Université Paris 7 - DEUG-MIAS Année 2004-2005

TP de *Logique et Circuits* n° 5

Encore des listes et des expressions

Exercice 1 *L'exercice 4 du TD.*
On souhaite écrire une fonction `ordre` de type `'a list -> 'a list -> bool` décrivant l'ordre dictionnaire entre listes. On adopte la convention que la liste vide est plus petite que toute autre suite.

1. Écrire une fonction `ordre_0` telle que `ordre_0 l1 l2` renvoie `true` si le premier élément de la liste `l1` est inférieur ou égal à celui de la liste `l2`, et `false` sinon. Si l'une des deux listes est vide, la fonction renvoie une erreur.
2. Écrire une fonction récursive `ordre` telle que `ordre l1 l2` renvoie `true` si la liste `l1` est inférieure ou égale à la liste `l2` et `false` sinon.
3. Tester.

Exercice 2 *Polynômes.*
Un polynôme $P(X) = a_nX^n + \cdots + a_2X^2 + a_1X + a_0$ peut être vu comme la liste $[a_0; a_1; \ldots; a_n]$ de ses coefficients. Les opérations sur les polynômes peuvent donc être vues comme des opérations sur des listes de nombres.

1. Quel type doit-on utiliser pour représenter des polynômes à coefficients entiers ?
2. Définir une fonction `monôme` telle que `monôme a n` soit la représentation du polynôme aX^n.
3. Définir une fonction `somme` telle que, si `p` et `q` sont les coefficients de $P(X)$ et $Q(X)$, l'expression `somme p q` renvoie la liste des coefficients de $P(X) + Q(X)$.
4. Définir une fonction `scalaire` telle que si `p` représente $P(X)$, `scalaire n p` renvoie les coefficients de $n \times P(X)$.
5. Définir une fonction `produit` telle que, si `p` et `q` sont les coefficients de $P(X)$ et $Q(X)$, `produit p q` renvoie la liste des coefficients du produit $P(X) \times Q(X)$.
6. Définir une fonction `égal_poly` qui teste l'égalité de deux polynômes. Attention : X et $X + 0X^2$ sont égaux.

Exercice 3 *Un peu de calcul formel.*
Une autre façon de représenter des polynômes est la suivante : l'ensemble des polynômes à coefficients entiers d'une variable X est la clôture par somme et produit de l'ensemble $\{X\} \cup \mathbb{N}$.

1. Définir un type `'a expr` pour représenter des polynômes à une variable X et à coefficients dans `'a`.
2. Définir une fonction `expr_of_poly` qui prend un polynôme (décrit comme une liste de coefficients, comme dans l'exercice 2) et renvoie une expression qui représente ce polynôme.
3. À l'aide des fonctions de l'exercice 2, définir une fonction `poly_of_expr` qui prend une expression et renvoie la liste des coefficients du polynôme qu'elle représente.
4. En déduire une fonction `égal_expr` qui teste si deux expressions représentent le même polynôme.

5. Pour une liste `p`, que renvoie `poly_of_expr (expr_of_poly p)` ?
6. Pour une expression `e`, que renvoie `expr_of_poly (poly_of_expr e)` ?

Exercice 4 *Une remarque sur l'égalité en OCaml.*
Il y a deux tests d'égalité en OCaml : le "`=`" et le "`==`". Le "`=`" teste l'égalité structurelle alors que le "`==`" teste l'égalité physique, c'est-à-dire la représentation en mémoire par OCaml.

1. Définir `type arbre = Feuille | Noeud of arbre * arbre`
2. Évaluer
   (a)
   ```
   let a = Feuille in
   let b = Feuille in
   (a = b), (a == b);;
   ```
   (b)
   ```
   let a = Feuille in
   let b = a in
   (a = b), (a == b);;
   ```
   (c)
   ```
   let a = Noeud(Feuille, Feuille) in
   let b = Noeud(Feuille, Feuille) in
   (a = b), (a == b);;
   ```
   (d)
   ```
   let a = Noeud(Feuille, Feuille) in
   let b = a in
   (a = b), (a == b);;
   ```
   (e)
   ```
   ("aa" = "aa"), ("aa" == "aa");;
   ```
3. Quel est le résultat de chaque évaluation ? Cela est-il raisonnable avec le préambule ? Quelle construction utilisera-t-on en général ?