Ensimag 1ère année Mai 2009

Théorie des langages 2

Durée : 3h.
Documents : tous documents autorisés.

Exercice (6 points)Partie Calculabilité

On considère les machines de Turing avec une tête de lecture seulement, MT_l. C'est-à-dire que de telles machines ne peuvent pas écrire sur le ruban. Une machine de Turing avec une tête de lecture est de la forme $M = (Q, V, \delta, q_0, B, F)$ avec :

- Q un ensemble fini d'états
- V, le vocabulaire, un ensemble fini de symboles
- δ la fonction de transition de profil $Q \times V \rightarrow Q \times \{G, D, N\}$
- q_0 l'état initial ($\in Q$)
- B le symbole "blanc" ($\in V$)
- F est l'ensemble d'états finaux ($F \subseteq Q$)

Le fonctionnement d'une telle machine est le même que celui des machines de Turing classiques, sauf que dans la fonction de transition, il n'est pas prévu de remplacer (réécrire) le caractère lu par la tête de lecture. Une chaîne ω est acceptée par une telle machine si et seulement si il existe une dérivation

$$c_0 \Rightarrow c_1 \Rightarrow \ldots \Rightarrow c_n \not\Rightarrow$$

telle que c_0 est la configuration initiale $q_0\omega$ et c_n est une configuration terminale de la forme $\alpha_1 q \alpha_2$ avec q un état final ($q \in F$) et α_1 et α_2 étant des chaînes sur V ($\alpha_1, \alpha_2 \in V^*$).

▷ **Question 1** (2 points)
Donner une MT_l qui accepte le langage a^*b^*.

▷ **Question 2** (2 points)
Soit M une MT_l quelconque et w une entrée quelconque. Montrer que si M s'arrête pour w alors on peut borner le nombre de transitions effectuées en fonction de la taille de w et du nombre d'états de M.

▷ **Question 3** (2 points)
Soit $M = (Q, V, \delta, q_0, B, F)$ une MT_l et w une chaîne sur V. Le problème " M accepte w " est-il décidable ?

Problème (14 points) Langage Hors-contexte et reconnaisseur

On considère le langage **LB** décrit par la grammaire suivante :

1 program → bloc
2 bloc → decl **begin** suite-inst **end**
3 suite-inst → suite-inst inst **;**
4 suite-inst → inst **;**
5 inst → bloc
6 inst → **idf** := exp
7 exp → **idf**
8 exp → **num**
9 decl → ϵ
10 decl → **declare** suite-idf **: integer ;**
11 suite-idf → **idf ,** suite-idf
12 suite-idf → **idf**

Les éléments de vocabulaire terminal sont notés en gras.

▷ **Question 1** (3 points)
Donner une grammaire LL(1) pour ce langage. On justifiera la réponse en calculant les directeurs de chaque règle. On pourra utiliser des numéros de règles, à condition que la numérotation utilisée soit claire.

▷ **Question 2** (3 points)
Ecrire les procédures d'analyse permettant de reconnaître les blocs (non-terminal bloc) et les suite d'instructions (non-terminal suite-inst). On utilisera les conventions du TP (variable mot-cour contenant le mot courant et fonction lire-mot retournant le prochain mot). **On supposera que les procédures permettant de reconnaître les instructions (non-terminaux inst et exp) et les déclarations (non terminaux decl et suite-idf) sont déjà écrites.**

▷ **Question 3** (3 points)
Le langage **LB** permet d'imbriquer des blocs qui délimitent la durée de vie des variables déclarées dans ces blocs. Soit le programme suivant :

```
1.    declare x : integer ;
2.    begin    x:= 1 ;
3.             declare y : integer ; begin y:=x ; end ;
4.             declare z, u : integer ;  begin u:=x ; z:=u ; end  ;
    end
```

Dans l'exemple précédent la variable `y` ne peut être utilisée que dans le bloc qui la déclare, de même que `z` et `u`. En terme de place mémoire on peut donc réutiliser le même emplacement mémoire pour stocker les variables `y` et pour `z` par exemple. On veut calculer la place mémoire maximum nécessaire à un programme. On supposera que les entiers sont codés sur 4 cases mémoire. Donner un calcul d'attributs permettant d'évaluer le nombre de cases mémoire nécessaires (ici 12 : soit 4 cases pour `x`, 4 pour `y` ou `z` et 4 pour `u`). On travaillera de préférence sur la grammaire initiale.

▷ **Question 4** (2 points)

On modifie la définition des expressions (règles 7 et 8) de la manière suivante :

exp → exp + exp | **idf** := exp | (exp) | **idf** | **num**

L'affectation devient donc maintenant une expression, comme dans le langage C. Montrer que cette grammaire est ambiguë. L'opérateur := étant moins prioritaire que l'opérateur + et l'opérateur + étant associatif à gauche, donner une grammaire non ambiguë respectant cette priorité.

▷ **Question 5** (3 points)

Les programmes d'analyse LL(1) vu en cours s'arrêtent à la première erreur. Cette approche n'est pas réaliste lorsqu'on traite des langages un peu conséquents. L'outil ANTLR propose la stratégie par défaut suivante : si une erreur apparaît dans la procédure A on produit un message d'erreur, on lit jusqu'à trouver un mot dans Suivant(A) (ou la fin de fichier représenté par $) et on stoppe la procédure A.

Reprendre le code de la procédure `bloc` pour implanter cette stratégie. On considère les deux erreurs suivantes (exemple de la question 3) :

- Cas a : oubli du ; après le mot réservé `integer`
- Cas b : oubli du `begin` en ligne 3

Expliquer comment se repositionne l'analyseur pour ces deux cas. Cette stratégie est-elle satisfaisante ? Comment pourrait-elle être améliorée ?