TP n°9 - Types avancés et arbres en Ocaml

Notions abordées

- Définition de nouveaux types, types somme et type produit
- Type pour les arbres binaires étiquetés
- Arbres binaires partiellement ordonnés
- Parcours d'arbres binaires (préfixe, infixe, postfixe, par niveau)
- Parcours par niveau d'un arbre d'arité quelconque

Toutes les fonctions doivent être commentées et **testées**. Elles doivent notamment être munies d'une description faisant suite à d'éventuelles hypothèses sur leurs arguments.
En OCaml, les commentaires s'écrivent comme suit `(*Commentaires*)`.

Types somme et produit

Exercice 1 Jeu de Scrabble

On cherche dans cet exercice à modéliser un jeu de Scrabble, du moins les pièces et le décompte des points en tenant compte de lettres déjà posées et de cases spéciales. On ne cherche pas à modéliser le plateau de jeu dans cet exercice.

Question 1 💻
Définir un type `piece` qui représente une pièce du jeu, c'est-à-dire un jeton blanc, qui sert de joker, ou bien un jeton avec une lettre de l'alphabet et le nombre de points correspondant. *Les jetons blancs ne rapportent pas de point mais remplacent n'importe quelle lettre.*

Question 2 💻
Définir un type `effet` qui représente les effets que peuvent avoir les cases du plateau, à savoir aucun effet, doubler les points de la lettre, doubler les points du mots, tripler les points de la lettre, ou tripler les points du mot.

Question 3 💻
Définir un type `case` qui représente l'état d'une case au cours du jeu, sachant qu'une case peut être vide, auquel cas son effet est visible, ou bien recouverte par une lettre, auquel cas son effet n'est plus visible, ce qui n'est pas gênant car on ne peut plus en profiter puisque l'effet sur les points ne vaut que pour le mot qui a recouvert la case la première fois.

Question 4 💻
Définir une fonction `points_lettre` qui prend en argument une pièce et un effet et qui calcule le nombre de points que rapporterait cette lettre si elle était posée sur une case ayant cet effet. On néglige à ce stade les effets sur les mots.

Question 5 💻
Définir une fonction `bout_de_mot` qui prend en argument une pièce et qui calcule la chaîne de caractères à un caractère correspondant à la lettre de la pièce, ou à `"_"` pour un blanc. *La fonction* `string_of_char` *n'existant pas, je vous invite à consulter la librairie du module String de Ocaml pour trouver une fonction qui ferait l'équivalent à l'adresse* `https://ocaml.org/releases/4.12/api/String.html` . Une fonction nommée `f` d'un module `m`, s'appelle par `m.f`..

Question 6 💻
Définir une fonction `mot_complet` qui prend en argument une liste de pièces `lp` et une liste de cases `lc` et qui calcule le mot construit en plaçant les pièces de `lp` sur les cases vides de `lc`, et en prenant en compte les pièces des cases pleines jouxtant les pièces de `lp` qu'on vient de poser.
Par exemple, si on souhaite poser les lettres `i`, `n`, et `o` sur une portion du plateau constituée

- d'une case vide,
- d'une case vide,
- d'une case occupée par la lettre `f`
- d'une case vide,
- d'une case vide,

on obtient le mot `info`, car les cases vides à la fin sont omises, tandis que sur sur une portion du plateau constituée

- d'une case occupée par la lettre `m`
- d'une case vide,
- d'une case vide,
- d'une case vide,
- d'une case occupée par la lettre `r`
- d'une case occupée par la lettre `e`
- d'une case vide,
- d'une case occupée par la lettre `r`

on obtient le mot `minore`, le dernier ne jouxtant pas les lettres posées, il n'est pas pris en compte.

Question 7 ✍
Définir une fonction `points_mot` qui prend en argument une liste de pièces `lp` et une liste de cases `lc` et qui calcule le nombre de points que rapporte le mot construit en plaçant les pièces de `lp` sur les cases vides de `lc`, en prenant en compte les pièces des cases pleines jouxtant les pièces de `lp` qu'on vient de poser.

Types récursifs pour des arbres

Exercice 2 Arbre binaire de recherche - partie 1

Dans cet exercice on s'intéresse à des arbres binaires dont le nœuds sont étiquetés par des valeurs d'une type quelconque munies d'une relation d'ordre. Un tel arbre est dit **partiellement ordonné** si toutes les valeurs des nœuds d'un sous-arbre gauche sont inférieures ou égales à celle de la racine, et toutes les valeurs des nœuds d'un sous-arbre droit sont supérieures ou égales à celle de la racine. Pour représenter de tels arbres on se donne les types paramétrés mutuellement récursifs suivants.

```
type 'a nd = { rac:'a ; fg:'a abr ; fd:'a abr }
and 'a abr = | Vide  | Nd of 'a nd
```

Question 1 ✍

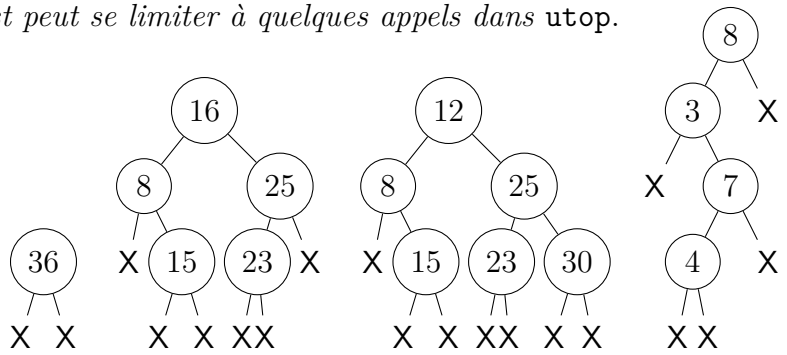
Pour ces deux types, préciser s'il s'agit d'un type somme ou d'un type produit, et combien il a de constructeurs ou de champs. Rappeler les éléments de syntaxe du cours pour ces deux genres de type.

Question 2 🖳

Définir une fonction `feuille` qui crée un arbre réduit à une feuille à partir de la valeur à stocker dans sa racine. *Pour cette fonction le test peut se limiter à quelques appels dans* `utop`.

Question 3 🖳

Définir les arbres représentés ci-contre. Ils pourront être utiles pour les jeux de tests par la suite. *Attention ils ne sont pas nécessairement tous partiellement ordonnés.*



Question 4 🖳

Définir une fonction `est_feuille` qui teste si un arbre est réduit à une feuille.

Question 5 🖳

Définir une fonction `racine` qui calcule la valeur stockée à la racine d'un arbre non vide.

Question 6 🖳

Définir une fonction `tous_inf` (resp. `tous_sup`) qui prend en argument un arbre partiellement ordonné et une valeur (du type des étiquettes) et qui teste si tous les nœuds de l'arbre ont une étiquette inférieure ou égale (resp. supérieure ou égale) à celle passée en argument.

Question 7 🖳

Définir une fonction `est_valide` qui prend en argument un arbre et qui teste s'il est partiellement ordonné. *On ne cherche pas ici à faire une fonction récursive terminale, il s'agit juste d'une fonction "outil" utile en phase de développement pour les tests.*

Question 8 🖳

Définir une fonction `est_présent` qui prend en argument un arbre partiellement ordonné et une valeur du type des étiquettes, et qui teste si l'un des nœuds de l'arbre a cette valeur pour étiquette.

Question 9 🖳

Définir une fonction `minimum` (resp. `maximum`) qui prend en argument un arbre partiellement ordonné et qui calcule la plus grande (resp. la plus petite) étiquette de cet arbre.

Exercice 3 Parcours d'arbres binaires

Dans cet exercice, on considère un type un peu simplifié par rapport à l'exercice précédent pour représenter les arbres binaires puisqu'on remplace le type produit des nœuds par un simple triplet.

```
type 'a ab =
  | Vide
  | Nd of 'a* ('a ab) * ('a ab)
```

Intuitivement, un parcours d'un arbre est une manière de représenter l'arbre à plat, par une liste des étiquettes de ses nœuds. On perd donc la structure arborescente. Formellement, en notant $\mathcal{N}$ l'ensemble des nœuds d'un arbre, la séquence $(e_i)_{i\in[1..n]}$ est un **parcours** d'un arbre A s'il existe une bijection φ de $\mathcal{N}(A)$ dans $[1..n]$ telle que pour tout $i \in [1..n]$, e_i est l'étiquette du nœud $\varphi^{-1}(i)$.

Si de plus pour tout sous-arbre de la forme (r, a_G, a_D) on a

→ $\varphi(r) \leqslant \min\limits_{n\in\mathcal{N}(a_G)} \varphi(n) \leqslant \max\limits_{n\in\mathcal{N}(a_G)} \varphi(n) \leqslant \min\limits_{n\in\mathcal{N}(a_D)} \varphi(n)$, alors ce parcours est dit **préfixe**

→ $\max\limits_{n\in\mathcal{N}(a_G)} \varphi(n) \leqslant \varphi(r) \leqslant \min\limits_{n\in\mathcal{N}(a_D)} \varphi(n)$, alors ce parcours est dit **infixe**

→ $\max\limits_{n\in\mathcal{N}(a_G)} \varphi(n) \leqslant \min\limits_{n\in\mathcal{N}(a_D)} \varphi(n) \leqslant \max\limits_{n\in\mathcal{N}(a_D)} \varphi(n) \leqslant \varphi(r)$, alors ce parcours est dit **postfixe**

Question 1 ✍
Donner une définition informelle, en français, des trois types de parcours précédemment définis.

Question 2 ✍
Donner les 3 parcours des arbres représentés à la question 3 de l'exercice précédent

Question 3 ✍
Où est placée la racine d'un arbre dans son parcours préfixe ? Dans son parcours postfixe ? Quel est le premier élément du parcours infixe d'un arbre ? Pour son parcours préfixe ? pour son parcours postfixe? Et le dernier ?

Question 4 💻
Définir une fonction `affiche_prefixe` qui prend en argument un arbre binaire et une fonction capable de transformer des valeurs d'étiquette de cet arbre en chaîne de caractères, et qui affiche les étiquettes de l'arbre en suivant un parcours préfixe de cet arbre. *Dans cette question, on n'attend pas une fonction récursive terminale. De plus pour le jeu de tests, on n'utilisera pas* `assert`*, mais on proposera un affichage qu'il est facile de vérifier d'un coup d'œil.*

Question 5 💻
Même question avec un parcours infixe.

Question 6 💻
Même question avec un parcours postfixe.

Question 7 💻
En utilisant une liste d'arbre pour enregistrer les arbres restant à traiter, en l'occurrence à afficher, proposer une version récursive de la fonction `affiche_prefixe`.

Question 8 💻
En remarquant en outre qu'afficher la racine d'un arbre étiquetée par un élément `e` est équivalent à afficher, pour n'importe quel parcours, les nœuds de l'arbre réduit à une feuille étiquetée par `e` proposer une version récursive des fonctions `affiche_infixe` et `affiche_postfixe` .

Question 9 💻
Définir une fonction récursive terminale `affiche_par_niveaux` qui prend en argument un arbre binaire, ainsi qu'une fonction transformant une étiquette en chaîne de caractères, qui affiche les étiquettes de tous les nœuds de l'arbre, niveau par niveau à partir de la racine, c'est-à-dire qui affiche tous les nœuds d'une profondeur donnée, de gauche à droite, avant d'afficher les nœuds plus profonds.

Exercice 4 Parcours par niveau d'arbres d'arité quelconque

Question 1 💻
Définir un type paramétré récursif nommé `arbre` pour représenter les arbres étiquetés d'arité quelconque.

Question 2 💻
Définir un arbre nommé `t0` réduit à une feuille d'étiquette `r`, un arbre nommé `t1`, dont la racine est étiquetée par `r` et a trois enfants d'étiquettes `a`, `b`, et `c`, et enfin un arbre nommé `t2` qui est un prolongement de `t1` dans lequel `a` a deux fils d'étiquettes `A` et `B`, et `c` en a trois d'étiquettes `X`, `Y` et `Z`.

Question 3 💻
Définir une fonction récursive terminale `affiche_par_niveaux` qui prend en argument un arbre d'arité quelconque, ainsi qu'une fonction transformant une étiquette en chaîne de caractères, et qui affiche les étiquettes de tous les nœuds de l'arbre, niveau par niveau.

Exercice 5 Transformation d'un arbre en arbre binaire

Dans cet exercice on utilise les types d'arbres des deux exercices précédents. Proposer une fonction permettant de transformer un arbre d'arité quelconque en arbre binaire de sorte que les enfants d'un nœud dans l'arbre initial, sont ses descendants dans l'arbre obtenu.