Les listes

7 mai 2003

Les listes en Caml

Il existe une liste dans laquelle il n'y a aucun élément appelée liste vide et notée ( ). À partir de la liste vide, on définit de manière inductive les listes d'éléments d'un ensemble E. si a est un élément de E et l une liste d'éléments de E alors (a, l) est encore une liste d'éléments de E. De manière ensembliste on construit de manière inductive les ensembles E_n de la manière suivante :

- $E_0 = \{(\,)\}, E_0$ contient seulement la liste vide,
- $E_{n+1} = E \times E_n$

L'ensemble des listes d'éléments de E est alors égal à $\bigcup_{n\in\mathbb{N}} E_n$. Les listes d'éléments de E sont donc de la forme $(a_1, (a_2, (\ldots, (a_n, ()) \ldots)))$. Les listes sont représentées de manière simplifiée en Caml sous la forme $[a_1; a_2; \ldots; a_n]$ et la liste vide sous la forme [ ].

Il faut bien garder à l'esprit la représentation symbolique des listes. Pour $l' = (a, l)$, on voit naturellement apparaître deux fonctions de base :

- $l' = (a, l) \mapsto a$
- $l' = (a, l) \mapsto l$

La première fonction est définie de $\bigcup_{n\in\mathbb{N}^*} E_n$ à valeurs dans E, c'est la fonction qui à la liste $l' = (a, l)$ renvoie la tête de liste a. En caml, elle se note `hd` (`hd` étant la contraction de head = tête). Ainsi `hd [`$a_1; a_2; \ldots; a_n$`]` est égal à a_1.

```
1  #hd [2;4;5;1;2;3;4;6];;
2  - : int = 2
```

La fonction `hd` n'est évidemment pas définie sur la liste vide :

```
1  #hd [];;
2  Exception non rattrapée: Failure "hd"
```

La seconde fonction est définie de $\bigcup_{n\in\mathbb{N}^*} E_n$ à valeurs dans $\bigcup_{n\in\mathbb{N}} E_n$, c'est la fonction qui à la liste $l' = (a, l)$ renvoie la queue de liste l. En caml, elle se note `tl` (`tl` étant la contraction de tail = queue). Ainsi `tl [`$a_1; a_2; \ldots; a_n$`]` est égal à $[a_2; \ldots; a_n]$.

```
1  #tl [2;4;5;1;2;3;4;6];;
2  - : int list = [4; 5; 1; 2; 3; 4; 6]
```

La fonction `tl` n'est évidemment pas définie sur la liste vide :

```
1  #tl [];;
2  Exception non rattrapée: Failure "tl"
```

Il faut bien retenir le fait que dans une liste, on ne voit que la tête et la queue, contrairement au cas des tableaux dans lesquels chaque élément à la même facilité d'accès. Pour accéder au dernier élément d'une liste, il faut «voir» tous ceux qui le précèdent. Tout se passe comme dans le cas d'une pile d'assiettes, où pour constituer une pile d'assiettes, on les empile les unes sur les autres et pour prendre l'assiette en bas de pile, il faut enlever toutes les assiettes au dessus.

Pour construire des listes, nous devons disposer d'une fonction définie sur $E \times \bigcup_{n\in\mathbb{N}} E_n$ à la valeur dans $\bigcup_{n\in\mathbb{N}^*} E_n$ qui pour à partir d'un élément a et d'une liste l construit la liste (a, l). En Caml, cette fonction est **::** et se présente sous forme d'un opérateur.

```
1   #let l = 21.5::[];;
2   l : float list = [21.5]
    #let g = 2.8::l;;
4   g : float list = [2.8; 21.5]
```

Pour traiter les listes, nous disposons en Caml d'une reconnaissance de motif.

```
1   #let type_liste = function
2      [] -> print_string "c'est la liste vide"
    | [a] -> print_string "cette liste n'a qu'un seul élément"
4   | a::b::q -> print_string "cette liste contient au moins deux éléments";;
    type_liste : 'a list -> unit = <fun>
```

Illustrons son utilisation en créant une fonction d'affichage des listes d'entiers :

```
1   #let affiche l =
2   let rec aff = function
      [] -> print_string "]"
4    | [x] -> print_int x; aff []
     | x::xs -> print_int x; print_string ";"; aff xs
6   in
      print_string "[";
8     aff l;;
    affiche : int list -> unit = <fun>
10  #affiche [1;6;5;4;7;3;2;-5];;
    [1;6;5;4;7;3;2;-5]- : unit = ()
```

exercices :

- Écrire une fonction qui calcule la somme des termes d'une liste d'entiers,
- Écrire une fonction qui calcule le produit des termes d'une liste de réels,
- Écrire une fonction qui à partir d'une liste, retourne la liste des termes d'ordre pair,
- Écrire une fonction qui à partir d'une liste retourne le couple formé par la liste des termes d'ordre pair et la liste des termes d'ordre impair.