Esercitazione

Esercizio 1 (parte 1)

- Si estenda il linguaggio funzionale didattico in modo da includere valori di *default* nella dichiarazione del parametro formale di una funzione unaria.
- Esempio (sintassi nello stile di Ocaml)
  - let myFun a[True] = if a then 5 else 10;;
  - [valore di deafult]
- Le invocazioni **myFun(True)** e **myFun()** restituiscono il medesimo risultato, **5**, mentre l'invocazione di **myFun(False)** restituisce **10**.

Esercizio 1

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

Soluzione


```
type exp = ...
   | Dfun of ide * exp * exp
   ...
   | Dcall of ide * exp
   | Ncall of ide

type evT= ...
   | Dfunval of ide * exp * exp * eval env
```

Esercizio 1 (parte 2)

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

Soluzione

```
let rec eval e r = match e with
  ...
  | Dfun(p, d, body) -> Dfunval(p, d, body, r)
  ...
  | Dcall(e1, e2) -> match eval(e1, r) with
            Dfunval(p, d, body, r1) ->
                    eval(body, bind(r1, a, eval(e2, r)))
            | _ -> failwith "wrong exp"
  | Ncall(e) -> match eval(e, r) with
             Dfunval(p, d, body, r1) ->
                    eval(body, bind(r1, a, eval(d, r1)))
            | _ -> failwith "wrong exp"
```

Esercizio 2


```
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
```

Simulando la valutazione dell'espressione **power 3 2**, si mostri la struttura della pila dei record di attivazione.

Un caso di studio: passaggio per nome

Passaggio per nome

- L'espressione passata in corrispondenza di un parametro per nome "x" non viene valutata al momento del passaggio
  - ogni volta che (eventualmente) si incontra una occorrenza del parametro formale "x" l'espressione passata a "x" viene valutata
- Per definire (funzioni e) sottoprogrammi non stretti su uno (o più di uno) dei loro argomenti
  - l'attivazione può dare un risultato definito anche se l'espressione, se valutata, darebbe un valore indefinito (errore, eccezione, non terminazione)
    - semplicemente perché in una particolare esecuzione "x" non viene mai incontrato

Passaggio per nome nei linguaggi imperativi

- Dato che l'espressione si valuta ogni volta che si incontra il parametro formale
  - nella memoria corrente
  - e l'espressione contiene variabili che possono essere modificate (come non locali) dal sottoprogramma
- Diverse occorrenze del parametro formale possono dare valori diversi

Passaggio per nome: semantica

- Conisderiamo un linguaggio funzionale: l'espressione passata in corrispondenza di un parametro per nome "x"
  - non viene valutata al momento del passaggio
  - viene (eventualmente) valutata ogni volta che si incontra una occorrenza del parametro formale "x"
- Una espressione non valutata è una chiusura

  exp * evT env
- La valutazione dell'occorrenza di "x" si effettua valutando l'espressione della chiusura nell'ambiente della chiusura

Passaggio per nome


```
    type exp = ...
        | Namexp of exp
        | Namden of ide
   type evT = :
     and namexp = exp * dval env
let rec eval((e: exp), (r: evTenv) =
   match e with
   | ...
   | Namexp e1 -> namexp(e1,r)
   | Nameden(i) -> match applyenv(r, i) with
                        namexp(e1, r1) -> eval(e1, r1)
```

Espressioni per nome e funzioni

- Una espressione passata per nome è chiaramente simile alla definizione di una funzione (senza parametri)
  - che "si applica" ogni volta che si incontra una occorrenza del parametro formale
- Stessa soluzione semantica delle funzioni
  - chiusura in semantica operazionale (e nelle implementazioni)
- L'ambiente che viene fissato (nella chiusura) è quello di passaggio
  - che, per le espressioni, è l'equivalente della definizione

  ma mentre la semantica delle funzioni è influenzata dalla regola di scoping, ciò non è vero per le espressioni passate per nome
  - che vengono comunque valutate nell'ambiente di passaggio, anche con lo scoping dinamico
- Il passaggio per nome è previsto in nobili linguaggi come ALGOL e LISP
  - è alla base dei meccanismi di valutazione lazy di linguaggi funzionali moderni come Haskell
  - può essere simulato in ML passando funzioni senza argomenti!

Argomenti funzionali à la LISP

- LISP ha lo scoping dinamico, pertanto il dominio delle funzioni è

  type efun = exp
- Un argomento formale "x" di tipo funzionale dovrebbe denotare un eval della forma Funval(efun)
- Se "x" viene successivamente applicato, il suo ambiente di valutazione dovrebbe correttamente essere quello del momento dell'applicazione
- La semantica di LISP prevede invece che l'ambiente dell'argomento funzionale venga congelato nel momento del passaggio
- Questo porta alla necessità di introdurre due diversi domini per funzioni e argomenti funzionali

  type funarg = exp * evT env
- I domini degli argomenti funzionali sono chiaramente identici ai domini delle funzioni con scoping statico
  - ma l'ambiente rilevante (quello della chiusura) è quello del passaggio e non quello di definizione

Argomenti funzionali à la LISP, 2

- Quando una funzione viene passata come argomento a un'altra funzione in una applicazione
  - passando un nome di funzione, una lambda astrazione o una qualunque espressione la cui valutazione restituisca una funzione

  la funzione deve essere prima di tutto "chiusa" con l'ambiente corrente r
  - inserendo r nella chiusura

  da **efun = exp** a **funarg = exp * evT env**
- Nelle implementazioni coesisteranno funzioni rappresentate dal solo codice e argomenti funzionali rappresentati da chiusure (codice, ambiente)
  - con un numero notevole di complicazioni associate

Ritorno di funzioni à la LISP

- Quando una applicazione di funzione restituisce una funzione, il valore restituito dovrebbe essere di tipo efun
- In alcune implementazioni di LISP si segue la medesima strada degli argomenti funzionali
  - viene restitito un valore di tipo funarg
- Anche questa scelta complica notevolmente l'implementazione
  - si hanno gli stessi problemi dello scoping statico (retention), senza averne i vantaggi (verificabilità, ottimizzazioni)

Chiusure per tutti i gusti

- Nelle semantiche operazionali e nelle implementazioni un'unica soluzione per
  - espressioni passate per nome (con tutte e due le regole di scoping)
  - funzioni, procedure e classi con scoping statico
  - argomenti funzionali e ritorni funzionali con scoping dinamico à la LISP