TD 1 : Haskell – Introduction

1 Fibonacci

Écrire une fonction qui étant donné un paramètre `n` retourne le −n-ème terme de la suite de Fibonacci. Rappel : la suite de Fibonacci est définie comme $1, 1, 2, 3, 5, 8, 13, 21, \ldots$, où chaque terme est la somme des deux termes précédents.

2 Opérations Élémentaires

Considérez les fonctions basiques suivantes qui manipulent des listes :

1. `length` retourne la taille d'une liste
2. `take n l` retourne une liste qui contient les `n` premiers éléments de `l`
3. `drop n l` retourne une liste qui contient tous les éléments de `l` sauf les `n` premiers
4. `(!!) l n` retourne l'élément qui se trouve en position `n` en `l` (on commence la numérotation à 0)
5. `(++) l1 l2` retourne la concaténation des deux listes `l1,l2`
6. `reverse l` retourne la liste `l` en ordre inverse.
7. `concat ll` prend comme argument une liste de listes et retourne leur concaténation
8. `elem x l` retourne `True` si l'élément `x` paraît dans la liste `l`.

Toutes ces fonctions sont déjà définies en Haskell. Cependant, donnez des redéfinitions avec des fonctions récursives qui n'utilisent que les opérations élémentaires `head, tail` et le pattern matching.

3 Sous-ensembles

Écrire une fonction qui, étant donné une liste l produit une liste qui contient toutes les sous-listes de l. Par exemple, pour la liste $[1, 2, 3]$ votre fonction doit retourner $[[1], [2], [3], [1, 2], [1, 3], [2, 3], [1, 2, 3], []]$ (dans n'importe quel ordre).

4 Multiplication de vecteurs

Écrire une fonction qui étant donné deux listes de la même taille calcule leur produit, c'est-à-dire, si les deux listes sont $a = [a_1, a_2, \ldots, a_n]$ et $b = [b_1, b_2, \ldots, b_n]$ votre fonction doit retourner $\sum_{i=1}^{n} a_i b_i$.

5 Matrices et Colonnes

Pour cet exercice on suppose qu'on représente de r lignes et c colonnes comme une liste de taille r dont chaque élément est une liste de taille c qui représente une ligne de la matrice.

Par exemple, la matrice $A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ correspond à la liste $[[1, 2, 3], [4, 5, 6]]$.

Étant donné une matrice `m` dans cette forme c'est très facile d'obtenir la ligne `i` en écrivant `m !! i`. Écrire une fonction qui fait la même chose pour les colonnes, c'est-à-dire, une fonction `getcol m j` qui, étant donné une matrice `m` et un entier `j` retourne une liste qui contient les éléments de la colonne `j`.