TP OCaml

1 Environnement de travail

Je vous propose deux options en ligne :

- le site `https://try.ocamlpro.com/`, qui propose une console interactive appelée « Toplevel », dans laquelle vous pouvez exécuter des phrases OCaml pour observer le résultat de leur évaluation, et un éditeur dans lequel vous pouvez écrire votre code et dont vous pouvez sauver le contenu ou bien l'envoyer vers le Toplevel pour exécution ;
- si vous préférez une interface type « Jupyter Notebook », vous pouvez vous rendre sur le site `https://notebook.basthon.fr/ocaml/`. Dans l'académie, nous avons également accès depuis peu à Capytale via Éclat.

2 Début en douceur

2.1

Écrire une fonction valeur absolue `val_abs_int` sur les entiers, de type `int -> int`.

Quelles sont les modications à apporter pour définir une fonction `val_abs_float` sur les flottants, de type `float -> float` ? Le faire effectivement.

2.2

Écrire une fonction `quadruple` de type `int -> int` qui définira une fonction locale `double` de type `int -> int` et qu'elle utilisera pour calculer la valeur égale à quatre fois celle reçue en paramètre.

2.3

Écrire une fonction OCaml de chacun des types suivants :

1. `int -> int`
2. `int * int -> int`
3. `int -> int -> int`
4. `int -> int * int`
5. `(int -> int) -> int`

2.4

1. Écrire une fonction `f1 : (float -> float) -> float` qui associe à une fonction f la valeur :

$$\frac{f(5) + f(-5)}{2}$$

2. Écrire une fonction `f2 : (float -> float) * float -> float` qui prend en entrée une fonction f et un flottant x et renvoie $f(x)^2$.
3. Soit `f3` la fonction qui prend en entrée une fonction f `: float -> float` et qui renvoie la fonction $x \mapsto f(2x)$. Quel est le type de `f3` ? Écrire `f3`.
4. Écrire une fonction `somme` qui prend en entrée deux fonctions `f` et `g` de type `int -> int`, et renvoie la somme de ces deux fonctions.
5. Écrire une fonction `produit` qui prend en entrée deux fonctions `f` et `g` de type `int -> int`, et renvoie le produit de ces deux fonctions.
6. Écrire une fonction `produit_ext : (int -> int) -> int -> int -> int` qui prend en entrée une fonction f ainsi qu'un entier a, et renvoie la fonction $a \times f$.
7. Écrire une fonction `derive : (float -> float) -> float -> float -> float` qui prend en entrée une fonction f et un réel ϵ et renvoie la fonction :

$$x \mapsto \frac{f(x+\epsilon) - f(x)}{\epsilon}.$$

Que doit renvoyer le code suivant ?

```
let f x  = x *. x *. x +. 2. *. x +. 1. in
derive f (10. ** (-. 5.)) 2.;;
```

3 Récursivité et listes

On représente un ensemble (fini) d'objets par la liste de ses éléments. La liste vide représente donc l'ensemble vide. Une liste représente un ensemble si et seulement si elle est sans doublons.

1. Écrire une fonction `valide` de type `'a list -> bool` qui calcule le booléen indiquant si une liste représente effectivement un ensemble.
2. Écrire une fonction `unique`, qui supprime les doublons d'une liste :

   `unique [1; 2; 1; 0; 3; 2]` calcule la liste `[1; 2; 0; 3]` ou toute autre liste égale à celle-ci à une permutation près.
3. Écrire les fonctions `union`, et `intersection` qui opèrent sur les ensembles (on pourra bien sûr réutiliser la fonction `unique` précédente si nécessaire).
4. Écrire une fonction `inclus` qui calcule le booléen indiquant si un ensemble est inclus dans un autre.
5. Écrire la fonction `egal` qui calcule le booléen indiquant si deux ensembles sont égaux.

4 Types inductifs

1. Définir un type `'a bin_tree` pour les arbres binaires étiquetés par des éléments de type `'a`.
2. Écrire une fonction `height : 'a bin_tree -> int` qui calcule la hauteur d'un arbre binaire.

3. Écrire une fonction `infix_dfs : 'a bin_tree -> 'a list` qui renvoie le parcours en profondeur infixe de l'arbre binaire passé en argument.
4. Écrire une fonction `is_bst : 'a bin_tree -> bool` qui détermine si son paramètre est un arbre binaire de recherche.
5. Écrire les opérations usuelles sur les arbres binaires de recherche, à savoir :
   (a) une fonction `mem : 'a -> 'a bin_tree -> bool` testant l'appartenance d'un élément ;
   (b) une fonction `max : 'a bin_tree -> 'a` renvoyant l'étiquette maximale.
   On pourra renvoyer une erreur grâce à `failwith "Empty tree"` si l'arbre est vide.
   (c) une fonction `add : 'a -> 'a bin_tree -> 'a bin_tree` insérant un élément ;
   (d) une fonction `del : 'a -> 'a bin_tree -> 'a bin_tree` supprimant un élément.
6. Pour ceux qui ont du temps devant eux : on veut garder nos arbres binaires de recherche équilibrés. Implémenter les arbres bicolores.