ISIMA 1ère année

8 février 2007
Durée : 2 heures
Documents autorisés

PROGRAMMATION FONCTIONNELLE

- *Il est permis d'écrire des fonctions auxiliaires.*
- *Penser à expliquer le principe de vos fonctions lorsque celui-ci ne découle pas immédiatement de la lecture de celles-ci.*

Problème :

Etant donnés deux nombres entiers a et b, on notera respectivement q et r le quotient et le reste de la division euclidienne de a par b définis par : $a = bq + r$ avec $0 \le r < b$. Le reste r sera dit égal à "a modulo b" et noté $|a|_b$. On dira également que a est congru à r modulo b.

On supposera que l'on dispose en Scheme de deux fonctions `div` et `mod` ayant comme arguments deux nombres entiers non nuls `a` et `b` et telle que l'évaluation des expressions `(div a b)` et `(mod a b)` retournent respectivement le quotient et le reste de la division euclidienne de `a` par `b`.

Première partie :

On note *Maxint* le plus grand entier représentable en machine. Pour représenter en numération de position un nombre entier N très grand devant *Maxint*, que l'on qualifiera par la suite de "grand entier", il faut utiliser une base b, elle même grande, et une liste de nombres $(c_0, \ldots, c_K)$ tels que $\forall\ k \in \{1, \ldots, K\}$, $0 \le c_k < b$ et $N = \sum_{k=0}^{K} c_k\, b^k$. Les nombres c_k représentent donc les chiffres de ce système de numération. On choisira b telle que $(b-1)^2 \le Maxint$, de façon à ce que la multiplication de deux chiffres puisse se faire sans avoir de problèmes de débordement.

1°) Ecrire une fonction Scheme `m` ayant comme arguments un entier `n` inférieur à b, et un grand entier `GE` représenté par une liste $(c_0, \ldots, c_K)$ et telle que l'évaluation de l'expression `(m n GE)` retourne la liste (`n` $\times c_0, \ldots,$ `n` $\times c_K$).

Solution :

```
(define m (lambda (n GE) (map (lambda (c) (* n c)) GE)))
```

2°) Ecrire une fonction `propag_retenue` ayant comme arguments une liste `E` du type (e_1, …, e_K), avec $\forall\, k \in \{1, \ldots, K\}$, $0 \le e_k \le Maxint$ et un entier `b`, et telle que l'évaluation de l'expression `(propag_retenue E b)` retourne une liste représentant le grand entier égal

à $\sum_{k=0}^{K} e_k\, \mathtt{b}^k$ en base `b`.

Remarque : les nombres e_k sont quelconques entre 0 et *Maxint*, et ne sont donc pas forcément inférieurs à *b*.

Par exemple l'évaluation de l'expression

```
(propag_retenue '(432 945 67 251) 10)
```

doit retourner la liste `(2 8 5 7 6 2)` car

$432 + 945 \times 10 + 67 \times 10^2 + 251 \times 10^3 = 2 + (945 + 43) \times 10 + 67 \times 10^2 + 251 \times 10^3 = \ldots$

$= 2 + 8 \times 10 + 5 \times 10^2 + 267 \times 10^3 = 2 + 8 \times 10 + 5 \times 10^2 + 7 \times 10^3 + 6 \times 10^4 + 2 \times 10^5$.

Solution :

```
(define propag_retenue (lambda (E b)
   (if (cdr E)
      (cons (mod (car E) b)
            (propag_retenue
               (cons (+ (div (car E) b) (cadr E)) (cddr E))
                b ) )
      (if (>= (car E) b)
         (cons (mod (car E) b)
               (propag_retenue (list (div (car E) b)) b) )
         E ) ) ))
```

3°) Ecrire une fonction Scheme `mult` ayant comme arguments un entier `n`, un grand entier `GE` et un entier `b`, représentant la base dans laquelle est représenté `GE`, et telle que l'évaluation de l'expression `(mult n GE b)` retourne le grand entier, représenté lui aussi en base `b`, et égal au produit de `n` par `GE`. L' entier `n` est supposé inférieur à `b`.

Solution :

```
(define mult (lambda (n GE b) (propag_retenue (m n GE) b)))
```

4°) Ecrire une fonction Scheme `plus` ayant comme arguments deux grands entiers `GE1` et `GE2` et un entier `b`, représentant la base dans laquelle sont représentés `GE1` et `GE2`, et telle que l'évaluation de l'expression `(plus GE1 GE2 b)` retourne un grand entier, représenté lui aussi en base `b`, et égal à la somme de `GE1` et `GE2`.

La fonction `plus` devra utiliser une fonction `map` en tenant compte des remarques ci-dessous :

- l'addition étant associative, la fonction scheme + peut admettre un nombre quelconque d'arguments ; appelée avec un seul argument `x`, elle retourne la valeur de `x` ;
- une fonction admettant un nombre quelconque d'arguments peut être “mappée” sur des listes de longueurs différentes.

Solution :

```
(define plus (lambda (GE1 GE2 b)
   (propag_retenue (map + GE1 GE2) b) ))
```

5°) Ecrire une fonction Scheme `plusliste` ayant comme arguments une liste non vide `LGE` de grands entiers, et un entier `b`, représentant la base dans laquelle sont représentés les grands entiers de `LGE`, et telle que l'évaluation de l'expression `(plusliste LGE b)` retourne un grand entier, lui aussi représenté en base `b`, et égal à la somme des grands entiers de `LGE`.

Solution :

```
(define plusliste (lambda (LGE b)
   (if (cdr LGE)
      (plus (car LGE) (plusliste (cdr LGE) b) b)
      (car LGE) ) ))
```

6°) Ecrire une fonction Scheme `f` ayant comme argument une liste `L` d'entiers, par exemple $(x_1, \ldots, x_n)$, et un entier `b`, et telle que l'évaluation de l'expression `(f L b)` retourne une liste de grands entiers représentés en base `b`, $(X_1, \ldots, X_n)$, définis par :

$$\forall\, k \in \{1, \ldots, n\}, X_k = \prod_{j \neq k} x_j \ .$$

Les entiers $x_1, \ldots, x_n$ sont supposés tous inférieurs à `b`.

On suppose également que la longueur de la liste `L` est supérieure ou égale à 2.

Par exemple, l'appel de `f` avec la liste :

- (x_3, x_4) doit retourner (x_4, x_3) ;
- (x_2, x_3, x_4) doit retourner (x_3x_4, x_2x_4, x_2x_3) ;
- (x_1, x_2, x_3, x_4) doit retourner $(x_2x_3x_4, x_1x_3x_4, x_1x_2x_4, x_1x_2x_3)$.

Solution :

```
(define f (lambda (L b)
```

```
    (if (cddr L)
       (let ((rcdr (f (cdr L) b)))
          (cons (mult (cadr L) (car rcdr) b)
               (map (lambda (ge) (mult (car L) ge b)) rcdr) ) )
       (list (list (cadr L)) (list (car L))) ) ))
```

Deuxième partie :

1°) L'algorithme d'Euclide permet de trouver le pgcd de deux nombres entiers non nuls a et b.

Cet algorithme se base sur la propriété suivante :

si $r > 0$, pgcd (a, b) = pgcd (b, r) sinon pgcd (a, b) = min (a, b).

En utilisant une fonction *mod* telle que *mod*(a,b) retourne le reste de la division euclidienne de a par b, cet algorithme peut s'écrire ainsi :

tant que $b > 0$ *faire* ($r \leftarrow mod(a,b)$; $a \leftarrow b$; $b \leftarrow r$) ;

retourner a ;

Ecrire en Scheme une fonction `Euclide` ayant comme arguments deux nombres entiers non nuls `a` et `b` et telle que l'évaluation de l'expression `(Euclide a b)` retourne le pgcd de `a` et `b`.

Solution :

```
(define Euclide (lambda (a b)
    (let ((r (mod a b)))
       (if (> r 0) (Euclide b r) b) ) ))
```

ou bien :

```
(define Euclide (lambda (a b)
    (if (> b 0) (Euclide b (mod a b)) a) ))
```

2°) D'après le théorème de Bezout, étant donnés deux nombres entiers non nuls a et b, il existe deux nombres entiers u et v tels que pgcd $(a, b) = au + bv$.

Notons encore q et r le quotient et le reste de la division euclidienne de a par b. On a alors pgcd $(a, b) = au + bv = (bq + r)u + bv = (uq + v)b + ur$.

Si $r = 0$, pgcd $(a, b) = b = 0 \times a + 1 \times b$. Si $r > 0$ comme pgcd (a, b) = pgcd (b, r) et comme il existe deux nombres entiers u' et v' tels que pgcd $(b, r) = bu' + rv'$, on a donc :

$$\begin{cases} u' = uq + v \\ v' = u \end{cases} \Leftrightarrow \begin{cases} u = v' \\ v = u' - v'q \end{cases}.$$

Ecrire en Scheme une fonction `Bezout` ayant comme arguments deux nombres entiers non nuls `a` et `b` et telle que l'évaluation de l'expression `(Bezout a b)` retourne une liste de trois nombres entiers telle que le premier nombre est le pgcd de `a` et `b`, et les deux suivants deux nombres entiers u et v tels que $\mathtt{a}u + \mathtt{b}v = \text{pgcd}\,(\mathtt{a}, \mathtt{b})$.

<u>Solution</u> :

```
(define Bezout (lambda (a b)
   (let ((r (mod a b)))
      (if (> r 0)
         (let* ((q (div a b)
                (L (Bezout b r))
                (pgcd (car L))
                (uprime (cadr L))
                (vprime (caddr L)) )
            (list pgcd vprime (- uprime (* vprime q))) )
         (list b 0 1) ) ) ))
```

ou bien :

```
(define Bezout (lambda (a b)
      (if (> b 0)
         (let* ((q (div a b)
                (r (mod a b)
                (L (Bezout b r))
                (pgcd (car L))
                (uprime (cadr L))
                (vprime (caddr L)) )
            (list pgcd vprime (- uprime (* vprime q))) )
         (list a 1 0) ) ) ))
```

3°) Etant donnés un nombre premier m et un entier n tel que $0 < n < m$, on appelle inverse de n modulo m, l'entier p tel que $n \times p$ modulo m est égal à 1. Par exemple, l'inverse de 7 modulo 11 est 8 car $7 \times 8 = 56$, qui est bien congru à 1 modulo 11.

Comme on a forcément pgcd $(n, m) = 1$, il existe deux entiers u et v tels que $nu + mv = 1$, et u est donc l'inverse de n modulo m.

Ecrire en Scheme une fonction `inverse` ayant comme arguments deux entiers `n` et `m` et telle que l'évaluation de l'expression `(inverse n m)` retourne l'inverse de `n` modulo `m`.

<u>Solution</u> :

```
(define inverse (lambda (n m) (cadr (Bezout n m))))
```

Troisième partie :

Un théorème chinois du IVème siècle indique que si l'on considère n nombres premiers $m_1, \ldots, m_n$, tout nombre entier x compris entre 0 et $M = \prod_{i=1}^{n} m_i - 1$ peut être représenté sans ambiguïté par la liste $(x_1, \ldots, x_n)$ de ses restes modulo m_i : $\forall\ i \in \{1, \ldots, n\}$, $x_i = x$ modulo m_i.
La liste $(x_1, \ldots, x_n)$ sera appelée la représentation de x dans la base $(m_1, \ldots, m_n)$.

1°) a) Ecrire une fonction Scheme `modg` ayant comme arguments un grand entier `GE`, un entier `m`, et un entier `b`, représentant la base dans laquelle est représenté `GE`, et telle que l'évaluation de l'expression `(modg GE m b)` retourne l'entier égal à `GE` modulo `m`.
L'entier `m` est supposé inférieur à `b`.

Solution :

```
(define modg (lambda (GE m b)
   (if (cdr GE)
      (let ((rcdr (modg (cdr GE) m b)) (bmodm (mod b m)))
         (mod (+ (mod (car GE) m) (mod (* bmodm rcdr) m)) m) )
      (mod (car GE) m) ) ))
```

b) En déduire une fonction Scheme `rep` ayant comme arguments un grand entier `GE`, une liste de nombres premiers `Lmi`, et un entier `b`, représentant la base dans laquelle est représenté `GE`, et telle que l'évaluation de l'expression `(rep GE Lmi b)` retourne le grand entier `GE`, représenté dans la base définie par `Lmi`.
Les entiers de la liste `Lmi` sont supposés tous inférieurs à `b`.

Solution :

```
(define rep (lambda (GE Lmi b)
   (map (lambda (mi) (modg GE mi b)) Lmi) ))
```

2°) La valeur d'un grand entier x peut être retrouvée à partir de sa représentation $(x_1, \ldots, x_n)$ dans la base $(m_1, \ldots, m_n)$ par la formule $x = \sum_{i=1}^{n} x_i \left| M_i \right|_{m_i}^{-1} M_i$, où les M_i sont définis par :

$$\forall\ i \in \{1, \ldots, n\},\ M_i = \prod_{j \neq i} m_j \text{ , et où } \left| M_i \right|_{m_i}^{-1} \text{ représente l'inverse de } M_i \text{ modulo } m_i.$$

a) Ecrire une fonction `g` ayant comme argument une liste `Lmi` de nombres premiers, par exemple $(m_1, \ldots, m_n)$, et un entier `b`, et telle que l'évaluation de l'expression `(g Lmi b)` retourne une liste comportant :

- la liste de grands entiers (M_1, …, M_n) représentés en base `b` ;
- la liste d'entiers $\left(|M_1|_{m_1}^{-1}, \cdots, |M_n|_{m_n}^{-1} \right)$.

Les entiers de `Lmi` sont supposés tous inférieurs à `b`.

On suppose également que la longueur de la liste `Lmi` est supérieure ou égale à 2.

Solution :

```
(define g (lambda (Lmi b)
   (let ((LGMi (f Lmi b)))
      (list LGMi
(map (lambda (GMi mi) (inverse (modg GMi mi b) mi)) LGMi Lmi)
       ) ) ))
```

b) Ecrire une fonction `invrep` ayant comme arguments une liste d'entiers `Lxi`, représentant un grand entier dans une base (m_1, …, m_n), une liste d'entiers `LGMi-1`, correspondant à la liste $\left(|M_1|_{m_1}^{-1}, \cdots, |M_n|_{m_n}^{-1} \right)$, une liste de grands entiers `LGMi`, correspondant à la liste (M_1, …, M_n) et un entier `b`, et telle que l'évaluation de l'expression `(invrep Lxi LGMi-1 LGMi b)` retourne la représentation en base `b` du grand entier représenté par `Lxi` dans la base (m_1, …, m_n).

On suppose que tous les entiers de la liste `Lxi` sont inférieurs à `b`.

Solution :

```
(define invrep (lambda (Lxi LGMi-1 LGMi b)
 (plusliste
   (map (lambda (xi GMi-1 GMi)
           (mult (mod (* xi GMi-1) b) GMi b))
          Lxi LGMi-1 LGMi) ) ))
```

3°) L'avantage de la représentation d'un grand entier dans une base de nombres premiers (m_1, …, m_n) est que les opérations, addition ou multiplication, peuvent se faire terme à terme de manière indépendante et sans propagation de retenue, puisque l'opération entre les $i^{\text{ème}}$ termes est effectuée modulo m_i. Par exemple la multiplication de deux grands entiers représentés respectivement par (x_1, …, x_n) et (y_1, …, y_n) est (z_1, …, z_n) avec :

$$\forall\, i \in \{1, \ldots, n\},\ z_i = x_i \times y_i \text{ modulo } m_i.$$

a) Ecrire une fonction Scheme `opliste` ayant comme arguments une fonction `op`, qui est soit + soit `*`, une liste non vide `LGE` de grands entiers et une liste de nombres premiers `Lmi`, représentant la base de nombres premiers dans laquelle les nombres de `LGE` sont représentés, et telle que l'évaluation de l'expression `(opliste op LGE Lmi)` retourne un grand entier, lui aussi représenté dans la base définie par `Lmi` et égal soit à la somme des grands entiers de `LGE` si la fonction `op` est `+`, soit au produit des grands entiers de `LGE` si la fonction `op` est `*`.

Solution :

```
(define opliste (lambda (op LGE Lmi)
   (if (cdr LGE)
      (let ((rcdr (opliste (cdr LGE) Lmi)))
         (map
         (lambda (xi zi mi) (mod (op xi zi) mi))
           (car LGE) rcdr Lmi ) )
      (car LGE) ) ))
```

b) En déduire une fonction Scheme `opliste_base_b` ayant comme arguments une fonction `op`, qui est soit `+` soit `*`, une liste non vide `LGE` de grands entiers, un entier `b`, représentant la base dans laquelle sont représentés les grands entiers de `LGE`, et une liste de nombres premiers `Lmi`, et telle que l'évaluation de l'expression `(opliste_base_b op LGE b Lmi)` retourne un grand entier, lui aussi représenté en base `b`, et égal soit à la somme des grands entiers de `LGE` si la fonction `op` est `+`, soit au produit des grands entiers de `LGE` si la fonction `op` est `*`. Ce calcul doit être obtenu en passant par la représentation des grands entiers dans la base définie par `Lmi`.

<u>Solution</u> :

```
(define opliste_base_b (lambda (op LGE b Lmi)
   (let* ((LGMiGMi-1 (g Lmi b))
          (LGMi (car LGMiGMi-1))
          (LGMi-1 (cadr LGMiGMi-1))
          (LGE2 (map (lambda (ge) (rep ge Lmi b)) LGE))
          (LGE3 (opliste op LGE2 Lmi)) )
      (invrep LGE3 LGMi-1 LGMi b) ) ))
```