1 Tableaux

1.1 Généralités

Mathématiquement, un tableau est une suite finie d'objets de même type (des entiers, des mots, d'autres tableaux, etc), indexée traditionnellement de 0 à $n-1$: n est appelé la longueur du tableau. Si t est un tableau de longueur n et i un indice compris entre 0 et $n-1$, alors on note habituellement $t[i]$ le terme d'indice i du tableau t.

Informatiquement, un tableau t de longueur n est représenté par une suite consécutives de cases-mémoires de même taille, déterminée par le type des objets de t. On peut voir un tableau comme une famille de variables indexées.

Si on note `t` le symbole informatique représentant le tableau mathématique t, `t` est en fait la position en mémoire de la première case du tableau (on dit aussi une référence ou un pointeur) : il est alors facile de calculer la position de la case contenant $t[i]$

$$\text{position de } \texttt{t[i]} = \texttt{t} + \texttt{i} \times \text{taille d'une case}$$

Au prix donc d'une multiplication et d'une addition entière, on peut donc accéder directement au contenu de la case-mémoire `t[i]` : on dit qu'un tableau est une structure à accès direct (ou aléatoire).

Un tableau étant déterminé dès sa création (longueur et type des données), il n'est possible ni de réduire, ni d'augmenter sa longueur en cours de traitement : c'est une structure statique. La seule façon de simuler la modification de la longueur d'un tableau consiste donc à créer un nouveau tableau et à y recopier les valeurs des cases : c'est donc un processus coûteux en temps.

Néanmoins, on peut accéder à la fois en lecture et en écriture à toute case du tableau. Le contenu de chaque case est modifiable à tout moment, on dit qu'un tableau est une structure mutable.

Un tableau est donc une structure de données particulièrement indiquée quand le nombre de données ne varie pas et est connu à l'avance, et quand on doit effectuer de multiples phases de lecture / écriture. Une mémoire réelle d'un ordinateur (mémoire vive RAM ou disque dur) est typiquement modélisable par un tableau.

1.2 Implémentation CAML

En langage CAML, on définit un tableau de deux façons.

- Soit concrètement en donnant la liste des éléments du tableau : `[| 1; 2; 5 ; -8; 7 |]` est un tableau de longueur 5, `[||]` est un tableau vide de longueur 0.
- Soit en définissant son modèle : `make_vect 12 8` est un tableau de longueur 12 dont toutes les cases contiennent l'entier 8.

Le type d'un tableau est `'a vect`.

Si `t` est un tableau, sa longueur est donnée par la fonction `vect_length t`.

La valeur de la case numéro i du tableau est notée `t.(i)`

Enfin, on peut modifier la valeur de cette case par l'opérateur `<-` : `t.(i) <- x`

Remarque. Attention ! La modification des valeurs d'un tableau ne fait pas partie du modèle de la programmation fonctionnelle, mais plutôt de la programmation impérative.

Trois erreurs courantes rencontrées quand on manipule des tableaux :

- les tableaux sont homogènes (un seul type d'objet à la fois dans les cases), donc attention aux erreurs de typage ;
- l'exception `vect_item` est levée lorsqu'on demande l'accès en lecture à une case inexistante (indice hors limites) ;
- l'exception `vect_assign` est levée lorsqu'on demande l'accès en écriture à une case inexistante.

2 Listes

2.1 Définition mathématique

Soit E un ensemble et Nil un objet n'appartenant pas à E.
On pose $E_0 = \{\text{Nil}\}$ et pour tout $n \in \mathbb{N}$, $E_{n+1} = E \times E_n$.
Habituellement, on identifie les triplets (a, b, c) avec les couples $(a, (b, c))$. Mais ici on s'interdit ce genre de raccourcis.
Enfin, on note $L(E) = \bigcup_{n\in\mathbb{N}} E_n$: une liste d'éléments de E est un élément de $L(E)$.
Une liste est donc un objet du type :

$$\ell = (a_0, (a_1, (a_2, ..., (a_{n-1}, \text{Nil}))...)$$

La liste à un seul objet Nil est appelée liste vide et est de longueur 0 (elle n'a aucun élément de E). Sinon la liste ℓ ci-dessus est dite de longueur n et a n éléments (sous-entendu « de E »).
La définition mathématique formalise l'idée intuitive que dans une liste, on ne peut accéder directement qu'à son premier élément : les autres éléments ne sont accessibles qu'en parcourant la liste. De même, pour construire une liste, on ne peut le faire qu'à partir d'une liste existante : comme la seule liste existante naturellement est la liste vide, tout doit partir de cette liste.
Formellement, on définit deux applications tête et queue sur $L(E) - \{\text{Nil}\}$, la première à valeurs dans E, la seconde dans $L(E)$:

$$T(\ell) = a_0 \text{ et } Q(\ell) = (a_1, (a_2, ..., (a_{n-1}, \text{Nil}))...).$$

La tête d'une liste est donc son premier élément et sa queue est elle-même privée de son premier élément.
On définit de même une application constructeur, définie sur $E \times L(E)$:

$$c_a(\ell) = Cons(a, \ell) = (a, \ell)$$

2.2 Représentation informatique

À l'inverse des tableaux, une liste est concrètement une suite de couples de cases-mémoires, couples qui peuvent être non consécutifs (on appelle cette structure une liste chaînée) : la première case du couple contient la donnée et la seconde l'adresse de l'élément suivant dans la liste. Cette structure est aisément modifiable : ajouter un élément en tête, supprimer la tête se font en temps constant, mais l'accès à une case quelconque nécessite de parcourir toutes les cases précédentes.
Selon le type de problème, l'usage de tableaux ou de listes doit être réfléchi en termes d'avantages/inconvénients.

	Tableaux	Listes
Accès en lecture/écriture	en complexité constante	en complexité linéaire
Longueur	fixe et estimée avant tout	variable et adaptable
Surestimation de la longueur	gaspillage mémoire	sans objet : selon les besoins
Sous-estimation de la longueur	recopie nécessaire	idem
Algorithmes	impératifs	récursifs

2.3 Implémentation CAML

a) Notation

En langage CAML, on note `[a1; a2; ...; an]` la liste $(a_1, (...(a_n, \text{Nil})...)$.
Les fonctions tête et queue sont notées `hd` et `tl`.
Enfin le constructeur c_a est noté `a ::`
Si `L` est une liste d'objets (de même type) et `a` un objet, la notation `a :: L` représente la liste `L` à laquelle on a ajouté l'élément `a` en tête.

Exemples

a) la liste vide est notée `[]`

b) les listes `[1; 3; 4]`, `1 :: 3 :: 4 :: []`, `1 :: [3; 4]` sont identiques

La reconnaissance de motif se fait aussi par l'intermédiaire de cette notation.

b) Fonctions sur les listes

Les listes sont naturellement utilisées pour écrire des algorithmes récursifs. Pour cela, on ne dispose que des fonctions hd et tl et des tests ou reconnaissances de motifs ! Avec ça on peut construire des fonctions plus complexes, dont certaines ont été prédéfinies en Caml (list_length , etc — voir aide-mémoire).

Fonction longueur d'une liste écrite de deux façons :

```
let rec longueur liste =
  if liste = [] then 0
  else 1 + longueur (tl liste);;

let rec longueur liste =
  match liste with
  | [] -> 0
  | _ :: q -> 1 + longueur q;;
```

Fonction maximum des termes d'une liste non vide :

```
let rec maximum liste =
  if tl liste = [] then hd liste
  else let a = hd liste and q = tl liste in
       let m = maximum q in
       max a m;;

let rec maximum liste =
  match liste with
  | [a] -> a
  | a :: q -> let m = maximum q in
              max a m;;
```

3 Filtrage en CAML

La fonction **match** ... **with** ... est une généralisation de la conditionnelle **if** ... **then** ... **else** ... , qui fonctionne selon le principe exposé ci-dessous.

```
match objet with
  | motif_1 -> valeur_1
  | motif_2 -> valeur_2
  etc
  | motif_n -> valeur_n
```

On lui présente un objet : CAML va comparer la **forme** de cet objet avec les **formes** des motifs dans l'ordre indiqué ; dès qu'un motif correspond à l'objet, on rend la valeur associée, qui est calculée en fonction du motif et de l'objet.

Les motifs sont constitués de deux façons :

- ils peuvent être des valeurs constantes concrètes : 0, 1, 1.25, "mot", [| 0; 1; 2 |], etc, ou des identificateurs ou du symbole spécial _ (« attrape-tout »).
- ils peuvent être des assemblages de valeurs concrètes ou d'identificateurs, associés grâce à des constructeurs.

Lorsqu'un motif correspond, tous les symboles qu'il contient sont liés aux valeurs reconnues par le motif dans l'objet pendant la durée du calcul de la valeur associée. Conséquence : il ne peut pas y avoir deux symboles identiques dans un motif.

```
match x with
  | false -> 0
  | true -> 1
```

```
match n with
  | 0 -> 0
  | m -> m - 1
```

```
let rec binomial cpl =
match cpl with
  | n, 0 -> 1
  | 0, k -> 0
  | n, k -> binomial (n - 1, k) + binomial (n - 1, k - 1);;
```

```
let rec binomial cpl =
match cpl with
  | n, 0 -> 1
  | n, k -> if k > n then 0
            else if k = n then 1
            else binomial (n - 1, k) + binomial (n - 1, k - 1);;
```

Les constructeurs qui peuvent apparaître dans un motif sont (entre autres) : la virgule (constructeur de couples, de triplets, etc), les constructeurs de liste ::, les constructeurs personnels. Attention : ne sont pas considérés comme des constructeurs les symboles de tableaux [| |] (en Caml-Light, car en Ocaml, ça marche).

Les fonctions suivantes sont incorrectes ou surprenantes :

```
let egal (a,b) =
  match (a,b) with
  | (x, x) -> true
  | _ -> false;;
```

```
let compare x y =
  match x with
  | y -> true
  | _ -> false;;
```