Structure de données récursives

12 octobre 2002

1 les listes

Une structure de données récursives est une structure de données qui dans sa définition fait appelle à elle-même. Nous avons déjà rencontré ce type de donnée avec la définition des listes. Nous pouvons redéfinir nous même le type liste de la manière suivante :

```
#type 'a liste =
  Vide |
 suite of 'a * 'a liste;;
Le type liste est défini.
```

Le mot ÇsuiteÈ est devenu un mot clé qui va nous permettre de construire des listes :

```
#let a = Vide;;
a : 'a liste = Vide
#let b = suite(5,suite(6,Vide));;
b : int liste = suite (5, suite (6, Vide))
```

On va pouvoir aussi faire de la reconnaissance de motifs sur le type `liste` que nous venons de construire. Voyons en quelques exemples en construisant l'équivalent des fonctions `hd`, `tl` et `list_length`.

```
#exception ListeVide;;
L'exception ListeVide est définie.
#let tete = function
 Vide -> raise ListeVide
| suite(x,xs) -> x;;
tete : 'a liste -> 'a = <fun>
#let queue = function
  Vide -> raise ListeVide
| suite(x,xs) -> xs;;
queue : 'a liste -> 'a liste = <fun>
#let rec longueur = function
 Vide -> 0
| suite(x,xs) -> 1 + (longueur xs);;
longueur : 'a liste -> int = <fun>
```

2 Les arbres

2.1 Les arbres binaires simples

Les arbres binaires se rencontrent fréquemment en informatique. Nous avons vu que les expressions algébriques peuvent facilement être représentées sous forme arborescente. Commençons par voir comment nous pouvons déclarer des arbres binaires simples définies de la manière suivante, un arbre est :

soit, une simple feuille

soit, un nœud composé de 2 arbres

Ceci nous amène tout naturellement à la déclaration suivante:

```
1  #type arbre =
2  feuille
   | noeud of arbre * arbre;;
4  Le type arbre est défini.
```

Définissons le plus simple des arbres :

```
1  #let a = feuille;;
2  a : arbre = feuille
```

Les fonctions de base pour les arbres sont

`nb_feuilles` qui calcule le nombre de feuilles d'un arbre,

`hauteur` qui calcule la hauteur (on dit aussi la profondeur) d'un arbre,

`largeur` qui calcule la largeur d'un arbre

```
1  #let rec nb_feuilles = function
2    feuille -> 1
   | noeud(fg,fd) ->  (nb_feuilles fg) + (nb_feuilles fd);;
4  nb_feuilles : arbre -> int = <fun>
   #let rec hauteur = function
6    feuille -> 0
   | noeud(fg,fd) -> max (hauteur fg) (hauteur fd);;
8  hauteur : arbre -> int = <fun>
   #let rec largeur = function
10   feuille -> 0
   | noeud(fg,fd) ->  (largeur fg) + (largeur fd);;
12 largeur : arbre -> int = <fun>
```

2.2 Les arbres binaires moins simples

On ne peut pas faire grand chose des arbres simples à par tester une structure récursive. Un arbre devient intéressant dès que l'on peut placer des objets au niveau de ces feuilles (et/ou) de ses nœuds. De la même façon que pour les listes ou les tableaux, les éléments placés au niveau des nœuds doivent être tous du même type. Il en est de même pour les éléments placés au niveau des feuilles. Commençons par voir comment déclarer un arbre homogène, c'est à dire tel que les feuilles et les nœud soient de même type.

```
1  #type 'a arbre =
```

```
2   feuille of 'a
    | noeud of 'a arbre * 'a arbre;;
4   Le type arbre est défini.
```

Il est souvent plus utile de construire des arbres pour lesquels les feuilles ne sont d'aucune utilité mais pour lesquels on place un élément à chaque nœud. Ce type d'arbre est utilisé dans le cadre des arbres binaires de recherche qui seront plus longuement utilisé en deuxième année.

```
1   type 'a arbre =
2   feuille
    | noeud of 'a arbre * 'a * 'a arbre;;
```

2.3 Les arbres binaires complexes

Dans le cadre de l'utilisation des arbres binaires pour la représentation des expressions algébriques, on place des opérateurs binaires aux niveaux des nœuds et les opérandes aux niveaux des feuilles. Il faut faire appel à des arbres un peu plus complexes, avec des arbres dit hétérogènes.

```
1   #type ('a,'b) arbre =
2   feuille of 'b
    | noeud of ('a,'b) arbre * 'a * ('a,'b)  arbre;;
4   Le type arbre est défini.
```

Je vous laisse deviner comment déclarer des arbres autorisant un nombre quelconque de sous-arbres au niveau de chaque nœud.
exercice : Réécrire les fonctions `hauteur`, `nb_feuilles`, `largeur` pour les 3 types d'arbres définis précédemment.