Paradigmi di Programmazione - A.A. 2021-22

Esame Scritto del 10/06/2022

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]

Eseguire le seguenti sostituzioni come da definizione di capture-avoiding substitution:

- $(\lambda y.\lambda z.(xyz))\{x := \lambda k.k\}$
- $(\lambda y.\lambda z.(xyz))\{x := y\}$
- $(\lambda y.\lambda z.(xyz))\{y := k\}$

SOLUZIONE:

- $\lambda y.\lambda z.((\lambda k.k)yz)$
- $\lambda k.\lambda z.(ykz)$
- $\lambda y.\lambda z.(xyz)$

Esercizio 2 [Punti 4]

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

```
let f x y =
  match x with
  | (0,z) -> y
  | (1,z) -> z
  | (_,z) -> y@z ;;
```

SOLUZIONE:
Struttura del tipo:

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

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

Vincoli:

```
X = int*A    (da (0,z) nel primo caso del p.m.)
Y = RIS      (da y nel primo caso del p.m.)
A = RIS      (da z nel secondo caso del p.m.)
Y = B list   (da y@z)
Z = B list   (da y@z)
```

Ne consegue:

```
X = int * B list
Y = B list
RIS = B list
```

Tipo inferito:

```
(int * B list) -> B list -> B list
```

che in sintassi OCaml corrisponde a:

```
(int * 'a list) -> 'a list -> 'a list
```

Esercizio 3 [Punti 7]

Si definisca, usando i costrutti di programmazione funzionale di OCaml, una funzione `merge` con tipo

```
merge : 'a list list -> 'a list
```

tale che, data una lista di liste `lis`, l'applicazione `merge lis` restituisca una lista contenente tutti gli elementi delle liste contenute in `lis`, senza ripetizioni.
Ad esempio, `merge [[3;4;5];[9;5];[3;6;9;6]]` deve restituire la lista `[3;4;5;9;6]`. L'ordine degli elementi nel risultato può essere qualunque.

SOLUZIONE:

Una possibile soluzione:

```
let merge lis =
  let rec f acc l =
    match l with
    | [] -> acc
    | x::l' -> if List.mem x acc then f acc l'
               else f (acc@[x]) l'
  in
    List.fold_left f [] lis;;
```

Esercizio 4 [Punti 15]

Si estenda il linguaggio MiniCaml con un costrutto per la definizione di funzioni condizionali che, dato un predicato (ovvero funzione che restituisce un valore di tipo `bool`) e due funzioni `f1` e `f2`, fornisce un nuovo tipo di funzione che si comporta come `f1` se il predicato è vero e come `f2` se il predicato è falso. Si assuma che sia il predicato che le due funzioni siano funzioni unarie non ricorsive.

Questa la sintassi concreta del nuovo costrutto:

```
cfun (predicato; funzione1; funzione2)
```

dove `predicato`, `funzione1` e `funzione2` sono espressioni che definiscono rispettivamente il predicato e le due funzioni.

Esempi d'uso (usando l'`Apply` già definita nel linguaggio):

```
Apply(cfun (fun x -> x>0; fun y -> y-1; fun z = z+1) ,  10)   (*risultato 9*)
Apply(cfun (fun x -> x>0; fun y -> y-1; fun z = z+1) , -10)   (*risultato -9*)
```

Si modifichi l'implementazione dell'interprete del linguaggio con quanto serve per gestire i costrutti iterativi descritti in precedenza.

SOLUZIONE:
Una possibile soluzione:

```
type exp = ...
   | CFun of exp * exp * exp

type evT = ...
   | CFunClosure of exp * exp * exp * evT env

let rec eval e s = match e with
     ...
     | CFun (p,f1,f2) -> CFunClosure (p,f1,f2,s)
     | Apply (e1,e2) ->
         let fclosure = eval e1 s in
         (match fclosure with
          ...
          | CFunClosure (p,f1,f2,fDecEnv) ->
              let aVal = eval e2 s in
              let pclosure = eval p fDecEnv in
              (match pclosure with
               | Closure(arg,body,pDecEnv) ->
                   let aenv = bind pDecEnv arg aVal in
                   let cfclosure = (match (eval body aenv) with
                        | Bool(true) -> eval f1 fDecEnv
                        | Bool(false) -> eval f2 fDecEnv
                        | _ -> raise ( RuntimeError "Wrong type")
                        )
                   in
                     (match cfclosure with
                      |(cfarg,cfbody,cfDecEnv) ->
                               let cfenv = bind cfDecEnv cfarg aVal in
                               eval cfbody cfenv
                      | _ -> raise (RuntimeError "Wrong type")
                     )
               | _ -> raise (RuntimeError "Wrong type")
              )
           )
```