

TD (feuille 5) : Arbres

Objectifs Définitions et manipulation d'arbres

Exercice 1 - Une autre représentation des arbres

En cours, vous avez vu la définition d'arbres binaires polymorphes suivante :

```
type 'a arbre = Vide | Noeud of 'a * 'a arbre * 'a arbre
```

1. Définir le type `('a, 'b) arbre` des arbres binaires polymorphes où une information de type `'b` est associée aux feuilles.
2. Définir les fonctions `taille: ('a, 'b) arbre -> int`, `profondeur: ('a, 'b) arbre -> int` et `est_complet: ('a, 'b) arbre -> bool`.
3. Définir la fonction `nb_feuilles: ('a, 'b) arbre -> int` qui compte le nombre de feuilles d'un arbre.
4. Définir la fonction `feuilles: ('a, 'b) arbre -> 'b list` qui retourne la liste des feuilles d'un arbre.
5. Définir la fonction `map n f a` qui applique la fonction `n` aux noeuds de `a` et la fonction `f` à ses feuilles.

```
val map : ('a -> 'b) -> ('c -> 'd) -> ('a, 'c) arbre -> ('b, 'd) arbre
```

Exercice 2 - Un peu de logique

On veut représenter des expressions booléennes contenant les opérateurs binaires \wedge (et), \vee (ou), \Rightarrow (implique) ainsi que l'opérateur unaire \neg (non), les constantes *vrai* et *faux* ainsi que les variables propositionnelles identifiées par un caractère (*a*, *b*, etc.).

1. Donner la représentation sous forme d'arbre des expressions suivantes :
 - $\neg(a \Rightarrow b)$
 - $(a \vee b) \wedge c$
 - $\neg(a \vee b) \Rightarrow \neg a \wedge \neg b$
2. Définir le type `expr_bool` des expressions booléennes.
3. Donner les valeurs OCaml correspondant aux expressions précédentes.
4. Spécifier et réaliser une fonction `enlever_implique: expr_bool -> expr_bool` qui renvoie un arbre ne contenant pas l'opérateur implique équivalent à celui qu'elle reçoit en paramètre.
5. Définir une fonction `simplifier: expr_bool -> expr_bool` qui effectue les simplifications suivantes :

- $\neg\neg a \rightarrow a$
- $vrai \wedge a \rightarrow a$ et $a \wedge vrai \rightarrow a$
- $faux \wedge a \rightarrow faux$ et $a \wedge faux \rightarrow faux$
- $vrai \vee a \rightarrow vrai$ et $a \vee vrai \rightarrow vrai$
- $faux \vee a \rightarrow a$ et $a \vee faux \rightarrow a$

6. Définir la fonction `eval env e` qui évalue l'expression `e` dans un environnement `env` qui à chaque constante identifiée par un caractère associe une valeur booléenne.

`val eval: (char * bool) list -> expr_bool -> bool`

7. Une formule est un forme normale disjonctive (FND) si elle est composée de disjonctions de conjonctions (par exemple, $(a \wedge b) \vee (a \wedge \neg c)$ est en FND). Définir un type `expr_fnd` plus adapté à la représentation des formules en forme normale disjonctive.

8. Définir la fonction `fnd: expr_bool -> expr_fnd` qui met une formule en forme normale disjonctive.