Programmazione Funzionale

Esercitazione 10 – Liste associative

Ricordiamo che una lista associativa e una lista di tipo di tipo `('a*'b) list`. Ad esempio `[(2,true) ; (1;false) ; (2;false)]` e `[(false, [13;2]) ; (true, []) , (true, [3;1;0])]` sono liste associative respettivamente di tipo `(int*bool) list` e `(bool * (int list)) list`.

Una lista associativa puo essere pensata come un *dizionario* in C# o Java. Dato una lista associativa `lst: ('a *'b) list` i elementi di tipo `'a` (cioè quelli a sinistra) sono chiamati *chiave* della lista `lst` invece i elementi di tipo `'b` (cioè quelli a destra) sono chiamati *valori*.

Esercizio 1. Dato una lista associativa vogliamo ottenere la sua lista di chiave e la sua lista di valori.

1. Senzare usare `List.split`, definire la funzione `chiave : ('a * 'b) list -> 'a list` che prende una lista associativa e restituisce la lista delle sue chiave.
2. Senzare usare `List.split`, definire la funzione `valori : ('a * 'b) list -> 'b list` che prende una lista associativa e restituisce la lista dei suoi valori.
3. Definire `chiave` e `valori` mediante `List.split`.

Esercizio 2. Definire le funzioni del modulo `List` segente:

1. `List.assoc : 'a -> ('a * 'b list) -> 'b` che prende un element `x` e una lista associativa `lst` e restituisce – se esiste – il primo elemento della forma `(x,y)` che occore in `lst`.
2. `List.mem_assoc : 'a -> ('a * 'b list) -> bool`. che prende un element `x` e una lista associativa `lst` e restituisce `true` se e solo se se un elemento della forma `(x,y)` occore in `lst`.
3. `List.remove_assoc : 'a -> ('a * 'b list) -> bool`. che prende un element `x` e una lista associativa `lst` e restituisce `lst` da cui e stato tolto la prima occorenza di un elemento di chiave `x`, cioè della forma `(x,y)`.

Esercizio 3. Dato una lista associativa vogliamo trovare dato un valore una chiave associata a quel valore.

1. Definire la funzione `findkey: 'b -> ('a * 'b list) -> 'a` che prende un elemento `x` di tipo `'b` e una lista `lst` e restituisce la prima occorenza della forma `(y,x)` in `lst`.
2. Definire `swap : ('a * 'b) -> ('b * 'a)` che prende una coppia `(x,y)` e restituisce `(y,x)`.
3. Mediante `swap` e `List.map` definire una funzione `swapList: ('a * 'b list) -> ('b * 'a) list` che prende una lista associative e inverta le chiave e i valori.
4. Definire `swapList` senza usare `List.map`.
5. Definire `findkey` mediante `swapList` e `List.assoc`.

Esercizio 4. Una lista `lst : 'a list` soddisfa una coppia `(a,n)` fatta di un elemento `a : 'a` e un intero `n : int` se e solo se l'elemento `a` occore *esatamente* `n` volte in `lst`.

- Definire una funzione `soddisfa: ('a * int) -> 'a list -> bool` che prende una lista `lst` un coppia `(a,n) : 'a * int` e restituisce `true` se l'elemento `a` occore *esatamente* `n` volte in `lst` e false altrimenti.
- Definire una funzione `sodfunctions: ('a * int) -> ('a list -> bool) list` che prende una lista `[c1 ; . . . ; cn]` di coppie e restituisce `[soddisfa c1 ; . . . ; soddisfa cn]`.
- Definire una funzione `apply : ('a -> 'b) list -> 'a -> 'b` che prende una lista di funzione `[f1 ; . . . ; fn]` e una elemento `a` e restituisce `fn(. . . f2((f1 a)). . . )`.
- Usando `sodfunctions` e `apply`, definire una funzione `soddis: ('a * int) list -> 'a list -> bool` che prende una lista associativa `alst` e una lista `lst` e restituisce `true` se e solo se `lst` soddisfa tutte le coppie della lista `alst`