TP n° 1 – Récursivité, listes

vie@jill-jenn.net/tp
thomas.h@ens.fr

Remarques. Nous avons piégé les touches @ de tous les claviers, n'appuyez pas dessus (sauf pour l'exercice 7, voire pour nous écrire). Efforcez-vous d'écrire des fonctions récursives terminales.

1 Oups !

Question 1. Yorel Reivax implémente le code suivant, qui crée une fonction `fibo0` telle que `fibo0 n` retourne le n-ième élément de la suite de Fibonacci.

$$u_n = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ u_{n-1} + u_{n-2} & \text{sinon.} \end{cases}$$

```
let rec fibo0 n = match n with
  | 0 -> 0
  | 1 -> 1
  | n -> fibo0 (n - 1) + fibo0 (n - 2)
;;
```

Recopier son code, et tester la fonction `fibo0` sur des valeurs croissantes de n. Pourquoi est-ce si lent ? Implémenter une fonction `fibo` plus efficace.

2 Listes

Question 2. Écrire une fonction `rev1` qui inverse une liste passée en argument.

Question 3. Écrire une fonction `map1` telle que `map1 f [x1;x2;...;xn]` retourne la liste `[f x1;f x2;...;f xn]`.

Question 4. Écrire une fonction `filter` telle que `filter p l` retourne la liste des éléments de `l` qui satisfont le prédicat `p : x -> bool` où `x` est le type des éléments de `l`.

Question 5. Écrire une fonction `fold_left` telle que `fold_left f a [x1;x2;...;xn]` retourne `f (...(f (f a (x1)) x2)...) xn`.

Question 6. Écrire une fonction `flatten` de type `'a list list -> 'a list` telle que `flatten l` retourne la concaténation des éléments de `l`. Par exemple, `flatten [[1;2];[3];[];[4;5]] = [1;2;3;4;5]`.

3 'a de Fibonacci

Question 7. On s'intéresse à écrire une fonction plus générique `fibogen` telle que `fibogen f0 f1 op n` retourne le n-ième élément de la suite définie par la relation suivante, où * représente l'opérateur `op` :

$$u_n = \begin{cases} \texttt{f0} & \text{si } n = 0 \\ \texttt{f1} & \text{si } n = 1 \\ u_{n-1} * u_{n-2} & \text{sinon.} \end{cases}$$

Quel sera le type de cette fonction ? Vérifier que `fibogen 0 1 prefix +`[1] est bien une fonction calculant le n-ième terme de la suite de Fibonacci. On la notera `fibo`.

1. Vous pourrez exécuter `prefix +`, puis `prefix + 4 2` pour comprendre ce que signifie le mot-clé `prefix`. Notez que cet appel est équivalent à `fibogen 0 1 (prefix +)`.

Question 8. Quels sont les paramètres à donner à la fonction `fibogen` pour obtenir les fonctions `fiboword` et `fibolist` correspondant aux suites suivantes ?

$w_0 = b$	$\ell_0 =$ `[1]`
$w_1 = a$	$\ell_1 =$ `[0]`
$w_2 = ab$	$\ell_2 =$ `[0;1]`
$w_3 = aba$	$\ell_3 =$ `[0;1;0]`
$w_4 = abaab$	$\ell_4 =$ `[0;1;0;0;1]`
$w_5 = abaababa$	$\ell_5 =$ `[0;1;0;0;1;0;1;0]`
...	...

Question 9. Écrire une fonction `nb_occ` qui détermine le nombre d'occurrences d'un élément d'une liste [2]. Vérifier pour quelques valeurs de n que `nb_occ 0` ℓ_n `= fibo n` puis démontrer cette propriété.

Question 10. Écrire une fonction `tronque` telle que `tronque n l` retourne `l` privée de ses `n` derniers éléments. Vérifier pour quelques valeurs de n, de préférence en vous aidant de `map1`, que ℓ_n privée de ses 2 derniers éléments est toujours un palindrome puis le démontrer.

Question 11. Écrire une fonction `remplace` telle que `remplace x lx l` remplace chaque occurrence de `x` dans `l` par la liste `lx` [3]. Vérifier pour quelques valeurs de n qu'en remplaçant tous les `0` par `[0;1]` et tous les `1` par `[0]` dans ℓ_n, on obtient ℓ_{n+1} [4] puis le démontrer.

4 Cryptanalyse

Question 12. Implémenter l'algorithme d'exponentiation rapide. Votre fonction devra être récursive terminale.

Question 13. Une implémentation récursive non terminale de l'algorithme effectue une séquence d'opérations *square* et *multiply* :

```
type op = S | M;;
```

Par exemple, lors du calcul de a^{27}, la séquence d'opérations est `SMSMSSMSM` :

$$1 \xrightarrow{\texttt{S}} 1 \xrightarrow{\texttt{M}} a \xrightarrow{\texttt{S}} a^2 \xrightarrow{\texttt{M}} a^3 \xrightarrow{\texttt{S}} a^6 \xrightarrow{\texttt{S}} a^{12} \xrightarrow{\texttt{M}} a^{13} \xrightarrow{\texttt{S}} a^{26} \xrightarrow{\texttt{M}} a^{27}.$$

Écrire une fonction qui prend en argument un exposant et retourne la liste d'opérations associée.

Question 14. Lorsque l'algorithme de déchiffrement du RSA déchiffre un message m, il effectue l'opération $c = m^d$ où d est la clé secrète. Shamir a posé un microphone sur un microprocesseur exécutant un algorithme d'exponentiation rapide. Il lui a fait déchiffrer 1 et -1, et écoute si les opérations de *square* et *multiply* font le même bruit (`true`) ou un bruit différent (`false`). À partir de la séquence de `true` et `false`, retrouver les valeurs possibles de la clé.

2. Par exemple, `nb_occ 'r' ['o';'c';'c';'u';'r';'r';'e';'n';'c';'e']` doit retourner 2.
3. Par exemple, `remplace 'r' ['r';'r'] ['o';'c';'c';'u';'r';'e';'n';'c';'e']` doit retourner `['o';'c';'c';'u';'r';'r';'e';'n';'c';'e']`.
4. Merci de ne pas répondre à cette question par `remplace 1 [0] (remplace 0 [0;1] l)`.