ENSEIRB-Matmeca
2024-2025

1ère année - Filière Info
PG104

TD n°6 - 1ère classe : généralisation, spécialisation

▷ Les listes chaînées ne font pas partie de la bibliothèque standard d'Ecmascript. Afin de pouvoir réaliser les algorithmes récursifs de cette feuille, les fonctions suivantes permettent de construire et déconstruire des listes :

```
import { anyToString } from '#src/utils/printers.js';
import { head, tail, nil, cons, isEmpty } from '#src/utils/list.functional.api.js';

let aList = cons(1, cons(2, nil));
console.log(anyToString(aList));          // → (|1, 2|)
console.logA(aList);                      // → (|1, 2|)

let anotherList = cons(3, tail(aList));
console.logA(anotherList)                 // → (|3, 2|)
```

▷ La bibliothèque `list.functional.api.js` contient aussi des fonctions pour construire des listes (`listIota`) ainsi que les transformer en chaînes de caractères (`listToString`).

Exercice 1: Des filtres, ça vaut la peine d'en rajouter

1. Écrire une fonction récursive `listFilterSup` prenant en paramètre une liste de nombres et un nombre `val`, et renvoyant une nouvelle liste ne conservant que les nombres strictement supérieurs à `val`.

Pour rappel, une *généralisation* d'une fonction f consiste à remplacer une partie du corps de f par un appel à une fonction qui est ajoutée aux paramètres de f.

2. Reprendre la fonction précédente et la généraliser en une fonction `listFilterGen`, prenant en paramètre une liste et un prédicat sur les éléments de cette liste, et ne conservant que les éléments vérifiants ce prédicat.

Pour rappel, une *spécialisation* d'une fonction f sur un de ses paramètres p consiste à retirer p des paramètres de f et le remplacer dans le corps de f par une valeur donnée.

3. Spécialiser `listFilterGen` pour écrire une fonction `listFilterAlpha` prenant en paramètres une liste de chaînes de caractères, et ne conservant que celles commençant par une majuscule.

Exercice 2: Fonctions et dérivées

Dans cet exercice, l'idée est de jouer avec la formule suivante qui,à partir d'une fonction F, fournit une approximation de sa dérivée :

$$F'(x) = \frac{F(x+h) - F(x-h)}{2h} + O(h)$$

1. Écrire une fonction `differentiate2` qui prenne 2 paramètres `(h, f)` et applique la formule ci-dessus. Quel est la nature de l'objet renvoyé par cette fonction ?
   Spécialiser cette fonction.
2. Écrire une fonction `differentiate3` qui prenne 3 paramètres `(h, f, x)` et applique la formule.
   Spécialiser cette fonction.
3. Écrire une fonction `differentiateCurry` qui prenne les paramètres un par un, dans l'ordre `(h)` ⇒ `(f)` ⇒ `(x)`, et applique la formule.
   Spécialiser cette fonction de manière à ce que :
   - elle prenne une fonction et renvoie sa fonction dérivée approximée avec `h=0.1`.
   - elle prenne une valeur h et renvoie la fonction dérivée de `Math.cos` approximée avec `h`.
4. (*Bonus*) Écrire une fonction `differentiateN` permettant d'obtenir les dérivées n-ièmes d'une fonction en appliquant plusieurs fois la fonction précédente
   Remarque : pour simplifier, cette fonction ne doit pas être nécessairement curryfiée.

Exercice 3: Calculs de points fixes

La récurrence suivante utilise la méthode de Newton pour approximer la racine carrée d'un nombre x, en construisant une suite (a_n) :

$$\begin{cases} a_0 &= 1 \\ a_n &= \frac{1}{2}\left(a_{n-1} + \frac{x}{a_{n-1}}\right) \end{cases}$$

Pour calculer la racine cubique de x, il existe un schéma de facture équivalente :

$$\begin{cases} a_0 &= 1 \\ a_n &= \frac{1}{3}\left(2a_{n-1} + \frac{x}{a_{n-1}^2}\right) \end{cases}$$

Ces deux algorithmes sont des exemples de *méthodes de point fixe*. Ils sont identifiables par les éléments suivants :

— une valeur de départ `start`,

— une fonction `getNext` qui calcule l'élément suivant de la suite à partir du précédent,

— une fonction `hasReached` qui, étant donné un élément de la suite, et un paramètre `epsilon` fixant la précision du résultat, annonce si l'algorithme est terminé ou pas.

Les méthodes de point fixe peuvent alors se décrire par le pseudo-algorithme impératif suivant :

```
let current = start;
while (!hasReached(current, x, epsilon))
    current = getNext(current, x);
return current;
```

1. Écrire une fonction `fixedPoint` qui implémente l'algorithme de point fixe indiqué ci-dessus de manière récursive.
   Cette fonction devra prendre en paramètre les 5 éléments suivants : `x`, `start`, `epsilon`, `hasReached` et `getNext`.

2. Écrire les fonctions `getNextSquare` et `hasReachedSquare` pour le calcul de la racine carrée.
3. Écrire les fonctions `getNextCubic` et `hasReachedCubic` pour le calcul de la racine cubique.
4. En quoi le travail réalisé ici est une généralisation ?