Ensimag 1ère année Mai 2012

Théorie des langages 2

Durée : 3h.
Documents : tous documents autorisés.

Exercice (6 points) Partie Calculabilité

Considérons les fonctions $g : \mathbb{N} \to \mathbb{N}$ et $f : \mathbb{N} \to \mathbb{N}$ définies ci-dessous.

$$g(x) = \begin{cases} 1 & \text{si } MTU(x,x) \text{ s'arrête} \\ 0 & \text{si } MTU(x,x) \text{ ne s'arrête pas} \end{cases}$$

$$f(x) = \begin{cases} (k+1)! & \text{si } MTU(x,x) \text{ s'arrête et k est le nombre de transitions effectuées par } MTU(x,x) \\ 0 & \text{si } MTU(x,x) \text{ ne s'arrête pas} \end{cases}$$

Notation : $n! = 1 \times 2 \times \ldots \times n$

▷ **Question 1 (2 points)**
Montrer que la fonction g n'est pas calculable.

▷ **Question 2 (2 points)**
Déduire de la question précédente que la fonction f n'est pas calculable.

▷ **Question 3 (2 points)** Soit m un entier naturel non nul. Considérons la fonction $f_m : \mathbb{N} \to \mathbb{N}$ définie par $f_m(x) = f(x) \ \ mod \ \ m$.

f_m est-elle calculable ? Justifiez votre réponse !

Notation : $\forall$ `a,r,m` $\in \mathbb{N} \times \mathbb{N} \times \mathbb{N}^*$, `a mod m = r` $\Leftrightarrow \exists$ `q` $\in \mathbb{N}$, `a = m * q + r` et `r < m`

Problème (14 points) Langage Hors-contexte et reconnaisseur

On considère le langage **LB** décrit par la grammaire G 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. L'axiome est le non-terminal program.

▷ **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** (2 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 `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, **sur la grammaire initiale** G, 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`).

▷ **Question 4** (2 points)
Expliquer comment modifier les procédures de la question 2 pour inclure le calcul d'attributs précédent. On illustrera la réponse en donnant le code de la procédure bloc uniquement.

▷ **Question 5** (2 points)

Les programmes d'analyse LL(1) vu en cours s'arrêtent à la première erreur : la détection d'une erreur dans les procédures d'analyse se traduit par la levée d'une exception et arrête l'analyse. Cette approche n'est pas réaliste lorsqu'on traite des langages un peu conséquents. Une stratégie de rattrapage d'erreur consiste à modifier le code des procédures d'analyse en cas de détection d'erreur de la manière suivante : si une erreur est détectée dans la procédure A un message d'erreur est produit, la procédure lit jusqu'à trouver un mot dans $Suivant(A) \cup \{\$\}$, \$ représentant la fin de fichier, puis A rend la main.

Par exemple l'oubli de **:** dans la première ligne de l'exemple de la question 3 se traduira par le message d'erreur " : attendu", l'arrêt de l'appel en cours de la procédure *decl* après repositionnement sur le mot **begin**. La procédure *bloc* ayant lancé l'appel de *decl* continuera alors l'analyse sans erreur supplémentaire.

En utilisant la grammaire LL(1) proposée dans la question 1, donner les valeurs des ensembles Suivant(bloc), Suivant(decl) et Suivant(suite-inst). Reprendre le code de la procédure bloc pour implanter la stratégie décrite ci-dessus.

▷ **Question 6** (2 points)

On considère les deux erreurs suivantes dans l'exemple de la question 3 :

- Cas 1 : oubli du ; après le mot réservé `integer` en ligne 1
- Cas 2 : oubli du `begin` en ligne 3

En supposant que l'analyseur implémente la stratégie précédente pour toutes les procédures d'analyse, expliquer comment se repositionnerait cet analyseur dans ces deux cas ci-dessus : quelle partie de texte est sautée, l'analyse continue t-elle correctement ?

Cette stratégie est-elle toujours satisfaisante ? Comment pourrait-elle être améliorée ?