Département d'Informatique
LICENCE 2ème année
Mercredi 10 septembre 2008

Contrôle terminal de Programmation Fonctionnelle

Durée : 2h. Documents autorisés : Notes personnelles de cours, de TD et de TP. Utiliser le langage OCAML sans ses éléments impurs. Veiller à la correction et la cohérence des types.

I Récursivité

Dans cet énoncé, un *mot* est une liste non vide $u =$ `[`u_1`; `u_2`; `...`; `u_n`]` d'entiers avec pour tout $i \in \{1, ..., n\}$, $u_i \in \{0, 1\}$. La *longueur d'un mot* est la longueur ($n \geq 1$) de la liste. Exemple : `[0; 1; 1; 1]` est un mot de longueur 4.

On définit la relation d'ordre $\prec$ (ordre lexicographique) sur l'ensemble des mots comme suit : Soient u et v deux mots de longueurs respectives n et m. On a $u \prec v$ si u est le début de v ($n < m$ et $u_i = v_i$ pour tout $i \in \{1, ..., n\}$) ou s'il existe un entier $k \geq 1$ inférieur ou égal à n et à m, tel que les $k - 1$ premiers éléments de u et v sont les mêmes ($u_i = v_i$ pour tout $i \in \{1, ..., k - 1\}$) et le $k$ième élément de u est strictement plus petit que le $k$ième élément de v.

On rappelle que `@` est l'opérateur de concaténation et que `List.length` donne la longueur d'une liste.

1. Définir la fonction `inferieur` de type `'a list -> 'a list -> bool` qui, étant donnés deux mots u et v, retourne vrai si $u \prec v$ et faux sinon.

   `inferieur [0; 0; 1] [0; 1; 0] = true`

2. Un *conjugué* d'un mot $u =$ `[`u_1`; `u_2`; `...`; `u_n`]` est un mot de la forme $u =$ `[`u_i`; `...`; `u_n`; `u_1`; `...`; `u_{i-1}`]` où $2 \leq i \leq n$. Définir la fonction `conjugue` de type `'a list -> int -> 'a list` qui, étant donnés un mot u de longueur n et un entier i tel que $2 \leq i \leq n$, retourne le conjugué du mot u qui débute par le $i$ème élément de u.

   `conjugue [0; 1; 1; 0] 3 = [1; 0; 0; 1]`

3. Un mot u est un *mot de Lyndon* si $u \prec v$ pour tout conjugué v de u. Définir la fonction `lyndon` de type `'a list -> bool` qui détermine si un mot est un mot de Lyndon.

   `lyndon [0; 0; 0; 1] = true`

II Tri par insertion

Dans cette section, on cherche à trouver tous les mots de Lyndon de longueur donnée.

On peut démontrer que tout mot de Lyndon de longueur > 1 est la concaténation f `@` g de deux mots de Lyndon f et g tels que $f \prec g$. Nous déduisons de cette propriété le procédé de calcul suivant : A partir des mots de Lyndon de longueur 1 à savoir `[0]` et `[1]`, on obtient le seul mot de Lyndon de longueur 2 à savoir `[0; 1]`. Les mots de Lyndon de longueur 3 sont obtenus en concaténant deux mots de longueur plus petite est dont la somme des longueurs est 3, et ainsi de suite ...

Longueur 1 : `[0]`, `[1]`
Longueur 2 : `[0; 1]`
Longueur 3 : `[0; 0; 1]`, `[0; 1; 1]`
Longueur 4 : `[0; 0; 0; 1]`, `[0; 0; 1; 1]`, `[0; 1; 1; 1]`

1. Ecrire la fonction `insere_lst` qui prend pour arguments deux listes de mots $lst1$ et $lst2$, la liste $lst2$ étant triée suivant l'ordre lexicographique. Cette fonction renvoie pour résultat, une liste triée de mots obtenue après avoir inséré tous les mots de la liste $lst1$ dans la liste $lst2$. L'insertion n'ayant lieu que si le mot à insérer n'est pas déjà présent.
2. Ecrire la fonction `fusion_lst` qui a pour arguments deux listes $lst1$ et $lst2$ de mots de Lyndon. Cette fonction renvoie pour résultat la liste triée de tous les mots de Lyndon obtenus par concaténation d'un mot de la liste $lst1$ avec un mot de la liste $lst2$. La concaténation f `@` g de deux mots de Lyndon est un mot de Lyndon seulement si $f \prec g$. Par ailleurs, un même mot ne doit pas figurer plusieurs fois dans cette liste résultat.
3. En déduire une fonction `genere` de type `int -> int list list` qui, étant donné un entier $n \geq 1$, retourne la liste de tous les mots de Lyndon de longueur n.

III Lambda-calcul

Réduire ces λ-expressions en utilisant l'ordre normal de réduction (NOR). Indiquer le redex avec le symbole $_\lambda$. Préciser la règle utilisée (α- ou β-conversion) et utiliser la α-conversion uniquement lorsque cela est strictement nécessaire (en cas de problème de capture de variable).

1. $\lambda f.\lambda g.\lambda h.((f\ h)\ (g\ h))\ \lambda h.\lambda i.h\ \lambda h.h$
2. $\lambda f.\lambda g.\lambda h.((f\ h)\ (g\ h))\ (\lambda h.\lambda i.h\ h)\ \lambda h.h$
3. $\lambda f.\lambda g.\lambda h.((f\ h)\ (g\ h))\ (\lambda h.\lambda i.h\ i)\ \lambda h.h$