

Interrogation écrite de Programmation Fonctionnelle

Durée : 2h. Aucun document autorisé. Toutes les fonctions sont à écrire en OCAML . Ne pas utiliser d'éléments impurs !

I Récursivité

1. (Puissance entière). Définir une fonction `puiss` qui calcule la puissance entière d'un nombre réel (x^n avec x , un réel et n , un entier naturel). On se servira de l'équation suivante :

$$x^n = x^{(n/2)} \times x^{(n/2)} \quad \text{si } n \text{ pair}$$

et on utilisera une définition temporaire (`let ... in ...`) pour éviter de calculer plusieurs fois la même valeur.

2. (Série entière). Définir une fonction `serie` dont le type est :
`int -> (int -> float) -> float -> float` et qui calcule la somme :

$$\sum_{i=0}^n a_i x^i$$

où n est un entier naturel, (a_n) est une suite de réels et x est un réel.

3. Se servir de la fonction `serie` de la question précédente pour écrire une phrase OCAML donnant une valeur approchée de π :

$$\frac{\pi}{4} \approx \sum_{i=0}^{100000} \frac{(-1)^i}{2i+1}$$

On utilisera la fonction prédéfinie `float_of_int` : `int -> float` pour convertir un entier en réel.

4. L'expression `serie 1 (function n -> 1.)` est-elle bien typée ? Si oui, préciser son type et sa valeur.

II Algorithme de typage

Dérouler l'algorithme de typage vu en Cours et en TD pour trouver le type de cette fonction : `let app = fonction f -> fonction x -> f x;;`. (On remplira 3 colonnes : une pour l'ensemble \mathcal{H} des hypothèses, une pour l'ensemble \mathcal{E} des équations entre types et une pour l'ensemble \mathcal{T} des expressions à typer).

III Types structurés

(Dans cet exercice, on utilisera la notation prédéfinie pour les listes.)

1. Écrire une fonction `take` qui, étant donné un entier `n` et une liste `l` retourne les `n` premiers éléments de `l`. Exemples :

```
take 3 [1;2;3;4;5;6] = [1;2;3]
take 0 [1;2;3;4;5]   = []
take 2 []             = []
```

2. Écrire une fonction `drop` qui, étant donné un entier `n` et une liste `l` retourne la liste `l` sans ses `n` premiers éléments. Exemples :

```
drop 3 [1;2;3;4;5;6] = [4;5;6]
drop 0 [1;2;3;4;5]   = [1;2;3;4;5]
drop 2 []             = []
```

3. En déduire une fonction `partage` qui scinde une liste en 2 moitiés de longueurs approximativement égales.

(On utilisera la fonction prédéfinie `List.length` qui retourne la longueur d'une liste). Exemples :

```
partage [1;2;3;4;5;6] = ([1;2;3], [4;5;6])
partage [1;2;3;4;5]   = ([1;2], [3;4;5])
```

4. Écrire une fonction `fusion` qui fusionne 2 listes triées et retourne une liste triée. Exemple :

```
fusion ([1;2;4], [3;5;6]) = [1;2;3;4;5;6]
```

5. (Tri fusion). Écrire une fonction `tri` qui trie une liste récursivement de la façon suivante : la liste est partagée en 2 moitiés approximativement de même longueur, puis on fusionne chaque moitié préalablement triée.

IV Vous pensiez avoir fini ?

1. Définir en OCAML le type des *expressions ensemblistes*. Une expression ensembliste est soit la notation de l'ensemble vide (`{}`), soit celle d'un singleton (un ensemble à un élément) (`{x}`), soit celle de l'union de 2 ensembles ($A \cup B$), soit celle de l'intersection de 2 ensembles ($A \cap B$). On conviendra que les éléments d'un ensemble sont tous du même type et que ce type est quelconque.
2. Écrire en OCAML une valeur dont le type est celui de la question 1 et correspondant à l'ensemble `{1, 2}`.