TD 3 : Inférence de types

1 Types inductifs, types polymorphes

Le type

```
type 'a option =
   None
 | Some of 'a
```

permet de représenter le fait qu'un élément est "présent" ou "existant" (par exemple `Some 42`) ou qu'il n'existe pas (`None`).

1.1 Listes d'association

Une liste d'association est une liste de paires (clé, valeur). Par exemple, la liste `[("Max", 10); ("Nicolas", 4); ("Nicole", 9)]` établit une association entre le nom et l'age d'une personne.

1. Écrivez une fonction `lookup_assoc` qui prend une clé et une liste d'association et renvoie `Some` (valeur associée) si la clé apparaît dans la liste, et `None` sinon.
   Exemple :
   `lookup_assoc "Nicolas" [("Max", 10); ("Nicolas", 4); ("Nicole", 9)]` donne `Some 4`,
   `lookup_assoc "Maurice" [("Max", 10); ("Nicolas", 4); ("Nicole", 9)]` donne `None`
2. Écrivez une fonction `add_assoc` qui prend une clé, une valeur et une liste d'association, et ajoute la paire (clé, valeur) à la liste.
3. Écrivez une fonction `remove_assoc` qui prend une clé et une liste d'association et supprime la (les) valeur(s) associée(s).

Il existe plusieurs manières d'implanter ces fonctions. Discutez les avantages et inconvénients. Vos fonctions doivent avoir le type le plus général possible, c.à.d., pas être uniquement applicables aux listes d'associations `string * int`.

Toutes les implantations doivent satisfaire certains invariants, par exemple : `lookup_assoc k (add_assoc k v kvs) = Some v`. Trouvez d'autres invariants de vos fonctions ! Testez vos fonctions pour vérifier qu'ils sont satisfaits.

1.2 Maps

Un map est une fonction qui associe une valeur (option) à une clé :

```
type ('k, 'v) map = 'k -> 'v option
```

Les maps peuvent être considérés comme des implantations spécifiques d'associations.

1. Écrivez les fonctions `lookup_map, add_map, remove_map` qui correspondent aux fonctions de l'exercice précédent.
2. Écrivez une fonction `empty_map`, qui renvoie `None` pour tout argument.
3. Écrivez une fonction `assoc2map` qui convertit une liste d'association en map et qui s'appuie sur `empty_map` et `add_map`. Testez vos fonctions, par exemple :

   ```
   # lookup_map "Nicole"
     (assoc2map [("Max", 10); ("Nicolas", 4); ("Nicole", 9)]) ;;
   - : int option = Some 9
   ```

2 Implantation de types en Caml

1. Proposez un type de donnés en Caml qui représente les types d'un langage avec polymorphisme, comme vu en cours.
2. Écrivez une fonction qui effectue une substitution dans un type.
3. Écrivez une fonction qui prend deux types T_1 et T_2 et détermine si T_2 est une instance de T_1.
   (a) Dans un premier temps, vous pouvez supposer que les variables de type de T_1 et T_2 sont disjointes et que toutes les occurrences de variables dans T_1 sont différentes.
   (b) Modifiez la fonction pour pouvoir traiter le cas où T_1 contient plusieurs occurrences de la même variable.
   (c) Quelles sont les problèmes qui se posent si T_1 et T_2 contiennent les mêmes variables ? (On vous demande uniquement d'identifier les problèmes, sans les résoudre ici).

3 Unification de types

Donnez les unificateurs des types suivants (`'a, 'b, 'c` sont des variables de type) :
— `int -> int` et `'a -> 'b`
— `(int -> int) -> 'a` et `int -> (int -> 'b)`
— `(int -> bool) -> 'a` et `'a -> 'b`

4 Inférence de types

Inférez le type des expressions suivantes :
— `fun x -> f (f x)`, pourvu que `f : int -> int`
— `fun p -> fun x -> (p 3) && (p x)` (avec le typage habituel de `&&`)
— `fun f -> f (f 42)`
— `fun f -> f (f true)`
— `fun g -> List.map g [[1];[2];[3]]`, où
`List.map : ('a -> 'b) -> 'a list -> 'b list`