PROGRAMMAZIONE II - a.a. 2017-18
Settima esercitazione — 13 dicembre 2017

Esercizio 1 [dicembre 2015]. Si estenda il linguaggio funzionale didattico in modo da includere valori di *default* nella dichiarazione del parametro formale di una funzione unaria. Si consideri la seguente dichiarazione di funzione, con una sintassi nello stile di OCaML

```
let myFun a[True] = if a then 5 else 10;;
```

Le invocazioni `myFun(True)` e `myFun()` restituiscono il medesimo risultato, ovvero il valore `5`, mentre l'invocazione di `myFun(False)` restituisce 10.

1. Si estenda la sintassi astratta del linguaggio didattico funzionale in modo da includere valori di default nella dichiarazione di funzioni unarie.

```
type exp = ...
   | DFun of ide * exp * exp
   ...
   | DFunCall of exp

type evT = ...
   | DFunVal of ide * exp * exp * (evT env)
```

2. Si definiscano le regole OCaML dell'interprete per trattare la valutazione di dichiarazione e la chiamata di funzioni unarie con parametri di default.

```
let rec eval (e : exp) (r : evT env) : evT = match e with
     ...
     DFun(p, d, b) -> DFunVal(p, d, b, r) |
     ...
     FunCall(f, eArg) ->
        let fClosure = (eval f r) in
             (match fClosure with
                    ...
                    DFunVal(arg, def, fBody, fDecEnv) ->
                           eval fBody (bind fDecEnv arg (eval eArg r)) |
                    _ -> failwith("non functional value")) |
     DefFunCall(f) ->
         let fClosure = (eval f r) in
              DefFunVal(arg, def, fBody, fDecEnv) ->
                    eval fBody (bind fDecEnv arg (eval def r)) |
              _ -> failwith("non functional value") |
     ...
```

Esercizio 2 [luglio 2014]. Si consideri il seguente programma OCaML, che realizza l'elevazione a potenza (la funzione `power`) con moltiplicazioni successive

```
let rec iterate n f d =
  if n = 0 then d
  else iterate (n-1) f (f d);;    (**)

let power i n =
  let i_times a = a * i in
    iterate n i_times 1;;

# power 3 2;;
- : int = 9
```

1. Si indichi il tipo inferito dall'interprete OCaML per le funzioni `iterate` e `power`.

```
val iterate : int -> ('a -> 'a) -> 'a -> 'a = <fun>
val power : int -> int -> int = <fun>
```

2. Quante volte viene eseguita l'istruzione marcata con `(**)` valutando l'espressione `power 3 2`?

   Viene eseguita 2 volte.

3. Simulando la valutazione dell'espressione `power 3 2`, si mostri la struttura della pila dei record di attivazione subito dopo l'invocazione di `f` e subito dopo l'invocazione di `iterate` per ogni esecuzione della linea marcata con `(**)`.

   Si segua la simulazione alla lavagna!

Esercizio 3. Si consideri un computer con parole di 32 bit e si assuma che il supporto a run-time gestisca lo heap come una lista libera di blocchi di dimensione (fissa) di 4 parole. Nel linguaggio imperativo usato le stringhe vengono allocate nello heap (con la primitiva `new(_)`) come sequenze di caratteri UNICODE (16 bit ognuno), terminanti con un delimitatore (il carattere `'\0'`, come in C). Dato il seguente programma in pseudo-codice, si dica quale valore ha la variabile `tick` quando parte il garbage collector, assumendo che lo heap sia di 1 KByte e che le stringhe non mentano. Durante l'esecuzione del programma si verifica frammentazione interna? In quale percentuale?

```
main() {
   int tick = 0;
   String s = new("Io sono lunga 26 caratteri");
   while (tick <= 1000)  do {
      String s1 = new("Io invece solo 16");
      String s2 = new("e io di meno:15");
      tick++;
   }
}
```

Ogni blocco ha dimensione 16 byte. Dato che 1Kbyte corrisponde a 1024 byte, nello heap trovano dunque spazio 64 blocchi. Ogni carattere occupa 2 byte, e dunque le tre frasi occupano rispettivamente 54 byte, 34 byte e 32 byte. Per la prima servono 4 blocchi, per la seconda 3 e per la terza 2. Dunque, il garbage collector parte per il più piccolo x tale che $64 - 4 - (3 + 2)x \leq 0$, ovvero, dopo 12 iterazioni. Si verifica una ampia frammentazione interna (il cui calcolo è lasciato al lettore).