Esercizi Vari del 26 Aprile 2019

Esercizio (1 - tratto da Esercizi di Lezione8)

Si supponga di disporre di un linguaggio $\texttt{C}^{\texttt{N}}$ *ottenuto estendendo C con trasmissione per Nome. È possibile in* $\texttt{C}^{\texttt{N}}$ *sostituire ogni occorrenza di* `"while E do C;"`*, per qualsivoglia espressione* `E` *e qualsivoglia comando* `C`*, con l'invocazione di un'opportuna procedura avente comportamento equivalente.*

a. *Si mostri la definizione* $\texttt{C}^{\texttt{N}}$ *di una tale procedura;*

b. *Si confrontino le computazioni (i.e. sequenze di stack di AR) ottenuti nel caso sotto:*

```
while n<k do {res = res * n; n=n+1};
```

c. *Si discutano i risultati di tale confronto.*

Esercizi Vari del 26 Aprile 2019

Esercizio (2 - Espressività Astrazioni Fully Abstract e Trasmissione Parametri)

Sia L un Linguaggio di Programmazione Procedurale con struttura di comandi ed espressioni simile al C, ma con astrazioni procedurali e funzionali di tipo fully abstract e trasmissione di parametri per Valore, Reference, Name (espressione e comando). Si discuta come sia possibile emulare la trasmissione per value-result in L. In particolare, si consideri la seguente dichiarazione di procedura:

```
void PP (p1,...,pk){
   dd;
   cc;
}
```

- `dd` e `cc` *sono una sequenza di dichiarazioni e una di comandi, rispettivamente, di L;*
- `p1,...,pk` *sono* `k` > 0 *parametri formali di L, tranne* `pi`*;*
- `pi` $\equiv$`value-result xi`*, ha identificatore* `xi` *e trasmissione* `value-result`*.*

[a.] *Si mostri una definizione di PP in L che calcola correttamente come se* `xi` *fosse trasmesso* `value-result`

[b.] *Si applichi la trasformazione introdotta sopra, alla seguente procedura:*

```
{
 void bar(value-result x,
          value-result y,
          value-result z){
      y = 2;
      x = 4;
      if(x==y)z=1;
 }
 a = 3;
 b = 0;
 bar(a,a,b);
}
```

[c.] *Si mostri la computazione (i.e. sequenza di stack) ottenuta e la si commenti per giustificarne la correttezza.*

Esercizi Vari del 26 Aprile 2019

Esercizio (3 - Espressività Astrazioni Fully Abstract e Trasmissione Parametri)

Sia L un Linguaggio di Programmazione Procedurale con struttura di comandi ed espressioni simile al C, ma con astrazioni procedurali e funzionali di tipo fully abstract e trasmissione di parametri per Valore, Reference, Name (espressione e comando). Si discuta come sia possibile emulare la trasmissione per result in L. In particolare, si consideri la seguente dichiarazione di procedura:

```
void PP (p1,...,pk){
   dd;
   cc;
}
```

- `dd` e `cc` *sono una sequenza di dichiarazioni e una di comandi, rispettivamente, di L;*
- `p1,...,pk` *sono* `k` > 0 *parametri formali di L, tranne* `pi`*;*
- `pi` $\equiv$`result xi`*, ha identificatore* `xi` *e trasmissione* `result`*.*

[a.] *Si mostri una definizione di PP in L che calcola correttamente come se* `xi` *fosse trasmesso* `result`

[b.] *Si applichi la trasformazione introdotta sopra, alla seguente procedura:*

```
{
 void bar(value x,
          result y,
          result z){
      y = 2;
      x = 4;
      if(x==y)z=1;
 }
 a = 3;
 b = 0;
 bar(a,a,b);
}
```

[c.] *Si mostri la computazione (i.e. sequenza di stack) ottenuta e la si commenti per giustificarne la correttezza.*

Esercizi Vari del 26 Aprile 2019

Esercizio (4 - tratto da Esercizio 8 di Lezione9)

Il termine λ`x.x x` *è un termine sintatticamente corretto del Lambda Calcolo e*

a. *può essere "trascritto" in OCaml ottenendo il termine ?*
b. *che per quanto sintatticamente corretto non ha un tipo associabile: Perchè?*

Esercizi Vari del 26 Aprile 2019

Esercizio (5 - Tratto da Attività di Laboratorio3)

Si consideri il seguente tipo algebrico di OCaml:

```
typedcl = Const of string * int | Var of string * int | VarN of string | Array of string * int
        | SeqDcl of dcl * dcl
```

per gli alberi astratti di dichiarazioni di costanti, variabili inizializzate e non, array e sequenze per SmallC.
Si fornisca una funzione OCaml:

a. *idelList che restituisce la lista degli identificatori dichiarati;*
b. *collisionList che restituisce la lista di tutti gli identificatori che sono dichiarati più volte.*
c. *Applicare ad una sequenza di almeno 5 diverse dichiarazioni.*

Esercizio (6 - tratto da Esercizio 6 di Lezione10)

Si consideri l'ADT polimorfo per valori 'a stack, Immutable, in OCaml, introdotto a Lezione10 e riportato nella pagina successiva. Si indichino:

a. *I valori astratti introdotti dal nuovo tipo (e tipo di dato) 'a stack.*
b. *gli stati concreti con cui tali valori sono rappresentati nell'ADT Stack.*
c. *la funzione di astrazione, AF.*
d. *l'invariante di rappresentazione, I*

ADT Polimorfo in OCaml: Stack Immutable

```
exception Error ;;

module type STACK =
(* Uno 'a stack e' un valore [v1,...,vk] per k>0 ([] per k=0), contenente k valori
   di tipo generico 'a. Il valore vk e' l'ultimo aggiunto (mediante un'operazione
   push) ed il primo ad essere estratto (mediante un'operazione pop)
*)
 sig type 'a stack
  val create_stack: unit -> 'a stack
  val push: 'a stack -> 'a -> 'a stack
  val top: 'a stack -> 'a
  val pop: 'a stack -> 'a stack
end;;
```

```
module Stack =
 (struct
  type 'a sk = E | SK of 'a sk * 'a
  type 'a stack = M of int * ('a sk)
  let create_stack = fun () -> M(0,E)
  let push (M(n,sk)) x =
        if n=100 then raise(Error)
        else M(n+1,SK(sk,x))
  let top (M(n,sk)) = match sk with
                E -> raise Error
                | SK(sk,v) -> v
  let pop (M(n,sk)) = match sk with
                E -> raise Error
                | SK(sk,v) -> M(n-1,sk)
end:STACK);;
```