Université Louis Pasteur
Département d'informatique

Licence d'informatique-semestre 6
année 2007/08

Théorie des langages et calculabilité

Épreuve de contrôle continu

Durée : 2h
Les notes de cours et de TD sont autorisées

Exercice 1

L'algorithme classique de la multiplication (celui que vous avez vu à l'école primaire) se traduit en base 2 par l'algorithme suivant (lui aussi classique) :

```
int mult(int x, int y)
s = 0
tant que x != 0 faire
     si x est impair alors s = s+y
     y = 2*y
     x = x/2
fait
retourner s
```

1- Construisez une machine de Turing de nom M2 qui implante la multiplication par 2 (en binaire), puis une autre machine de Turing de nom D2 qui implante la division par 2 (division entière).

2- On suppose qu'on a une machine de Turing à deux rubans de nom Add capable de faire l'addition de deux entiers notés en binaire, l'un sur le ruban 1 et l'autre sur le ruban 2. À la fin de l'addition, le ruban 1 contient le résultat et le ruban 2 est vide. Utilisez cette machine pour implanter l'algorithme précédent.

3- Définissez une grammaire, de nom G2, qui calcule la fonction $x \mapsto 2x$, puis une grammaire de nom H2, qui calcule la fonction $x \mapsto x/2$.

4- Pour implanter en termes de grammaires générales l'algorithme donné au début, on suppose qu'on a une grammaire de nom Add avec comme symbole initial S_a, et on va "composer" des grammaires en se conformant au schéma suivant où x, y et s sont des entiers en binaires, S,R, T_z, T_i, T_p et S_a sont des symboles non terminaux :

- (i) on transforme $Sx;yS$ en $Sx;yTR0R$, la partie entre les deux non-terminaux R étant destiné à recevoir la somme s de l'algorithme du début ;
- (ii) on teste x en parcourant le mot par un symbole non terminal T et qui se transforme en T_z lorsque x est nul, en T_i lorsque x est impair, et en T_p lorsque x est pair ;
- (iii) on transforme $T_i;yRsR$ en $;yS_ay;sS_a$, pour pouvoir appliquer la grammaire Add ;
- (iv) le marqueur T_z est utilisé à a fin pour "nettoyer" le mot et ne garder que la somme.

Écrivez des règles permettant d'effectuer les transformations décrites à chacun des points précédents. Indiquez comment on doit définir les règles pour que les actions décrites plus haut se déroulent en séquences et que les règles de grammaires ne se mélangent pas.

Exercice 2

1- On considère un alphabet $\Sigma = \{a, b, c\}$, définissez une grammaire générale calculant la fonction de Σ^* dans Σ^* qui à w associe w^r qui est le renversé de w (i.e. w écrit à l'envers).

2- Soit Σ un alphabet, on rappelle qu'un homomorphisme de monoïdes est une application $\varphi : \Sigma^* \to \Sigma^*$ telle que $\varphi(uv) = \varphi(u)\varphi(v)$, et qu'une telle application est uniquement déterminée par les valeurs qu'elle prend

sur les lettres.

a) On considère $\Sigma = \{a, b, c\}$ et φ_1 définie par $\varphi_1(a) = a$, $\varphi_1(b) = e$ et $\varphi_1(c) = c$. Définissez une grammaire calculant φ_1

b) Même question, avec la fonction φ_2 définie par $\varphi_2(a) = ab$, $\varphi_2(b) = e$ et $\varphi_2(c) = cc$

Exercice 3

1- On suppose connue la définition en terme de fonctions primitives récursives, d'une fonction de nom prem testant la primalité d'un nombre entier, i.e. $\text{prem}(n) = 1$ si n est premier et 0 sinon. Montrez, en la définissant, que la fonction qui à n associe le $n^{\text{ème}}$ nombre premier est primitive récursive.

2- Définissez un fonction primitive récursive, de nom is_square testant si un nombre entier est un carré ou non.
Utilisez cette fonction pour montrer que la fonction qui teste si un nombre entier est la somme de deux carrés est primitive récursive.

3- Soit p un prédicat primitif récursif et f une fonction μ-récursive définie par : $f(n) = \mu k[p(k)]$. Montrez que s'il existe une fonction primitive récursive b telle que $f(n) \leqslant b(n)$, alors f est une fonction primitive récursive (I.e. on n'a pas besoin de l'opération de minimisation).
Utilisez ce résultat pour montrer que la fonction qui à tout entier n associe son nombre de chiffres dans l'écriture en base b donnée est une fonction primitive récursive.