Paradigmi di Programmazione - A.A. 2021-22

Esempio di Testo d'Esame n. 1

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:

$$(\lambda x.xxy)(\lambda x.\lambda y.xyy)$$

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.

SOLUZIONE:

$$\begin{array}{rl} & \underline{(\lambda x.xxy)(\lambda x.\lambda y.xyy)} \\ \rightarrow & \underline{(\lambda x.\lambda y.xyy)(\lambda x.\lambda y.xyy)y} \\ \rightarrow & \underline{(\lambda y.(\lambda x.\lambda y.xyy)yy)y} \\ \rightarrow & (\lambda x.\lambda y.xyy)yy \\ \equiv_\alpha & \underline{(\lambda x.\lambda z.xzz)yy} \\ \rightarrow & \underline{(\lambda z.yzz)y} \\ \rightarrow & yyy \end{array}$$

Esercizio 2 [Punti 4]

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

```
let f x y z =
  match x with
  | [ ] -> z
  | w::ws -> w y;;
```

SOLUZIONE:
Struttura del tipo:

```
X -> Y -> Z -> RIS
```

Uso per convenzione `X,Y,Z` come variabili di tipo per i parametri `x,y,z`, `RIS` come variabile di tipo del risultato, e `A,B,C,...` come variabili di tipo "fresche" per la definizione dei vincoli.

Vincoli:

```
X = A list      (da pattern matching)
A = B -> C      (da uso di w)
Y = B           (da w y)
Z = C           (da casi del pattern matching)
RIS = Z         (da primo caso del pattern matching)
RIS = C         (da secondo caso del pattern matching)
```

Ne consegue:

```
X = (B -> C) list
Y = B
Z = RIS = C
```

Tipo inferito:

```
(B -> C) list -> B -> C -> C
```

che in sintassi OCaml corrisponde a:

```
('a -> 'b) list -> 'a -> 'b -> 'b
```

Esercizio 3 [Punti 7]

Definire, usando i costrutti di programmazione funzionale in OCaml, una funzione `duemax` con tipo

`duemax : 'a list -> ('a * 'a) option`

in modo che `duemax lis` restituisca `Some (x,y)` tale che per ogni elemento `z` di `lis` vale $x \geq y \geq z$. La funzione restituisce `None` in caso di lista vuota. Esempi:

```
duemax [2;3;1;5] = Some (5,3)
duemax ['t'] = Some ('t','t')
duemax [] = None
```

SOLUZIONE:

Una possibile soluzione:

```
let rec duemax lis =
  match lis with
  | [] -> None
  | x::[] -> Some (x,x)
  | x::y::[] -> if x>y then Some (x,y) else Some (y,x)
  | x::lis' -> match duemax lis' with
               | None -> failwith "Errore impossibile"
               | Some (a,b) -> if x>a then Some (x,a)
                               else if x>b then Some (a,x)
                               else Some (a,b);;
```

Esercizio 4 [Punti 15]

Estendere il linguaggio MiniCaml visto a lezione con una nuova forma di astrazione funzionale `both-fun`. L'astrazione consente di definire una coppia di funzioni non ricorsive. In sintassi concreta l'astrazione `both-fun(fun x=x+1, fun x=x*2)` applicata al valore `5` produce come risultato la coppia `<6,10>`.
Definire le regole di inferenza operazionali che descrivono la valutazione di `both-fun`, ed estendere consistentemente l'implementazione in OCaml dell'interprete del linguaggio.

SOLUZIONE:
Una possibile soluzione:

$$\Gamma \triangleright BothFun(x, e_1, e_2) \Rightarrow BothClosure(x, e_1, e_2, \Gamma)$$

$$\frac{\Gamma \triangleright e_1 \Rightarrow BothClosure(arg, body_1, body_2, \Gamma_{fDecl}) \qquad \Gamma \triangleright e_2 \Rightarrow aVal \quad \Gamma_{fDecl}[arg = aVal] \triangleright body1 \Rightarrow v_1 \qquad \Gamma_{fDecl}[arg = aVal] \triangleright body2 \Rightarrow v_2}{\Gamma \triangleright Apply(e_1, e_2) \Rightarrow (v_1, v_2)}$$

```
type exp = ...
           | BothFun of ide * exp * exp
           (* assumo che i parametri delle due funzioni abbiano lo stesso nome *)

type evT = ...
           | BothClosure of ide * exp * exp * evT env
           | Pair of evT * evT

let rec eval e s = match e with
                   ...
                   | BothFun (arg,body1,body2) ->
                          BothClosure (arg,body1,body2,s)
                   | Apply (e1,e2) ->
                          let fclosure = eval e1 s in
                          (match fclosure with
                           ...
                           | BothClosure (arg,body1,body2,fDecEnv) ->
                               let aVal = eval e2 s in
                               let aenv = bind fDecEnv arg aVal in
                               Pair (eval body1 aenv, eval body2 aenv)
                          )
```