Fiche 3: Modèles de calcul polynomialement équivalents - Classe P

Modèle RAM

Le modèle RAM est le modèle de calcul qui se rapproche le plus des ordinateurs actuels et de leur progammation en langage assembleur.

Les machines RAM que nous considérons (il en existe des variantes) sont constituées :

- d'une suite (potentiellement infinie) de registres numérotés à partir de 0 (r_0, appelé accumulateur, r_1, ... r_n,...), chacun pouvant contenir un entier (naturel, donc positif ou nul) arbirairement grand, et tous initialisés à 0,
- une bande d'entrée, dont chaque case peut contenir un entier arbirairement grand, sur laquelle la machine lit ses données en mettant dans l'accumulateur le contenu de la case pointée par la tête de lecture (la tête de lecture est alors déplacée d'une case vers la droite),
- une bande de sortie, dont toutes les cases, initialement vides, peuvent contenir un entier naturel arbirairement grand, sur laquelle la machine écrit, en mettant dans la case pointée par la tête d'écriture le contenu de l'accumulateur (la tête d'écriture est alors déplacée d'une case vers la droite),
- un programme constitué d'une suite finie d'instructions numérotées à partir de 0.

On note $c(i)$ le contenu du registre numéro i.

Les instructions RAM sont de 4 types : manipulations de registres, opérations arithmétiques, instructions d'entrée/sortie, et instructions de saut et d'arrêt.

- **Manipulation de registres : LOAD, STORE**
  - **LOAD =i**
    $c(0) := i$ (l'entier i est mis dans l'accumulateur, on parle d'**adressage absolu**).
  - **LOAD i**
    $c(0) := c(i)$ (l'entier contenu dans le registre numéro i est mis dans l'accumulateur, on parle d'adressage direct).
  - **LOAD *i**
    $c(0) := c(c(i))$ (l'entier contenu dans le registre dont le numéro est contenu dans le registre numéro i est mis dans l'accumulateur, on parle d'adressage indirect).
  - **STORE i**
    $c(i) := c(0)$ (l'entier contenu dans l'accummulateur est chargé dans le registre numéro i).
  - **STORE *i**
    $c(c(i)) := c(0)$ (l'entier contenu dans l'accummulateur est chargé dans le registre dont le numéro est contenu dans le registre numéro i).
- **Opérations arithmétiques : ADD, SUB**
  - **ADD =i** ou **ADD i** ou **ADD *i**
    $c(0) := c(0) + i$ ou $c(0) := c(0) + c(i)$ ou $c(0) := c(0) + c(c(i))$
  - **SUB =i** ou **SUB i** ou **SUB *i** (**"différence propre"**).
    $c(0) := c(0) - i$ ou $c(0) := c(0) - c(i)$ ou $c(0) := c(0) - c(c(i))$
    et 0 si $c(0) < i$ (ou $c(0) < c(i)$ ou $c(0) < c(c(i))$)
- **Instructions d'entrée/sortie : READ, WRITE**
  - **READ i** ou **READ *i**
    $c(i)$:= contenu de la case courante de la bande d'entrée
    ou $c(c(i))$:= contenu de la case courante de la bande d'entrée
    + déplacement vers la droite de la tête de lecture de la bande d'entrée
  - **WRITE =i** ou **WRITE i** ou **WRITE *i**
    i ou $c(i)$ ou $c(c(i))$ est écrit dans la case courante de la bande de sortie
    + déplacement vers la droite de la tête d'écriture de la bande de sortie.
- **Instruction de rupture/saut : JUMP, JGTZ, JZERO, HALT**
  - **JUMP n**
    met le compteur des instructions (parfois appelé compteur ordinal) à **n**.
  - **JGTZ n**
    met le compteur des instructions à **n** si $c(0) > 0$, l'incrémente de 1 sinon.
  - **JZERO n**
    met le compteur des instructions à **n** si $c(0) = 0$, l'incrémente de 1 sinon.
  - **HALT**
    arrêt du programme.

1. **Question :** En complétant le programme suivant, écrire un programme RAM qui, à partir d'un entier n fourni en entrée , renvoie comme résultat n^2.

0 :	*READ*	1	(c(1) := n)
1 :	*LOAD*	1	(c(0) := c(1), et donc c(0) vaut n)
2 :	*JGTZ*	5	(si n > 0 aller à l'instruction numéro 5)
3 :	*WRITE*	= 0	(sinon, puisque n = 0, écrire 0 sur la bande de sortie)
4 :	*JUMP*	*end*	(sauter à l'instruction dont l'étiquette est "end")
5 :	*STORE*	2	(c(2) := c(1), et donc c(2) vaut n)
6 :	...		
	...		
endwhile :	*WRITE*	2	(écrire sur la bande de sortie le résultat stocké dans le registre numéro 2)
end :	*HALT*		

2. La *complexité d'un programme RAM* est l'ordre de grandeur de la fonction qui estime le nombre maximum d'instructions à exécuter par le programme sur une entrée de taille n, *en supposant que les entiers sont codés en binaire, et donc que la taille d'un entier (ou d'une séquence d'entiers) est le nombre de bits (c'est-à-dire de 0 ou 1) de son codage en binaire.*
3. **Question :** Quelle est la complexité du programme RAM de la question précédente ?

Simulation d'instructions RAM par Machine de Turing

Un programme RAM qui effectue $f(n)$ instructions sur une entrée de taille n peut être simulé par une Machine de Turing à 7 bandes (dont 3 servent à simuler les opérations arithmétiques, comme l'addition de 2 entiers codés en binaire), en $O(f(n)^3)$.

— Le principe de la simulation est le suivant :
  — La bande 1 de la MT simule la bande d'entrée de la machine RAM, et contient donc le codage en binaire de l'entrée du programme RAM.
  — La bande 2 de la MT code le contenu des différents registres actifs de la machine RAM comme une séquence de mots $b(i) : b(c(i))$ séparés par des B, où $b(i)$ est le codage binaire de l'entier i et donc $b(c(i))$ est le codage binaire de l'entier contenu dans le registre numéro i.
  La partie codante de cette bande 2 est délimitée par un symbole d de début et un symbole $*$ de fin.
  La simulation de la mise à jour du contenu d'un registre de la machine RAM se fera en cherchant sur la bande 2 de la MT le mot $b(i) : \gamma B$ correspondant, en l'effaçant (en le remplaçant par des B) et en ajoutant à la fin le nouveau mot $b(i) : b(c(i))$, déplaçant d'autant le symbole $*$.
  — La bande 3 de la MT contient le codage binaire de la valeur courante du compteur d'instructions.
  — La bande 4 de la MT contient le codage binaire de l'adresse (c'est-à-dire du numéro) du registre de travail.
  Pour savoir quel est le contenu du registre de travail (sur lequel on doit appliquer des opérations arithmétiques ou de test à zéro ou qu'on doit écrire sur la bande sortie selon l'instruction courante), on cherche sur la bande 2 de la MT un mot $b(i) : b(c(i))$ et on extrait $b(c(i))$.
  — Les 3 autres bandes servent à faire les opérations de sorte, et on fait en sorte que la bande 7 contienne le résultat quand l'instruction HALT est rencontrée.

— **Question :** Indiquer les différentes étapes de la simulation par cette MT multi-bande de l'instruction RAM : ADD *20, c'est-à-dire : c(0) := c(0)+ c(c(20)).

— **Propriété 1 :** Après l'exécution de la $t-$ième instruction du programme RAM, le contenu de chaque registre a une taille d'au plus $t + l(I) + l(M)$ où $l(I)$ est la taille du codage binaire de l'entier I fourni en entrée, et $l(M)$ est la taille du codage binaire du plus grand entier M apparaissent comme opérande d'une instruction du programme RAM.

La preuve se fait par récurrence sur t, en passant en revue toutes les instructions qui modifient le contenu des registres : LOAD, STORE, READ, ADD ou SUB.

— **Propriété 2 :** Le nombre d'instructions de la MT nécessaires pour simuler une instruction du programme RAM est en $O(f(n)^2)$, où $n = l(I)$.

La preuve repose sur l'observation que les instructions RAM les plus coûteuses à simuler sont celles qui nécessitent d'accéder au contenu des registres, et sur les éléments suivants concernant la bande 2 (dans laquelle la MT doit aller chercher le contenu des registres) :

1. le nombre de mots $b(i) : b(c(i))$ sur la bande 2 est égal au nombre de registres actifs de la machine RAM : au plus $f(n)$ car on ne peut changer le contenu d'un registre que par une instruction et car au départ tous les registres sont inactifs,

2. La taille de chaque $b(i) : b(c(i))$ est en $O(f(n))$ par application de la propriété 1 avec $t = f(n)$ et $n = l(I)$.
3. La taille de la partie codante de la bande 2 est donc en $O(f(n)^2)$.
4. Chercher un mot de la forme $b(i) : b(c(i))$ sur la bande 2 et copier $b(c(i))$ sur une autre bande nécessite un nombre d'instructions de la MT en ordre de grandeur égal à la taille du mot total sur la bande 2, soit en $O(f(n)^2)$.
5. Les autres instructions RAM sur des entiers peuvent être simulées par la MT en temps linéaire par rapport à la taille du codage binaire de ces entiers, qui est en $O(f(n)$. Donc, leur simulation est en $O(f(n)$, et a fortiori en $O(f(n)^2$.

— On en déduit le théorème suivant qui est un résultat fondamental.

Théorème : *Le modèle RAM est polynomialement équivalent au modèle des Machines de Turing.*

Modèle pseudo-PASCAL

Le modèle pseudo-PASCAL est le modèle de calcul qui se rapproche le plus des langages de haut niveau pour décrire des algorithmes. Il suit les conventions habituelles pour décrire des algorithmes, tout en acceptant, quand il n'y a pas d'ambiguïté, certaines libertés sur l'écriture des variables ou des expressions.
Les instructions pseudo-PASCAL que nous considérons (il en existe des variantes) sont les suivantes (avec leur sémantique habituelle) :

— variable := expression
— **if** condition **then** statement (**else** statement)
— **while** condition **do** statement
— **repeat** statement **until** condition
— **for** variable := initial-value **until** final-value **do** statement
— **begin** statement ; statement ; ... ; statement **end**
— **read** variable
— **write** expression

1. **Question :** Écrire un programme pseudo-Pascal utilisant uniquement comme opérations l'addition et la soustraction, qui lit un entier, calcule son carré et écrit (ou affiche) le résultat.
2. La *complexité d'un programme pseudo-Pascal* est l'ordre de grandeur de la fonction qui estime le nombre maximum d'instructions *élémentaires* à exécuter par le programme sur une entrée de taille n, où les instructions élémentaires sont :
   — les opérations arithmétiques considérées comme élémentaires dans le modèle (l'addition et la soustraction, ou selon la variante du modèle l'addition, la soustraction et la multiplication),
   — l'affectation d'une valeur à une variable.
3. **Question :** Quelle est la complexité du programme pseudo-Pascal de la question précédente ?
4. **Question :** Montrer qu'une instruction élémentaire d'un programme pseudo-Pascal peut être simulée par un nombre maximum constant d'instructions RAM.
5. Il en découle que la complexité d'un programme pseudo-Pascal est la même en ordre de grandeur que la complexité du programme RAM résultant de sa compilation.

On en déduit le résultat central suivant :

Théorème : *Les modèles de calcul des Machines de Turing, des machines RAM et des programmes pseudo-Pascal, et toutes leurs variantes, sont polynomialement équivalents.*

La classe P des problèmes/fonctions polynomiaux

On peut maintenant définir cette classe indépendamment de tout modèle de calcul (dès lors qu'il est montré polynomialement équivalent à celui des Machines de Turing).

Définition : La classe P des problèmes/fonctions polynomiaux est l'ensemble des problèmes/fonctions qui peuvent être décidés/calculées par un algorithme de complexité polynomiale dans n'importe quel modèle de calcul.