Institut Informatique Claude Chappe

Master Informatique
Unité 177UD05
Programmation fonctionnelle

TP n°2

Objectifs du TP :

- Fonctions récursives
- Fonctions de manipulation de liste
- Type de données algébriques

Exercice n°1 : fonctions récursives

a) Ecrivez la fonction *product* qui calcul le produit des nombres d'une liste.

b) Ecrivez la fonction *length* qui calcul la longueur d'une liste.

c) Ecrivez la fonction *init* qui renvoie une liste sans le dernier élément.

d) Ecrivez l'opérateur *(++)* de concaténation de deux listes en une seule liste.

e) Ecrivez une fonction *insert* d'insertion d'un élément dans une liste déjà ordonnée.

f) Ecrivez la fonction *isort* de tri d'une liste par insertion. Utilisez *insert*.

Exercice n°2 : encore des fonctions récursives

a) Ecrivez les fonctions de test *paire* et *impair* qui s'appellent l'une et l'autre. On considère que 0 est paire.

b) Ecrivez la fonction *drop* qui supprime les n premiers éléments d'une liste.

c) Ecrivez la fonction *take* qui garde les n premiers éléments d'une liste.

d) Ecrivez la fonction *halve* qui coupe une liste en deux et retourne un tuple d'arité 2. Utilisez *take* et *drop*.

e) Ecrivez la fonction *merge* qui fusionne 2 listes triées en une seule liste triée.

f) Ecrivez la fonction *msort* de tri par fusion. On utilisera *merge* et *halve*.

Exercice n°3 : évaluateur de tautologies

On va écrire un évaluateur de tautologie. Une tautologie est une expression logique toujours vraie. Considérons un langage de proposition construit sur des valeurs basiques (*True*, *False*) et des variables (*A, B, ... , Z*) utilisant la négation (*Not*) , la conjonction (*And*), l'implication (*Imply*) et les parenthèses.

On utilisera le type de données suivant :

```
data Prop    = Const Bool
             | Var Char
             | Not Prop
             | And Prop Prop
             | Imply Prop Prop
```

On définit ainsi les quatre propositions suivantes :

```
p1 :: Prop
p1 = And (Var 'A') (Not (Var 'A'))

p2 :: Prop
p2 = Imply (And (Var 'A') (Var 'B')) (Var 'A')

p3 :: Prop
p3 = Imply (Var 'A') (And (Var 'A') (Var 'B'))

p4 :: Prop
p4 = Imply (And (Var 'A') (Imply (Var 'A') (Var 'B'))) (Var 'B')
```

L'implication est définie de la manière suivante :

p	q	Imply p q
False	False	True
False	True	True
True	False	False
True	True	True

Enfin, pour évaluer une proposition logique nous devons connaître la valeur de chaque variable. Pour cela nous définissons les deux types suivants :

```
type Assoc k v = [(k, v)]
type Subst = Assoc Char Bool
```

Par exemple, pour évaluer la proposition *p1*, il faudra utiliser une des 2 tables de substitution suivantes : *[('A', True)]* ou *[('A', False)]*. Pour évaluer la proposition *p2*, il faudra utiliser une des 4 tables de substitution suivantes : *[('A', True), ('B', True)]* ou *[('A', True), ('B', False)]* ou *[('A', False), ('B', False)]* ou *[('A', False), ('B', True)]*.

a) Ecrivez la fonction *find :: Eq k => k -> Assoc k v -> v.* Vous pouvez utiliser une liste par compréhension ou définir récursivement la fonction.

Exemples : *find 'B' [('A', True), ('B', False)] == False*
find 'A' [('A', True), ('B', False)] == True

b) Ecrivez la fonction *eval :: Subst -> Prop -> Bool* qui évalue une proposition donnée selon une liste de substitution donnée.
Rappel : *False < True* car le type *Bool* est défini ainsi.

Exemples : eval *[('A', True), ('B', False)] p2 == True*
eval [('A', True)] p1 == False

Pour décider si une proposition est une tautologie, il faut tester pour toutes les substitutions possibles des variables de la proposition. Dans la suite, vous allez définir plusieurs fonctions dont le but sera de générer toutes les tables de substitutions possibles pour une proposition donnée.

c) Commencez par écrire la fonction *vars2 :: Prop -> [Char]* qui génère une liste des variables d'une proposition. Pour supprimer les variables qui apparaissent plusieurs fois vous utiliserez la fonction suivante :

```
rmdups :: Eq a => [a] -> [a]
rmdups [] = []
rmdups (x:xs) = x : filter (/= x) (rmdups xs)

vars :: Prop -> [Char]
vars p = rmdups (vars2 p)
```

Exemple : *rmdups [1,2,3,3,2] == [1,2,3]*
rmdups "ABCBBCA" == "ABC"
vars p1 == "A"
vars p2 == "AB"

d) Ecrivez maintenant la fonction *bools :: Int -> [[Bool]]* qui génère la table de vérité pour n variables.

Exemple : *bools 3 =>*
```
[[False, False, False],
 [False, False, True],
 [False, True, False],
 [False, True, True],
 [True, False, False],
 [True, False, True],
 [True, True, False],
 [True, True, True]]
```

On peut observer que *bools 3* contient deux fois le résultat de *bools 2*. Ceci permet de définir récursivement la fonction *bools* : dans le cas de *bools 0* on renvoie une liste contenant une liste vide, dans le cas de *bools n* on prend deux copies des listes produit par *bools (n-1)*, on place *False* dans la première liste, on place *True* dans la seconde liste puis on concatène le tout.

Complétez le code suivant :

```
bools :: Int -> [[Bool]]
bools 0 = ...
bools n = map (False :) bss ++ map (True :) bss
   where bss = ...
```

e) Il reste maintenant à « injecter » dans la table de vérité générer par *bools* les variables trouvées par *vars* pour obtenir les tables de substitution. On utilise la fonction *zip :: [a] -> [b] -> [(a,b)]* qui prend deux listes et renvoie une liste de couples.

Complétez le code suivant :

```
substs :: Prop -> [Subst]
substs p = map (zip vs) (bools (length vs))
   where vs = ...
```

f) Ecrivez la fonction *isTaut :: Prop -> Bool* qui teste si une proposition est une tautologie. On utilisera la fonction *and :: [Bool] -> Bool*.

Exemples : isTaut p1 == False
isTaut p3 == False
isTaut p2 == True
isTaut p4 == True

Références web

Site officiel Haskell
https://www.haskell.org

Site officiel des modules Haskell
https://downloads.haskell.org/~ghc/latest/docs/html/libraries/