

PROGRAMMATION FONCTIONNELLE 4

FILTRAGE PAR MOTIFS

Mise en jambes

Exercice 1. Que fait la fonction suivante ?

```
let f l =
  let rec aux acc = function
    | [] -> acc
    | h::t -> aux (h::acc) t
  in
  aux acc []
```

Notez que cette utilisation de sous-fonctions auxiliaires est très commune sous OCaml et, plus généralement, en programmation fonctionnelle.

Si l est une liste de longueur n , combien de fois sera appelée l'opération de concaténation (c'est-à-dire l'opération $::$) ?

Exercice 2. Écrivez une fonction qui cherche le plus petit élément d'une liste – et renvoie `None` si la liste est vide. Vous aurez besoin d'employer le filtrage par motifs – et probablement de définir une fonction auxiliaire, comme dans l'exercice 1.

Exemple :

```
# minimum [1;2;0;-5;7;8];;
- : int option = Some (-5)
# minimum [];;
- : int option = None
```

Manipulation formelle

Pour ce qui suit, nous emploierons le type `formule_ensembliste` des formules ensemblistes, défini par :

```
type formule_ensembliste = Nom          of string
                          | Union        of formule_ensembliste
                                      * formule_ensembliste
                          | Intersection of formule_ensembliste
                                      * formule_ensembliste
                          | Complement  of formule_ensembliste
```

Ainsi, on écrira `Intersection (Nom "A", Complement (Nom "A"))` pour représenter $\overline{A \cup B \cup C}$, ou encore `Intersection (Nom "A", Complement (Nom "A"))` pour représenter $A \cap \overline{A}$. Bien entendu, à ce niveau, OCaml n'a aucune idée de ce que signifient les intersections ou les unions.

Normalisation de De Morgan

Rappelons les lois de De Morgan :

$$\overline{A \cup B} = \overline{A} \cap \overline{B} \text{ et } \overline{A \cap B} = \overline{A} \cup \overline{B}.$$

Exercice 3. Écrivez une fonction récursive `descendre_complements` qui parcourra une formule ensembliste `f` et la réécrira de manière à faire *descendre les compléments*, c'est-à-dire réécrire les formules du style $\overline{A \cup B}$ sous la forme $\overline{A} \cap \overline{B}$ et, de même, $\overline{A \cap B}$ sous la forme $\overline{A} \cup \overline{B}$. Si cette fonction n'a rien trouvé à réécrire, par exemple parce qu'il n'y avait pas de compléments ou parce que les compléments étaient tout en bas, on renverra `None`. Dans le cas contraire, on renverra `Some f`, où `f` est la formule ensembliste après modification.

Pour le moment, on se contentera d'un seul passage à travers la formule.

Exemple :

```
# descendre_complements (Complement (Union (Union (Nom "A", Nom "B"), Nom
"C"))));
- : formule_ensembliste option = Some (Intersection (Complement (Union (Nom
"A", Nom "B")), Complement (Nom "C")))
# descendre_complements (Intersection (Complement (Union (Nom "A", Nom "B")),
Complement (Nom "C"))));
- : formule_ensembliste option = Some (Intersection (Intersection (Complement
(Nom "A"), Complement (Nom "B")), Complement (Nom "C")))
# descendre_complements (Intersection (Intersection (Complement (Nom "A"),
Complement (Nom "B")), Complement (Nom "C"))));
- : formule_ensembliste = None
```

Exercice 4. Écrivez une fonction `descendre_complements_au_maximum` qui, à l'aide de `descendre_complements`, fait descendre le plus profondément possible les négations dans la formule. Cette fonction ne s'arrêtera que lorsqu'il n'est plus possible de faire descendre les négations.

Exemple :

```
# descendre_complements (Complement (Union (Union (Nom "A", Nom "B"), Nom
"C"))));
- : formule_ensembliste = Intersection (Intersection (Complement (Nom "A"),
Complement (Nom "B")), Complement (Nom "C"))
```

Exercice 5. Écrivez une fonction `string_from_formule_ensembliste` qui transforme une formule ensembliste en une chaîne de caractères un peu plus lisible qui représentera cette formule.

C'est un exercice difficile. Faites de votre mieux.

Exemple :

```
# string_from_formule_ensembliste (Nom "B");;
- : string = "B"
# string_from_formule_ensembliste (Union (Nom "A", Nom "B"));;
- : string = "(A union B)"
# string_from_formule_ensembliste
```

Exercice 6. * Écrivez une fonction qui calcule le nombre d'apparitions d'un nom d'ensemble donné dans une formule.

```
# apparitions_de "A" (Complement (Union (Union (Nom "A", Nom "B"), Nom "C"))));
- : int = 1
```

Exercice 7. * Écrivez une fonction qui détermine tous les noms d'ensembles apparaissant dans une formule.

```
# noms_dans (Complement (Union (Union (Nom "A", Nom "B"), Nom "C"))));
- : string list = ["A"; "B"; "C"]
```

Comment se comporte votre fonction lorsqu'un même nom apparaît plusieurs fois ?