BE de programmation fonctionnelle

1h30 : avec documents

Année 2019-2020

Préambule

- Le code rendu doit impérativement compiler. Pour cela, les fonctions non implémentées peuvent être remplacées par un code quelconque, par exemple **let** evalue = **fun** _ -> **assert** false .
- Vous devez tout écrire dans le fichier `be.ml`.
- Les noms et types des fonctions doivent être respectés (tests automatiques).
- Pour tester dans `utop` vous devez ouvrir le module Be (**open** Be;;).
- La non utilisation d'itérateur sera pénalisée ainsi que l'utilisation inutile d'accumulateurs.
- Les exercices sont indépendants.

1 Manipulation des listes

Nous allons modéliser les polynômes par une liste de coefficients réels (ici donc des float). Le polynôme

$$a_0 + a_1 x + a_2 X^2 + ... + a_n X^n$$

avec $a_n \neq 0$, est modélisé par la liste [a0;a1;a2 ;...; an].
Dans le fichier `be.ml`, p0, p1, p2 et p3 correspondent, respectivement à

2.5
$2 + 3X$
$2 + 5.5X - 2X^2$
$4.5 + 6X - 3.5X^2 - 8X^3$

▷ **Exercice 1**

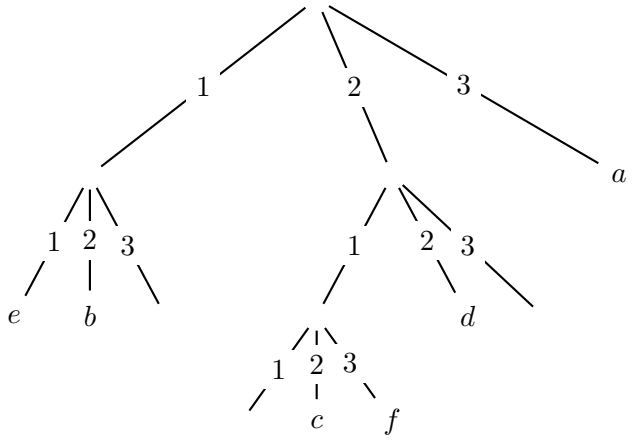
- *Écrire le contrat, les tests unitaires et la fonction* evalue *qui évalue un polynôme pour une valeur donnée de x.*
  Aide : Utiliser le schéma de Horner

$$evalue\ x_0\ [a_0; a_1; ...; a_{n-1}; a_n] = ((...((a_n x_0 + a_{n-1})x_0 + a_{n-2})x_0 + ...)x_0 + a_1)x_0 + a_0$$

- *Écrire la fonction* retract *qui prend une liste de coefficients et supprime, s'il y en a les zéros inutiles. La liste d'entrée n'est pas une représentation correcte d'un polynôme car nous avons demandé que* $a_n \neq 0$*, la liste de sortie l'est.* List.rev *non autorisée.*
- *Écrire la fonction* scal_mult *qui multiplie un polynôme par un scalaire. Le retour doit être un polynôme.*
- *Écrire la fonction* plus *qui additionne deux polynômes. Le retour doit être un polynôme.*

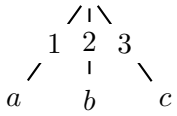
2 Manipulation des arbres

Les arbres peuvent être utiliser pour coder des mots. Nous allons voir une façon de les utiliser. Le principe est simple : les caractères sont les feuilles d'un arbre et le chemin pour atteindre le caractère depuis la racine donne sa façon de l'encoder.
Par exemple en utilisant l'arbre de codage suivant :



'a' se code 3, 'b' se code 12, 'c' se code 212, ... Le mot "bac" se code 123212.

Mais en utilisant



le mot "bac" se code 213.

Un arbre de profondeur p dont les nœuds ont au plus n fils ayant au plus n^p feuilles, il est possible de représenter toutes les lettres de l'alphabet grâce à un arbre ternaire de profondeur maximale 3 ($3^3 = 27$) .
Nous représenterons donc les arbres d'encodage par le type OCaml suivant :
type arbre_encodage = Vide | Lettre **of** char | Noeud **of** arbre_encodage * arbre_encodage * arbre_encodage.
Nous ne mettons pas de donnée dans les branches : la première branche correspondra à 1, la seconde à 2 et la troisième à 3. Il serait très facile de généraliser.
Le premier exemple d'arbre d'encodage est représenter en OCaml par :

```
let arbre_sujet =
  Noeud (
          Noeud (Lettre 'e', Lettre 'b', Vide) ,
          Noeud (
                  Noeud (Vide, Lettre 'c', Lettre 'f'),
                  Lettre 'd',
                  Vide),
          Lettre 'a'
        )
```

▷ **Exercice 2**

— *Écrire une fonction* decoder *qui prend un entier (*code*) et un arbre d'encodage et renvoie la chaîne de caractère associée :* decoder 123212 arbre_sujet = "bac".
Aide : vous pourrez utiliser une fonction auxiliaire aux_decoder qui travaille sur une liste d'entiers (qui sera générée à partir de code en utilisant la fonction decompose_int) et qui renvoie liste de caractères. La chaîne sera alors recomposée à l'aide de recompose_chaine.
aux_decoder [1;2;3;2;1;2] arbre_sujet = ['b';'a';'c'].

— *Écrire une fonction* arbre_to_liste *qui renvoie, pour un arbre d'encodage donné, la liste qui associe à un caractère son code.*
arbre_to_liste arbre_sujet = [('e', 11); ('b', 12); ('c', 212); ('f', 213); ('d', 22); ('a', 3)] *ou une permutation de cette liste.*

— *Écrire une fonction* encoder *qui prend une chaîne de caractères et un arbre d'encodage et renvoie le code associé :* encoder "bac" arbre_sujet = 123212.
Aide : vous pourrez utiliser la fonction arbre_to_liste ainsi qu'une fonction auxiliaire encoder_char qui pour un caractère renvoie son code s'il existe.

▷ **Exercice 3** ***BONUS***

— *Définir un itérateur* fold *sur la structure précédente. Donner son type. Les tests unitaires ne sont pas demandés (sera validé avec les fonctions suivantes).*
Aide : Un fold*, en plus de la structure sur laquelle il itère, prend en paramètre une fonction par constructeur du type (fonction de même arité que le constructeur).*

— *Écrire une fonction* nbLettres *permettant de calculer le nombre de lettres dans l'arbre de codage, à l'aide de l'itérateur précédent.*

— *Écrire une fonction* lettres *permettant de construire la liste des lettres présentes dans l'arbre de codage, à l'aide de l'itérateur précédent.*

— *Écrire une fonction* arbre_to_liste_2 *, de même spécification que* arbre_to_liste *, à l'aide de l'itérateur précédent.*