_1, M_2) = (M_1, M_1 + M_2)$. Déterminez les suites de valeurs que ces tâches écriront dans les cases mémoire M_1 et M_2, pour tous les comportements possibles w du système S ($w \in L(S)$). Pour rappel, un comportement du système S est une suite de débuts et de fins de tâches d_1, d_2, f_1, f_2 qui satisfait les contraintes de précédence du système. Pour chaque comportement, construisez les suites des valeurs distinctes $V(M_1, w)$ et $V(M_2, w)$, $w \in L(S)$, et déduisez-en si le système S est déterminé.

Refaites la même analyse pour le système S' avec les changements suivants par rapport à S : $E_1 = \{M_1\}$, $E_2 = \{M_2\}$, $F_1(M_1, M_2) = F_2(M_1, M_2) = (M_1 + M_2)$.

3.2 Variabilité des comportements d'un système non déterminé

Pour le programme suivant, déterminer précisément les bornes inférieure et supérieure sur la valeur finale de la variable partagée `compte` calculée par ce programme parallèle. Pour chaque borne, donner un exemple de comportement pour lequel elle est atteinte.

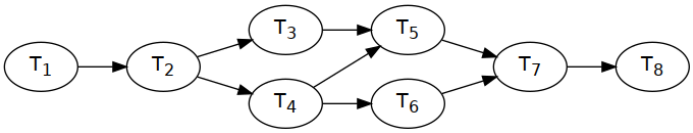
On supposera que les processus s'exécutent avec des vitesses relatives quelconques et qu'une valeur peut être incrémentée ou décrémentée seulement après qu'elle ait été chargée dans un registre par une instruction machine (c'est-à-dire que la valeur de `compte` ne peut pas être incrémentée ou décrémentée directement dans la mémoire).

```
const n = 50;
var compte : entier;

procedure somme
  var count : entier;
  début
    pour count := 1 à n faire
      compte := compte + 1
    fpour;
    pour count := 1 à n faire
      compte := compte - 1
    fpour
  fin

début (* programme principal *)
  compte := 0;
  parbegin
    somme; somme; somme
  parend;
  écrire(compte)
fin
```

4 Parallélisme maximal

4.1 Construction d'un système de tâches de parallélisme maximal

Considérons le système de tâches S avec le graphe de précédence suivant :



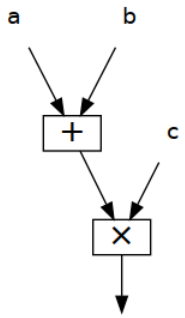
Ce système utilise 5 cases mémoire (variables). Les domaines de lecture et d'écriture des tâches sont donnés dans le tableau suivant :

	L_i	E_i
T_1	M_1, M_2	M_3
T_2	M_1	M_4
T_3	M_3, M_4	M_1
T_4	M_3, M_4	M_5
T_5	M_4	M_2
T_6	M_5	M_5
T_7	M_1, M_2, M_4	M_4
T_8	M_1, M_3	M_5

Vérifiez que ce système est déterminé et construisez un système de parallélisme maximal équivalent.

4.2 Parallélisation d'un programme

Un arbre d'évaluation d'une expression arithmétique est un graphe explicitant l'ordre d'exécution des opérations. Par exemple, l'arbre d'évaluation de l'expression $(a + b) \times c$ est le suivant :



Construisez l'arbre d'évaluation de l'expression suivante :

$$2 \times ((a + b)/(c - d) + e \times f) + (a + b) \times (c - d).$$

Faites en sorte que chaque sous-expression apparaisse dans l'arbre une seule fois. Associez une variable à chaque nœud de l'arbre et écrivez un programme *séquentiel* qui évalue l'expression opération par opération. Construisez ensuite un système de parallélisme maximal équivalent à ce programme séquentiel. Est-il possible de représenter ce système avec les opérations de composition parallèle et séquentielle uniquement ? Si oui, écrivez le programme parallèle utilisant les constructions `début` / `fin` et `parbegin` / `parend`. Sinon, écrivez un programme parallèle utilisant ces constructions et exhibant le plus de parallélisme possible.