▶ **MPSI – Option Informatique**
Année 2001, Troisième TP Caml
Vincent Simonet (`http://cristal.inria.fr/~simonet/`)

Plus longue sous-liste commune

Pour cette troisième séance nous allons écrire des fonctions qui manipulent des listes en Caml. Nous nous intéresserons en particulier au problème de la plus longue sous-liste commune : si m est une liste, on appelle sous-liste de m une liste obtenue à partir de m en supprimant zéro, un ou plusieurs éléments. Étant données deux listes m_1 et m_2, on cherche alors la longueur de la plus longue liste m qui soit une sous-liste de m_1 et de m_2.

1 Quelques fonctions sur les listes

Nous allons dans cette première partie écrire quelques fonctions simples sur des listes *Caml*. Les fonctions qui manipulent des listes sont généralement écrites en *Caml* de manière récursive en effectuant un filtrage sur la forme de la liste. Par exemple, pour calculer la longueur d'une liste on peut écrire :

```
let rec longueur = function
  [] -> 0
| t :: q -> 1 + (longueur q)
;;
```

Cette fonction est de type $'a\ list \to int$. Elle est *polymorphe* : elle peut être appliqué à des listes d'éléments de n'importe quel type (listes d'entiers, listes de flottants, listes de listes...).

▶ **Question 1** *Écrivez sur le même modèle une fonction* applique *qui prend pour arguments une fonction f et une liste $[x_0; x_1; \ldots; x_{n-1}]$ et retourne la liste $[f(x_0); f(x_1); \ldots; f(x_{n-1})]$.*

▶ **Question 2** *Définissez une fonction qui prend pour argument deux listes et retourne la concaténation de ces deux listes (i.e. la liste formée de tous les éléments de la première liste, suivis de tous les éléments de la seconde liste).*

▶ **Question 3** *Définissez une fonction* max_liste *qui prend pour argument une liste et retourne le plus grand élément de cette liste (pour l'ordre usuel de Caml, $<$). Vous veillerez à ce que votre fonction soit bien polymorphe.*

2 Sous-listes

Une sous-liste de la liste $m = [x_0; \ldots; x_{n-1}]$ est une liste obtenue en supprimant certains éléments de m (et en conservant l'ordre), c'est-à-dire une liste (éventuellement vide) de la forme $[x_{\sigma(0)}; \ldots; x_{\sigma(k-1)}]$ avec $0 \leq \sigma(0) < \ldots < \sigma(k-1) \leq n-1$.

Par exemple, $[0; 1; 2; 2; 3; 3]$ est une sous-liste de $[4; 0; 1; 2; 2; 2; 2; 3; 4; 3]$.

▶ **Question 4** *Écrivez une fonction* est_sous_liste *qui prend pour argument deux listes m_1 et m_2 et teste si m_1 est une sous-liste de m_2.*

Étant donné une liste m, on note $\Sigma(m)$ l'ensemble des sous-listes de la liste m.

▶ **Question 5** *Soit m une liste telle que $m = t :: q$. Montrez que $\Sigma(m) = \Sigma(q) \sqcup \{t :: r \mid r \in \Sigma(q)\}$.*

▶ **Question 6** *Déduisez-en une fonction* sous_listes *qui prend pour argument une liste m et calcul l'ensemble $\Sigma(m)$ (cet ensemble sera retourné sous forme d'une liste de listes, la même sous-liste pouvant éventuellement être répétée plusieurs fois).*

3 Plus longue sous-liste commune

On s'intéresse maintenant au problème suivant : étant données deux listes m_1 et m_2, trouver la longueur de la plus longue liste m qui soit à la fois une sous-liste de m_1 et une sous-liste de m_2 (on s'intéresse dans un premier temps seulement à la longueur de cette sous-liste).

▶ **Question 7** *Quelle est la longueur de la plus longue sous-liste commune à* $[0;1;0;0;1]$ *et* $[0;0;1;1]$ *? Retrouvez ce résultat en utilisant les fonctions que nous avons définies jusqu'à présent.*

3.1 Une méthode de calcul « naïve »

Soient $m_1 = [x_0; \ldots; x_{p-1}]$ et $m_2 = [y_0; \ldots; y_{q-1}]$ deux listes. On note $\ell(i,j)$ la longueur de la plus longue sous-liste commune de $[x_0; \ldots; x_i]$ et $[y_0; \ldots; y_j]$.

▶ **Question 8** *Montrer que l'on a la relation suivante :*

$$\ell(i,j) = \max(\ell(i-1,j-1) + \delta(x_i, y_j), \\ \ell(i,j-1),\ \ell(i-1,j-1))$$

où $\delta(x,y) = 1$ *si* $x = y$ *et* 0 *sinon.*

▶ **Question 9** *Déduisez-en une fonction* `plslc` *qui calcule la longueur de la plus longue sous-liste commune de deux listes.*

▶ **Question 10** *Évaluez le nombre d'appels récursifs effectués par votre fonction pour calculer la longueur de la plus longue sous-liste commune de deux listes de longueurs respectives* p *et* q*.*

3.2 Programmation dynamique

La fonction `plslc` que nous avons écrite recommence de nombreuses fois le même calcul. Pour éviter cela, il suffit de stocker les résultats intermédiaire dans une matrice *sol* : la case $(i+1, j+1)$ de cette matrice contiendra tout simplement la longueur d'une plus longue sous-liste commune de $[x_0; \ldots; x_i]$ et $[y_0; \ldots; y_j]$, c'est-à-dire $\ell(i,j)$.

▶ **Question 11** *Expliquez comment on peut remplir de proche en proche le tableau sol.*

▶ **Question 12** *Écrivez une fonction* `plslc2` *qui implante cette méthode. Vous pourrez, si cela vous semble plus efficace, convertir les listes en vecteurs à l'aide de la fonction* `vect_of_list`.

▶ **Question 13** *Sauriez-vous adapter votre fonction de telle sorte qu'elle retourne l'une des plus longues sous-listes communes ? toutes les plus longues sous-listes communes ?*

▶ **MPSI – Option Informatique**
Année 2001, Troisième TP Caml
Vincent Simonet (`http://cristal.inria.fr/~simonet/`)

Plus longue sous-liste commune

Un corrigé

▶ Question 1

```
let rec applique f = function
    [] -> []
  | t :: q -> (f t) :: (applique f q)
;;
```

Cette fonction est prédéfinie dans la bibliothèque standard de *Caml* sous le nom map.

▶ Question 2

```
let rec concatene m1 m2 =
  match m1 with
    [] -> m2
  | t :: q -> t :: (concatene q m2)
;;
```

La concaténation est prédéfinie dans la bibliothèque standard de *Caml* comme l'opérateur infixe @.

▶ Question 3

```
let rec max_liste = function
    [] -> failwith "liste vide"
  | t :: [] -> t
  | t :: q -> max t (max_liste q)
;;
```

Dans le cas où la liste passée en argument est vide, on déclenche une exception. Le « vrai » cas de base est celui de la liste de longueur un.

▶ Question 4

```
let rec est_sous_liste m1 m2 =
  match m1, m2 with
    [], _ -> true
  | _, [] -> false
  | t1 :: q1, t2 :: q2 ->
    if t1 = t2 then est_sous_liste q1 q2
    else est_sous_liste m1 q2
;;
```

Si m_1 est vide, c'est une sous-liste de m_2 (1^{er} cas). Si m_2 est vide et m_1 n'est pas vide, m_1 n'est pas une sous-liste de m_2 ($2^{\text{ème}}$ cas). Enfin si les deux listes sont non-vides ($3^{\text{ème}}$ cas), on distingue deux possibilitées :

- Si elles ont même tête ($t_1 = t_2$), m_1 est sous-liste de m_2 si et seulement si q_1 est sous-liste de q_2.
- Si elles n'ont pas la même tête ($t_1 \neq t_2$), m_1 est sous-liste de m_2 si et seulement si m_1 est une sous-liste de q_2.

▶ Question 6

```
let rec sous_listes = function
    [] -> [[]]
  | t :: q ->
    let e = sous_listes q in
    e @ (map (fun m -> t :: m) e)
;;
```

Dans le cas où la liste est vide, on retourne tout simplement [[]] : il n'y a qu'une seule sous-liste, la liste vide (c'est important de ne pas renvoyer [] car sinon, par récursion, on retournera toujours []).
Dans les autres cas, e représente l'ensemble $\Sigma(q)$. L'expression (map (**fun** m -> t :: m) e) permet de calculer $\{t :: r \mid r \in \Sigma(q)\}$. Pour obtenir le résultat final, il suffit de concaténer à l'aide de @ les deux listes ainsi obtenues.

▶ Question 7

Une fois les fonctions précédentes définies, on peut calculer la longueur de la plus longue sous-liste de la manière suivante :

```
# max_liste (applique list_length
  (intersect (sous_listes [0; 1; 0; 0; 1])
             (sous_listes [0; 0; 1; 1])))
  ;;
- : int = 3
```

On calcule l'ensemble des sous-listes de chaque liste, puis leur intersection. On recherche ensuite la sous-liste de longueur maximale.

▶ Question 9

```
let rec plslc m1 m2 =
  match m1, m2 with
    [], _ -> 0
  | _, [] -> 0
  | t1 :: q1, t2 :: q2 ->
    max ((plslc q1 q2) + (if t1 = t2 then 1 else 0))
      (max (plslc m1 q2) (plslc q1 m2))
;;
```

▶ **Question 11** Grâce à la relation de récurrence sur ℓ, on peut remplir le tableau suivant les diagonales croissantes.

▶ **Question 12**

```
let plslc2 m1 m2 =
  let delta x y = if x = y then 1 else 0 in
  let t1 = vect_of_list m1
  and t2 = vect_of_list m2
  in
  let n1 = vect_length t1
  and n2 = vect_length t2
  in
  let sol = make_matrix (n1+1) (n2+1) 0 in

  for d = 0 to n1 + n2 do
    for i = 1 to n1 do
      let j = d - i in
      if 1 <= j && j <= n2 then
        sol.(i).(j) <- max
          (sol.(i-1).(j-1) + (delta t1.(i-1) t2.(j-1)))
          (max sol.(i-1).(j) sol.(i).(j-1))
    done;
  done;
  sol.(n1).(n2)
;;
```