TD : définition de types

Définitions de types

Exercice 1. Montrer qu'il existe une bijection entre les deux ensembles suivants :

$$A \times B \times C$$

$$\{ f : \{1,2,3\} \longrightarrow A + B + C \mid f(1) \in A, f(2) \in B, f(3) \in C \}$$

Généraliser au cas d'un produit Cartésien $A_1 \times \ldots \times A_n$.
Pour un ensemble de fini d'étiquettes N, choisissons un ensemble A_n pour tout $n \in N$ et considérons l'ensemble

$$\{ f : N \longrightarrow \sum_{n \in N} A_n \mid \forall n \in N \ f(n) \in A_n \}.$$

Comment montrer que cet ensemble est un produit Cartésiens ? Quel rapport avec les records de Caml ?

Listes

Exercice 2. Définir un nouveau type `'a liste` ayant la forme d'un record contenant un liste conventionnelle, un fonction pour comparer les élément de la liste, un booléen pour dire si cette liste est trié.

Exercice 3. Écrire une fonction `insert : 'a -> 'a liste -> 'a liste` qui ajoute un élément de la liste.

Exercice 4. Écrire une fonction `sort : 'a liste -> 'a liste` qui fait le tri par insertion d'une liste donnée.

Arbres lexicaux

On se sert des arbres lexicaux pour représenter les dictionnaires.

```
# type lex_noeud = Letter of char * bool * lex_arbre
  and
  lex_arbre = lex_noeud list;
# type mot = char list;
```

Exercice 5. Écrire une fonction `exists : mot -> lex_arbre -> bool` pour tester si un mot appartient au dictionnaire.

Exercice 6. Écrire une fonction `insert : mot -> lex_arbre -> lex_arbre` pour ajouter un mot dans un dictionnaire. Écrire une fonction `construire : mot list -> lex_arbre` qui construit le dictionnaire à partir de une liste de mots.

Ensembles et graphes

Exercice 7. Proposer 3 définition d'un type `ens` pour représenter les ensembles. Écrire une liste d'opérations sur les ensembles qu'on souhaiterai implémenter. Écrire le « prototype »de chaque fonction.

Exercice 8. Proposer une définition de type pour les graphes. Écrire une liste d'opérations sur les graphes qu'on souhaiterai implémenter. Écrire le « prototype »de chaque fonction.

Évaluation par valeur (et par nom)

Exercice 9. Expliquer ce qu'il se passe pendant la session suivante :

```
# let x = 3 + 5;;
# let rec fonction = function
   [] -> x
 | (tete::queue) -> x + tete + (fonction queue);;
# let x = 2;;
# fonction [x;3;x+x;5];;
```

Exercice 10. Que se passe-t'il si dans un langage fonctionnel dont la stratégie d'évaluation est par valeur, on ne possède ni des constructeurs comme `if ... then ... else ...` ?

Exercice 11. Définissons

```
#let sialorssinon x y z = if x then y else z
```

Pour quelle raison une expression

```
sialorssinon expr1 expr2 expr3
```

n'est pas équivalent à

```
if expr1 then expr2 else expr3
```

? Proposer des expressions `expr1`, `expr2` `expr3` qui montrent que les deux expressions ne sont pas équivalentes.

Un peu de code

Exercice 12. Expliquer ce que fait le code suivant :

terms.ml

```
 1 : type 'a signature = ('a  * int) list ;;
 2 :
 3 : let arity sign x =
 4 :   try
 5 :     List.assoc x sign
 6 :   with
 7 :       _ -> failwith "arity"  ;;
 8 :
 9 : type variable = string ;;
10 :
11 : type ('a,'b) term =
12 :     Var of 'b
13 :   | Term of 'a * ('a,'b) term list ;;
14 :
15 :
16 : let rec term_trav f gluefun gluestart varfun =
17 :   function (Term(oper,sons)) ->
18 :     let
19 :       recurrencelist
20 :       = List.map (term_trav f gluefun gluestart varfun) sons
21 :     in
22 :       f(oper, List.fold_left gluefun gluestart recurrencelist)
23 :     | (Var n) -> varfun n ;;
24 :
25 : let ok_sig sign term =
26 :   let revcons l a = a::l in
27 :     try term_trav
28 :       ( fun (fsymb,list)  ->
29 :    ( List.for_all (fun x -> x=true) list
30 :      && arity sign fsymb = List.length list) )
31 :       revcons [] (fun x -> true) term
32 :     with Failure "arity" -> false ;;
```