Paradigmi di Programmazione - A.A. 2022-23

Esame Scritto del 23/05/2023

CRITERI DI VALUTAZIONE:
La prova è superata se si ottengono almeno 12 punti negli esercizi 1,2,3 e almeno 18 punti complessivamente.

Esercizio 1 [Punti 4]

Applicare la β-riduzione alla seguente λ-espressione fino a raggiungere una espressione non ulteriormente riducibile o ad accorgersi che la derivazione è infinita:

$$(\lambda g.\lambda y.gyx)(\lambda f.fy)(\lambda z.z)$$

Nella soluzione, mostrare tutti i passi di riduzione calcolati, sottolineando ad ogni passo la porzione di espressione a cui si applica la β-riduzione (redex) ed evidenziando le eventuali α-conversioni.

Esercizio 2 [Punti 4]

Indicare il tipo della seguente funzione OCaml, mostrando i passi fatti per inferirlo:

```
let f x y z =
  match (x,y) with
  | (0,[]) -> []
  | (x1,[]) -> x
  | (x1,x2::lis) -> if x2<0 then lis else [];;
```

Esercizio 3 [Punti 7]

Assumendo il seguente tipo di dato che descrive alberi n-ari di interi:

```
type ntree =
  | Node of int * ntree list
```

Definire in OCaml le funzioni `sum` e `flat` con i seguenti tipi

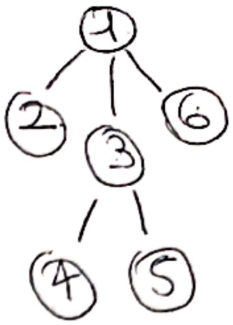
```
sum : ntree -> int
flat : ntree -> int list
```

tali che:

- `sum` prende un albero e restituisce la somma di tutti i valori memorizzati nei suoi nodi
- `flat` esegue una **visita posticipata** dell'albero, ossia restituisce la lista di tutti i valori contenuti nei nodi tenendo conto che, per ogni nodo, la lista prodotta come risultato contiene prima i valori dei figli e poi il valore del nodo stesso.

Ad esempio, considerando il seguente albero:

```
let t = Node (1, [
                 Node (2, []);
                 Node (3, [
                          Node (4,[]);
                          Node (5,[])
                          ]);
                 Node (6, [])
                 ])
```



le funzioni danno i seguenti risultati

```
sum t;;      (* restituisce 21 *) ok
flat t;;     (* restituisce [2;4;5;3;6;1] *) ok
```

NON ho le conoscenze di controllore

Esercizio 4 [Punti 15]

Si estenda il linguaggio MiniCaml visto a lezione con il costrutto `Seq` per la definizione di una struttura dati per memorizzare sequenze di valori non modificabili. Una sequenza viene creata definendone il contenuto e può essere processata solo tramte le seguenti operazioni `merge` e `map`:

- `merge(seq1,seq2)`: date due espressioni `seq1` e `seq2` che descrivono sequenze, restituisce una nuova sequenza ottenuta concatenando le due. Ad esempio (con una sintassi simile a OCaml):

  `merge ( Seq [1;2;3] , Seq [4;5;6] )` restituisce il valore $[1;2;3;4;5;6]$.

- `map(seq,x,body)`: data una espressione che descrive un sequenza `seq`, un identificatore `x` e una espressione `body` che utilizza `x`, restituisce una nuova sequenza ottenuta valutando ripetutamente `body` sostituendo ad `x` ogni elemento di `seq`, uno alla volta nell'ordine in cui sono in seq. Ad esempio (con una sintassi simile a OCaml):

  `map ( Seq [1;2;3] , x , x+1 )` restituisce il valore $[2;3;4]$.

Si mostri come deve essere modificato l'interprete OCaml del linguaggio.

? forse dovevo usare le o merge