Composition Hors Classement d'Informatique

Jean-Jacques Lévy

16 décembre 1998 – 9h-11h

Avertissement On attachera une grande importance à la clarté, à la précision, à la concision de la rédaction. Les sections A et B sont indépendantes.

A. Ensembles et relations

Soit $n \geq 0$ un entier. On représente une partie X de l'ensemble $E = \{0, \ldots, n-1\}$ par la liste chaînée de ses éléments en ordre croissant. Une classe déclarée `Partie` correspond à ces listes.

Question 1 Ecrire la fonction `static boolean estEnOrdre(Partie x)` qui teste si X est en ordre croissant.

Question 2 Ecrire la fonction `static Partie union(Partie x, Partie y)` qui calcule l'union de X et Y et `static Partie inter(Partie x, Partie y)` qui calcule leur intersection en temps $O(1 + \operatorname{Card} X + \operatorname{Card} Y)$.

Question 3 Une relation binaire R sur E est représentée par un tableau `r` tel que, pour chaque $i \in E$, `r[i]` pointe sur la liste représentant $\{j \in E \mid (i,j) \in R\}$. Ecrire la fonction `reverse` qui calcule la représentation de la relation $S = \{(i,j) \mid (j,i) \in R\}$. Calculer le temps mis par cette opération.

Question 4 Ecrire une procédure `comp` qui calcule le produit $T = RS = \{(i,j) \mid \exists k \ (i,k) \in R,\ (k,j) \in S\}$. Calculer le temps mis par cette opération.

B. Fonctions récursives primitives

Borobudur est un langage de programmation qui ressemble à Java, mais qui est très simplifié. Les variables x, y, …sont de type entier (positif ou nul). Le langage ne contient que les 5 types d'instructions suivantes:

1.	*affectations simples*	`x = 0;` `x = y+1;` `x = y;`
2.	*séquence*	$S\ S'$
3.	*conditionnel*	`if (x < y)` S `else` S'
4.	*répétition*	`for y do` S
5.	*boucle while*	`while (x < y)` S

où S et S' sont aussi des instructions de Borobudur. Dans les 3ème et 5ème cas, la relation $<$ peut aussi être $>$, $\geq$, $\leq$, $=$, ou $\neq$. On peut parenthéser une séquence d'instructions avec { et }. L'instruction `for y do` S examine la valeur de la variable y et exécute y fois l'instruction S. Si y est modifié à l'intérieur du `for`, cela ne change pas le nombre d'itérations à faire.

On peut montrer que toutes les fonctions calculables peuvent être calculées par Borobudur. Soit Stupa le sous-ensemble de Borobudur qui n'utilise que les instructions 1-2-3-4 (donc sans `while`). Dorénavant nous ne considérons que ce sous-ensemble de programmes. Il est suffisant, par exemple, pour calculer (dans z) le minimum de deux nombres x et y

```
if (x < y)
  z = x;
else
  z = y;
```

De même, on peut calculer l'addition de x et de y

```
z = x;
for y do
  z = z + 1;
```

Question 5 Ecrire les programmes Stupa calculant le produit $x \times y$, la factorielle $x!$, le prédécesseur $x - 1$, la soustraction $x - y$ et la moitié $\lfloor x/2 \rfloor$ (On posera $0 - 1 = 0$ et plus généralement $x - y = 0$ si $x < y$). Donner la complexité du programme dans chaque cas.

Question 6 Peut-on écrire des programmes ne terminant pas en Stupa? En Borobudur?

Les fonctions exprimables en Stupa sont les fonctions connues sous le nom de «récursives primitives». Cette classe $\mathcal{P}$ de fonctions de $\mathbf{N}^n \rightarrow \mathbf{N}$ ($n \geq 1$) contient la fonction constante 0, la fonction successeur et les fonctions de projection. Elle est fermée par composition et par le schéma de récursion primitive récursive. C'est donc la plus petite classe de fonctions contenant:

(1) $Z(x) = 0$
(2) $S(x) = x + 1$
(3) $U_i^n(x_1, x_2, \ldots x_n) = x_i$ pour tout i tel que $1 \leq i \leq n$
(4) $f(g_1(\overline{x}), g_2(\overline{x}), \ldots g_n(\overline{x}))$ si $f, g_1, g_2, \ldots g_n \in \mathcal{P}$
(5) $f(0, \overline{x}) = h(\overline{x})$
$f(x + 1, \overline{x}) = g(x, \overline{x}, f(x, \overline{x}))$ si $g, h \in \mathcal{P}$

où $f(\overline{x})$ est une notation concise pour $f(x_1, x_2, \ldots x_n)$.

Question 7 Donner un exemple de fonction dont le schéma de définition n'est pas récursif primitif.

Question 8 Montrer qu'on peut écrire un programme en Stupa pour calculer toute fonction récursive primitive. (On supposera les arguments dans les variables $x_1, x_2, \ldots x_n$ et le résultat retourné dans la variable x_0).

En Ocaml pour les questions 1-2-3-4, on calculera les fonctions

```
let est_en_ordre x = ... ;;
let union x y = ... ;;
let inter x y = ... ;;
let reverse r = ... ;;   où r.(i) pointe sur la liste ...
let comp r s = ... ;;
```