TD : itération, exceptions

Définitions de types (encore)

Listes

Exercice 1. 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 2. Écrire une fonction `insert : 'a -> 'a liste -> 'a liste` qui ajoute un élément de la liste.

Exercice 3. Écrire une fonction `sort : 'a liste -> 'a liste` qui fait le tri par insertion d'une liste donnée.

Ensembles et graphes

Exercice 4. 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 5. 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.

Exceptions

Exercice 6. Est ce que le code suivant (incomplet) vous rappelle quelque-chose ?

algox.ml

```
 1 : type variable = Var of string;;
 2 : type literal = Positive of variable | Negative of variable;;
 3 : type clause = literal list;;
 4 : type ens_clause = clause list;;
 5 :
 6 : let rec algox (enscl : ens_clause) =
 7 :   if est_vide enscl then Some(empty)
 8 :   else
 9 :     if contient_faux enscl then None
10 :     else
11 :       let
12 :         appliquer1 regle_unaire ens siechecquoifaire =
13 :                ...
14 :       and
15 :         appliquer2 regle_binaire ens siechecquoifaire =
16 :                ...
17 :       in
18 :         appliquer1 regle1 enscl
19 :           (
20 :             appliquer1 regle2 enscl
21 :              (
22 :                appliquer1 regle3 enscl
23 :                 (
24 :                   appliquer1 regle4 enscl
25 :                    (
26 :                      appliquer2 regle5 enscl None
27 :                    )
28 :                 )
29 :              )
30 :           )
31 : ;;
```

1. On se sert du mécanisme des exceptions si une règle `reglei` n'est pas applicable à un ensemble `ens` : elle lèvera l'exception `Failure "regle"`. Écrire, comme premier exemple, le code de la `regle2`.
2. Compléter la définition (récursive) de `algox` avec les définitions de `applquer1` et `appliquer2` en utilisant `try ... with`.

3. Faire une liste des symboles non défini dans ce code, et en donner les types.
4. ... compléter le code.

Termes, itération

Considérer le code suivant :

terms.ml

```
 1 : type ('a,'b) term =
 2 :     Var of 'b
 3 :   | Operation of 'a * ('a,'b) term list ;;
 4 :
 5 : let rec term_iter funvar funop funcoller premier = function
 6 :     (Var x) -> funvar x
 7 :   | (Operation (oper,fils)) ->
 8 :       let
 9 :         recur term = term_iter funvar funop funcoller premier term
10 :       in
11 :       let
12 :         coller = fun x term -> funcoller x (recur term)
13 :       in
14 :         funop oper (List.fold_left coller premier fils);;
15 :
16 : let algox1 term =
17 :   let
18 :     funvar v = [v]  and funop op lv = lv
19 :   and
20 :     funcoller = (@) and premier = []
21 :   in
22 :     term_iter funvar funop funcoller premier term;;
23 :
24 : type 'a signature = ('a  * int) list ;;
25 : let arity (sign : 'a signature) op =
26 :   try
27 :     List.assoc op sign
28 :   with
29 :       _ -> failwith "arity"  ;;
30 :
31 : let algox2 sign term =
32 :   let
33 :     funvar v = true
34 :   and
35 :     funop op (n,b) =
36 :     try
37 :       b && (n = (arity sign op))
38 :     with
39 :         Failure "arity" -> false
40 :   and
41 :     funcoller (n,b1) b2 = (n+1, (b1 && b2)) and premier = (0,true)
42 :   in
43 :     term_iter funvar funop funcoller premier term;;
```

Exercice 7. Dire ce qui est calculé par les fonctions `algox1` et `algox2`.

Exercice 8. Utiliser la fonction `term_iter` pour définir une fonction `string_of_term` dont le comportement est (par exemple) :

```
# let t1 = Operation("g",[
                      Operation("f",[Var "x"]);
                      Var("y")
                    ]);;
      val t1 : (string, string) term =
  Operation ("g", [Operation ("f", [Var "x"]); Var "y"])
# string_of_term t1;;
- : string = "g(f(x),y)"
```