PROGRAMMAZIONE II - a.a. 2018-19
Esercitazione extra — 12 dicembre 2018

Esercizio 1 [luglio 2014]. Si estenda il linguaggio didattico funzionale in modo da includere espressioni e valori di tipo **record**. Un *valore (di tipo) record* è un dato strutturato composto da un numero finito di coppie *nome-valore*, detti **campi**. Analogamente, una *espressione (di tipo) record* è composta da un numero finito di coppie *nome-valore*. La valutazione di una espressione record produce un valore record.

Un identificatore può esser legato a un valore record tramite il costrutto `let`: nel seguente esempio (in sintassi OCaML-like) si evidenzia che l'espressione record che compare nel `let` è valutata in un valore record

```
# let rect = record{base = 5 * 5, altezza = 10 - 6}
val rect = record{base = 25, altezza = 4}
```

Sui record è definita l'operazione di selezione di una componente. Continuando l'esempio precedente

```
# let b = rect.base
val b = 25
```

1. Si estenda la sintassi astratta del linguaggio didattico funzionale in modo da includere valori ed espressioni record e l'operatore di selezione.

   Prima soluzione con lista di coppie

```
type exp = ...
   | Rec of (ide * exp) list
   | Select of exp * ide

type evT = ...
   | RecVal of (ide * evT) list
   ...
```

   Seconda soluzione con funzione

```
type exp = ...
   | Rec of (ide * exp) list
   | Select of exp * ide

type evT = ...
   | RecVal of (evT env)
   ...
```

2. Si estenda il codice OCaML dell'interprete per trattare la valutazione di espressioni record e dell'operatore di selezione.

   Prima soluzione con lista di coppie

```
let rec eval (e : exp) (r : evT env) : evT = match e with
     ...
     Rec(lst) -> RecVal(evalList lst r) |
     ...
     Select(rd, id) ->
         (match (eval rd r) in
              RecVal(rdPairs) -> lookup id rdPairs |
              _ ->  failwith("wrong select value")) |
     ...
and evalList (lst : (ide * exp) list) (r : evT env) : (ide * evT) list = match lst with
     [ ] -> [ ] |
     (id, arg) :: rest -> (id, eval arg r) :: evalList rest r |
and lookup (id : ide) (lst : (ide * evT) list) : evT = match lst with
     [ ] -> Unbound |
     (id1, val):: rest -> if (id = id1) then val else (lookup id rest);;
```

Seconda soluzione con funzione

```
let rec eval (e : exp) (r : evT env) : evT = match e with
     ...
     Rec(lst) -> RecVal(evalEnv lst r) |
     ...
     Select(rd, id) ->
         (match (eval rd r) in
              RecVal(r1) -> (applyenv r1 id) |
              _ ->  failwith("wrong select value")) |
     ...
and evalEnv (lst : (ide * exp) list) (r : evT env) : (evT env) = match lst with
     [ ] -> (emptyEnv Unbound) |
     (id, arg) :: rest -> (bind id (eval arg r) (evalEnv rest r));;
```

Esercizio 2 [gennaio 2015]. Si consideri il seguente programma OCaML

```
let n = 5;;
let h = fun x -> n + x;;
let rec f p n =
   let g = fun y -> n * y in
      if n = 0 then p 1
      else if n > 1 then f g (n-1)
           else f p (n-1);;
f h 2;;
```

1. Si indichi il tipo inferito dall'interprete OCaML per la funzione ricorsiva `f`.

```
val n : int = 5
val h: int -> int = <fun>
val f: (int -> int) -> int -> int = <fun>
```

2. Si simuli la valutazione del programma mostrando la struttura della pila dei record di attivazione.

   Si segua lo svolgimento alla lavagna!

3. Si indichi il valore restituito dal programma.

```
- : int = 2
```