TD (feuille 4) : Ordre supérieur et itérateurs

Rappel : Un relation binaire $\mathcal{R}$ sur un ensemble E est une *relation d'ordre* si elle est :
- réflexive ($\forall x \in E.\ x\mathcal{R}x$)
- transitive ($\forall x, y, z \in E.\ x\mathcal{R}y$ et $y\mathcal{R}z$ implique $x\mathcal{R}z$)
- antisymétrique ($\forall x, y \in E.\ x\mathcal{R}y$ et $y\mathcal{R}x$ implique $x = y$)

Par exemple, sur les entiers naturels, les relations "inférieur ou égal" et "est multiple de" sont des relations d'ordre.

Exercice 1 : Ordre lexicographique

L'ordre lexicographique est la relation d'ordre qui permet de trier les mots dans un dictionnaire. Sur les listes, l'ordre lexicographique est défini de la façon suivante :

$$[x_1\ ;\ \ldots\ ;x_n] \leq [y_1\ ;\ \ldots\ ;y_m] \text{ ssi } \begin{cases} \exists i.x_i \leq y_i \wedge \forall j < i.x_j = y_j \\ \text{ ou} \\ \forall j \leq n.x_j = y_j \wedge n \leq m \end{cases}$$

1. Écrire la fonction `lexico: 'a list -> 'a list -> bool` telle que `lexico l1 l2` détermine si une liste `l1` est plus petite qu'une liste `l2` selon l'ordre lexicographique.
2. Écrire la fonction `compare_lexico : ('a -> 'a -> int) -> 'a list -> 'a list -> int` qui compare deux listes selon l'ordre lexicographique (et une relation d'ordre sur les éléments).

Exercice 2 : Ordre multi-ensemble

Un multi-ensemble est un ensemble dans lequel des éléments peuvent être répétés. Ainsi, un multi-ensemble m construit à partir d'éléments d'un ensemble E peut être défini comme une fonction qui à chaque élément de E retourne le nombre de fois qu'il apparaît dans m. On note $m(x)$ le nombre d'occurrences de x dans m (ainsi $m(x) = 0$ quand x n'apparaît pas dans m). L'ordre multi-ensemble est défini de la façon suivante :

$$m_1 \leq m_2 \text{ ssi } \begin{cases} m_1 = m_2 \text{ ou} \\ \forall x \in E.m_2(x) < m_1(x) \Rightarrow \exists y \in E.x < y \wedge m_1(y) < m_2(y) \end{cases}$$

Les listes étant une implantation des multi-ensembles, on peut définir l'ordre multi-ensemble sur les listes de la façon précédente.

1. Écrire, en utilisant un itérateur, une fonction `occurrence : 'a -> 'a list -> int` qui compte le nombre d'occurrences d'un élément dans une liste.

2. En utilisant un itérateur, écrire la fonction `supprime : 'a -> 'a list -> 'a list` qui supprime toutes les occurrences d'un élément dans une liste. Cette fonction ne doit pas nécessairement préserver l'ordre de la liste.
3. En utilisant les fonctions précédentes, écrire la fonction `multiset_of_list : 'a list -> ('a * int) list` qui à partir d'une liste crée le multi-ensemble correspondant où à chaque élément différent de la liste, on lui associe son nombre d'occurrences dans la liste d'origine. Par exemple, `multiset_of_list [1; 6; 3; 3; 1; 3]` retourne la liste `[(1,2); (6, 1); (3; 3)]`.
4. Donner une autre méthode pour écrire cette fonction.
5. En utilisant un itérateur, écrire la fonction `find: 'a -> ('a * int) list -> int` qui prend en arguments un élément x et un multi-ensemble m et qui retourne $m(x)$.
6. En utilisant la fonction précédente, écrire la fonction `sup : ('a -> 'a -> int) -> ('a * int) list -> ('a * int) list -> 'a option` qui prend en arguments une relation d'ordre et deux multi-ensembles m_1 et m_2 et retourne, s'il existe, le plus grand élément de m_1 tel que $m1 > m2$.
7. Écrire la fonction `multiset : ('a -> 'a -> int) -> 'a list -> 'a list -> int` qui implémente une relation d'ordre multi-ensemble (selon une relation d'ordre sur les éléments).

Exercice 2 : Tri par sélection

1. Écrire une fonction `selection_min : ('a -> 'a -> int) -> 'a list -> 'a option * 'a list` qui sépare le plus petit élément (selon un relation d'ordre passée en argument) d'une liste du reste de la liste. Cette fonction ne doit pas nécessairement préserver l'ordre de la liste.
2. À partir de la fonction précédente, définir une fonction `tri_selection : ('a -> 'a -> int) -> 'a list -> 'a list` qui trie une liste selon un relation d'ordre passée en argument.

Exercice 3 : Classement aux jeux olympiques

À la fin des jeux olympiques on peut réaliser un classement final pour déterminer le pays qui a obtenu le plus grand nombre de médailles. Les résultats de chaque pays sont représentés par un couple `(p,m)` où `p` est le nom du pays et `m` est le multi-ensemble des médailles obtenues par ce pays. Le type des résultats est donc `string * medaille list` où `medaille` est le type défini par `type medaille = Or | Argent | Bronze`.

Le classement final prévilégie les médailles d'or (puis les médailles d'argent et de bronze). En cas d'égalité, on utilise un classement lexicographique sur les noms du pays.

1. En utilisant la fonction de tri définie dans la section précédente, écrire une fonction `classement` de type `('a -> 'a -> int) -> ('b * 'a list) list -> ('b * 'a list) list` pour classer la liste des résultats de jeux olympiques.