Laboratorio di LPO

7 novembre 2019

a.a. 2019/2020

Definire le seguenti funzioni in OCaml fornendo per ognuna di esse una versione che usa parametro di accumulazione e ricorsione di coda e una ottenuta per specializzazione della funzione `List.fold_left`.

1. `cat : string list -> string`
   `cat` l restituisce la stringa ottenuta concatenando tutte le stringhe di l secondo l'ordine stabilito dalla lista.

   Esempio: `cat ["This";"is ";"awesome!"] = "This is awesome!";;`

2. `filter : ('a -> bool) -> 'a list -> 'a list`
   `filter` p l restituisce la lista ottenuta eliminando da l gli elementi che non soddisfano il predicato p.

   Esempio: `filter ((< ) 0) [-1;1;2;-2] = [1;2];;`

Provare a definire le seguenti funzioni in OCaml come specializzazione di `List.fold_left`.

3. `reverse : 'a list -> 'a list`
   `reverse` l restituisce la lista ottenuta rovesciando l.

   Esempio: `reverse [1;2;3] = [3;2;1];;`

4. `map : ('a -> 'b) -> 'a list -> 'b list`
   `map` f l restituisce la lista ottenuta applicando a ogni elemento di l la funzione f.

   Esempio: `map String.length ["apple"; "orange"] = [5;6];;`

(Più difficili) *Definire le seguenti funzioni in OCaml usando parametro di accumulazione e ricorsione di coda.*

5. `init : int -> (int -> 'a) -> 'a list`
   `init` si comporta come la funzione `List.init`; in particolare, solleva l'eccezione `Invalid_argument` se l'argomento è negativo. Esempi:

   ```
   init 0 (fun x->x) = [];;
   init 10 (fun x->x*x) = [0; 1; 4; 9; 16; 25; 36; 49; 64; 81];;
   ```

6. `concat : 'a list list -> 'a list`
   `concat` si comporta come la funzione `List.concat`; può essere implementata con complessità lineare nella lunghezza della lista risultante, senza usare l'operatore `@`.

   Esempi:

   ```
   concat [] = [];;
   concat [[];[]] = [];;
   concat [[1;2];[3];[4;5;6]] = [1;2;3;4;5;6];;
   ```

7. `combine_no_err : 'a list -> 'b list -> ('a * 'b) list`
   `combine_no_err` si comporta come la funzione `List.combine` ma restituisce sempre un risultato, anche quando le due liste hanno lunghezza diversa.

   Esempi:

   ```
   combine_no_err [1;2;3] ["a";"b";"c"]=[(1, "a"); (2, "b"); (3, "c")];;
   combine_no_err [1;2] ["a";"b";"c"]=[(1,"a");(2,"b")];;
   combine_no_err [1;2;3] ["a";"b"]=[(1,"a");(2,"b")];;
   combine_no_err [] ["a";"b";"c"]=[];;
   combine_no_err [1;2;3] []=[];;
   ```

8. `combine : 'a list -> 'b list -> ('a * 'b) list`
   `combine` si comporta come la funzione `List.combine`; in particolare, solleva l'eccezione `Invalid_argument` se le due liste hanno lunghezza diversa.

   Esempi:

   ```
   combine [] []=[];;
   combine [1;2;3] ["a";"b";"c"]=[(1, "a"); (2, "b"); (3, "c")];;
   ```

9. `max_list : 'a list -> 'a option`
   `max_list` restituisce `None` se la lista è vuota, altrimenti `Some` m, con m il massimo della lista; usare la funzione predefinita `max : 'a -> 'a -> 'a` per semplificare il codice.

   Esempi:

   ```
   max_list [] = None;;
   max_list [3;4;6;-1] = Some 6;;
   max_list ["apple";"orange";"banana"] = Some "orange";;
   ```

Definire le seguenti funzioni in OCaml che usano tipi varianti.

10. `area : shape -> float`
    `area` calcola l'area di una figura geometrica; il tipo `shape` è così definito:

    ```
    type shape = Square of float | Circle of float | Rectangle of float * float;;
    ```

11. `eval : exp_ast -> bool`
    `eval` restituisce il valore dell'espressione booleana rappresentata dall'albero della sintassi astratta; il tipo `exp_ast` è così definito:

    ```
    type exp_ast =
      BoolLit of bool | Not of exp_ast | And of exp_ast * exp_ast | Or of exp_ast * exp_ast;;
    ```

    Esempio:

    ```
    let ast = Not(And(BoolLit true,Or(BoolLit false,BoolLit false)));;
    eval ast=not(true && (false || false));;
    ```