Fiche outil – OCaml

Modules List et Array

I. Module `List`

Créer une liste.

- `List.init n f` crée une liste de taille `n`, où le i-ème élément est `f`(i),
- `x :: q` concatène l'élément `x` avec la liste `l`,
- `l @ p` concatène la liste `l` avec la liste `p`,
- `[a1; a2; a3; ...; an]` crée la liste d'éléments `a1`, `a2`, ..., `an`.

Accéder au i-ème élément.

- `List.nth l i` accède au `i`-ème élément de `l`. Cette opération est en $\mathcal{O}($`i`$)$.

Trouver la taille de la liste.

- `List.length l` donne le nombre d'éléments de la liste `l`.

Itérer sur la liste.

- `List.iter f l` appelle `f` sur chacun des éléments de la liste `l` *sans conserver* le résultat,
- `List.map f l` appelle `f` sur chacun des éléments de la liste `l` *en conservant* le résultat,
- ou de la récursivité (mais ***pas*** de boucle `for`).

Opérations booléennes sur les listes.

- `List.mem x l` teste si l'élément `x` est dans la liste `l` (même plusieurs fois),
- `List.for_all p l` teste si tous les éléments de `l` vérifient[1] le prédicat `p`,
- `List.exists p l` teste si un élément de `l` vérifie[1] le prédicat `p`,
- `List.filter p l` ne conserve que les éléments vérifiant[1] le prédicat `p`.

« Réduire » la liste à un élément.

- `List.fold_left f a0 l` calcule a_n, où $n+1$ est la taille de la liste et, pour tout élément x_i d'indice i, on définit $a_i =$ `f`(a_{i-1}, x_i) et on définit $a_0 =$ `a0`. Ceci correspond à :

$$\texttt{List.fold\_left f a0 l} = \texttt{f}\ (\texttt{f}\ (\ldots\texttt{f}(\texttt{f}(\texttt{a0}, x_0), x_1)\ldots, x_{n-1}), x_n)$$

[1] *i.e.* si `p`$(\texttt{x}) =$ `true` pour tous les éléments `x` du tableau ou de la liste

II. Module `Array`

Créer un tableau.

- `Array.init n f` crée un tableau de taille `n`, où le i-ème élément est `f`(i),
- `Array.make n x` crée un tableau de taille `n`, où tous les elements valent `x`,
- `Array.make_matrix m n x` crée une matrice de taille `m` $\times$ `n`, où tous les elements valent `x`,
- `Array.copy a` copie le tableau `a`,
- `[|a1; a2; a3; ...; an|]` crée le tableau d'éléments `a1`, `a2`, ..., `an`.

Accéder au i-ème élément.

- `a.(i)` accède au `i`-ème élément de `a`. Cette opération est en $\mathcal{O}(1)$,
- `a.(i) <- x` définit le `i`-ème élément de `a`, avec `x`,

Trouver la taille du tableau.

- `Array.length a` donne le nombre d'éléments du tableau `a`.

Itérer sur le tableau.

- `Array.iter f a` appelle `f` sur chacun des éléments de la liste `a` *sans conserver* le résultat,
- `Array.map f a` appelle `f` sur chacun des éléments de la liste `a` *en conservant* le résultat,
- ou une boucle `for`.

Opérations booléennes sur les tableaux.

- `Array.mem x a` teste si l'élément `x` est dans la liste `l` (même plusieurs fois),
- `Array.for_all p a` teste si tous les éléments de `a` vérifient[1] le prédicat `p`,
- `Array.exists p a` teste si un élément de `a` vérifie[1] le prédicat `p`.

« Réduire » le tableau à un élément.

- `Array.fold_left f a0 t` calcule a_n, où $n+1$ est la taille de du tableau et, pour tout élément x_i d'indice i, on définit $a_i =$ `f`(a_{i-1}, x_i) et on définit $a_0 =$ `a0`. Ceci correspond à :

$$\texttt{Array.fold\_left f a0 t} = \texttt{f}\ (\texttt{f}\ (\ldots\texttt{f}(\texttt{f}(\texttt{a0}, x_0), x_1)\ldots, x_{n-1}), x_n)$$

Hugo Salou