L3 MIASHS Info

1 Élimination de la concaténation

Soit la fonction `truc` définie de la façon suivante pour `n` strictement positif :

```
truc :: Int -> [Int]
truc 1 = [1]
truc n = truc (n - 1) ++ [n]
```

Proposer une version sans concaténation (en utilisant la méthode vue en cours). Prendre un exemple d'appel et faire la trace dans les deux cas.

2 Ensembles

Pour toutes les questions qui suivent, vous n'avez pas le droit d'utiliser les fonctions prédéfinies de Haskell, sauf les fonctionnelles vues en cours..

On rappelle qu'un ensemble est une collection d'éléments *de même type* dans laquelle un même élément ne peut apparaître *qu'une seule fois*. Les ensembles ne sont pas ordonnés : $2, 3, 4$ est égal à $3, 4, 2$. Un ensemble peut être vide, auquel cas il ne contient aucun élément.

On veut représenter le type abstrait `Ensemble` en Haskell à l'aide d'une liste (comme on l'a vu en cours pour les types `Pile` et `File`) :

1. Préciser les contraintes Haskell que doivent respecter le type des éléments d'un ensemble (en expliquant pourquoi, évidemment).
2. Écrire les fonctions suivantes. Il sera également tenu compte de l'optimisation. Si vous butez sur une fonction et que vous en avez besoin pour définir une autre fonction, contentez-vous de donner sa signature et supposez qu'elle est écrite.
   (a) `card`, `ajouter`, `appartient`, `oter` qui, respectivement, renvoie le nombre d'éléments de l'ensemble, ajoute un élément, teste si un élément appartient à un ensemble, et ôte un élément de l'ensemble. Pour `oter`, vous fournirez trois versions : une avec les listes en intension, une avec une fonctionnelle et une version purement récursive.
   (b) `inclus`, `égal` qui testent, respectivement, si un ensemble est inclus dans un autre ou égal à un autre (deux ensembles sont égaux si l'un est inclus dans l'autre et réciproquement). Par définition, l'ensemble vide est inclus dans tout ensemble.
   (c) `intersection`, `union`, `différence` qui renvoient, respectivement, l'intersection, l'union et la différence de deux ensembles ($A - B$ est l'ensemble des éléments de A qui n'appartiennent pas aussi à B). Pour `intersection`, vous fournirez trois versions : une avec les listes en intension, une avec une fonctionnelle et une version purement récursive.

(d) Écrire les fonctions `liste2Ens` et `Ens2Liste` permettant, respectivement, de produire un ensemble à partir d'une liste et une liste à partir d'un ensemble.

3. Question subsidiaire (pouvant rapporter quelques points supplémentaires) : proposer une implémentation plus abstraite du type `Ensemble` en écrivant notamment les fonctions `appartient` et `intersection`.