UNIVERSITÉ GRENOBLE ALPES **M1-info**

Complexité algorithmique de problèmes **2016-17**

TD05 – Simulation entre modèles RAM, pseudo-Pascal, et MT — énoncé

Modèle RAM

Le modèle RAM (Random Access Machine) est un modèle de programmation plus proche des automates programmables que les MT. Dans ce modèle, le programme est séparé de la mémoire de travail. Le modèle qui se rapproche le plus des ordinateurs actuels (modèle de von Neumann) et de leur progammation en assembleur est le modèle RASP (Random Access Stored Program), non vu dans ce cours.

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.
- d'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.
- d'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.
- d'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 r_0, on parle d'**adressage absolu**).
  - **LOAD i**
    $c(0) := c(i)$ (l'entier contenu dans le registre r_i est mis dans r_0, 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 r_i est mis dans l'accumulateur r_0, on parle d'**adressage indirect**).
  - **STORE i**
    $c(i) := c(0)$ (l'entier contenu dans l'accumulateur 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**	(addition usuelle)
$c(0) := c(0) + i$	$c(0) := c(0) + c(i)$	$c(0) := c(0) + c(c(i))$	

  -

SUB =i	ou **SUB i**	ou **SUB *i**	(différence propre)
$c(0) := c(0) \dot{-} i$	$c(0) := c(0) \dot{-} c(i)$	$c(0) := c(0) \dot{-} c(c(i))$	
(0 si $c(0) < i$)	(0 si $c(0) < c(i)$)	(0 si $c(0) < c(c(i))$)	

- **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 (souvent 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.

Mesure de la "taille de l'entrée" dans une RAM : on suppose toujours que les entiers contenus dans les registres d'une RAM sont écrits en binaire, sans 0 inutile en tête.
Rappel : le codage binaire d'un entier i, noté $b(i)$, est le mot $b_k b_{k-1} ... b_0$ où les symboles b_i sont appelés des *bits* et valent 0 ou 1, avec $b_k = 1$ si $i \neq 0$, et $b_0 = 0$ ($k = 0$), si $i = 0$ [0 est noté par la chaîne "0"]. Il correspond à son écriture "sans 0 inutile en tête" en base 2 :

$$i = \Sigma_{j=0}^{k} \, b_j 2^j$$

La taille binaire de l'entier i, notée $l(i)$, est la taille du mot qui le code en binaire :

$$l(i) = |b(i)| = k + 1$$

Remarque : on pourrait définir un modèle RAM traitant les entiers relatifs. Dans ce cas, la soustraction serait l'opération usuelle (-) et pas la différence propre ($\dot{-}$), et il faudrait ajouter 1 à la longueur, pour coder un "bit de signe".

Compilation de programmes pseudo-Pascal en programmes RAM

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. Écrire un programme pseudo-Pascal utilisant uniquement comme opérations l'addition et la soustraction, qui lit deux entiers, calcule leur produit et écrit (ou affiche) le résultat. Le traduire en un programme RAM.
2. Ecrire un programme en pseudo-Pascal, puis le traduire en un programme RAM, qui lit une suite d'entiers positifs dont la fin est indiquée par la lecture d'un 0 et qui affiche le maximum des entiers de la suite.

3. Écrire un programme en pseudo-Pascal, puis le traduire en un programme RAM, qui lit une suite d'entiers dont la fin est indiquée par la lecture d'un 0 et qui affiche 1 si le nombre de 1 est égal au nombre de 2 dans la suite, 0 sinon. [Rappel : un registre contient entier *naturel* arbitraire. Donc *la solution qui suit est erronée, il faudrait la corriger en utilisant un autre registre pour coder le signe de la différence.*]
4. Reconnaître un palindrome. On suppose comme usuellement que les caractères sont codés sur [1..k] et que 0 marque la fin de la chaîne en entrée.
   – produire 1 si $w = \widetilde{w}$
   – produire 0 sinon
   Qu'arriverait-il si on ne disposait pas de l'adressage indirect ?

[On met une réponse ici, même dans l'énoncé, car il semble qu'il y a un conflit entre notre macro \Reponse et l'environnement minipage.]

RAM

```
    LOAD   =0
    STORE    2
tantq1:
    READ   1
    LOAD   1
    JZERO    ftql
    LOAD   2
    ADD    =1
    STORE    2
    LOAD   2
    ADD    =6
    STORE    5
    LOAD   1
    STORE  *5
    JUMP   tantq1
ftql:
    LOAD   2
    STORE    3
    JZERO    ecr1
    LOAD   =1
    STORE  4
tantq2:
    LOAD   2
    JZERO    ecr1
    ADD    =6
    STORE    5
    LOAD   4
    ADD    =6
    STORE    6
    LOAD   *5
    SUB    *6
    JZERO    cont2
    LOAD   3
    STORE    2
    WRITE    =0
    LOAD   2
tantq3:
    JZERO    fin
    ADD    =6
    WRITE    *0
    SUB    =7
    JUMP   tantq3
cont2:
    LOAD   4
    ADD    =1
    STORE    4
    LOAD   3
    SUB    =1
    STORE  3
    JUMP   tantq2
ecr1:
    WRITE    =1
fin:
    HALT
```

Pseudo-Pascal

Registres :

- r0 : accumulateur
- r1 : x
- r2 : k
- r3 : n
- r4 : l
- r5 : k+6
- r6 : l+6
- $r_{k+6} : a_k$

```
debut k := 0 ; lire x;
  tantque x != 0 faire
     k := k+1;
     r_{k+6} := x;
     lire x;
  refaire
  n := k;
  si n = 0 alors ecrire 1
  sinon
     l := 1
     tantque k > 0 faire
          si r_{k+6} = r_{l+6}
          alors l := l+1; k := k-1;
          sinon k := n;
              ecrire 0;
              tantque k > 0 faire
                  ecrire r_{k+6};
                  k := k-1;
              refaire
              sortir
          fsi
     refaire
     ecrire 1
  fsi
fin
```