Astrazione sul controllo

R.S

April 15, 2025

Passaggio di parametri

Passaggio di parametri per valore:

- il parametro formale è una variabile locale, il parametro attuale viene valutato con r-value
- le modifiche al parametro formale non si ripercuotono su quello attuale
- anche dati grossi vengono copiati a scapito dell'efficienza (formale trattato come costante)

Passaggio di parametri per riferimento:

- il parametro è un puntatore ad un riferimento di memoria
- posso utilizzarlo con parole chiave (&, ref..) oppure simulandolo coi puntatori

```
void foo (ref int x){ x = x+1;}
...
y = 1;
foo(y);
foo(V[y+1]);
```

Passaggio di parametri per sharing:

- l'assegnazione crea la condivisione di una copia (Java, Python...), stessa cosa il passaggio per valore
- implementato ad esempio in array e liste

```
def f(a_list):
    a_list += [1]
m = []
f(m)
print(m)
```

Call by name:

- l'espressione del parametro viene passata senza valutazione
- una chiamata esegue il corpo della procedura P valutando il parametro soltanto quando viene utilizzato (di default in Algol-W)
- la valutazione del parametro viene fatta ogni volta che il parametro viene utilizzato (effetti collaterali ripetuti)

Chiusura: la coppia [exp, env] dove exp è il puntatore al codice dell'espressione ed env è un puntatore di catena statica a RdA

Call by need:

- l'argomento viene valutato al massimo una volta
- le valutazioni multiple restituiscono sempre lo stesso risultato perché la prima valutazione viene conservata
- più efficiente del call by name, utilizzati nei linguaggi funzionali

Funzioni di ordine superiore

Si possono passare funzioni come parametro ad altre funzioni e possono essere anche restituite come risultato

Semantica

- la funzione f viene passata come parametro e valutata eventualmente più volte
- problemi di scope delle variabili utilizzate da f (scope statico oppure dinamico)

Binding

- l'ambiente è quello della chiamata di g con parametro la funzione f → deep binding
- l'ambiente è quello della chiamata di f dentro g → shallow binding

Implementazione della chiusura

- nel RdA di g si inserisce la chiusura [code, env] della funzione f
- alla chiamata di f si alloca il suo record di attivazione con puntatore alla catena statica env

```
int x = 1;
int f (int y){ return x + y; }
int g (int h (int i)){
int x = 2;
return h(3) + x;}
...
int x = 4;
int z = g(f);
```

Politiche di binding
Ambiente di valutazione della funzione f (con nome h) dentro g:

- scope statico → ambiente della definizione
- scope dinamico:
  - al momento della creazione del legame h → f ovvero alla chiamata di f con parametro g → **deep binding**
  - al momento della chiamata di f in g tramite h → **shallow binding**

Implementazione dello scope dinamico
L'implementazione può avvenire sia con deep che con shallow binding:

- Deep binding:
  - alla funzione passata come parametro si associa la chiusura per mantenere in memoria l'RdA della chiamata
  - per le procedure chiamate tramite parameteo si usa un link statico → salto della pila
- Shallow binding:
  - non necessita la chiusura → si associa soltanto il codice della funzione
  - per accedere a x si risale la pila → uso di A-List e CRT

Gestione delle eccezioni
I primi linguaggi utilizzavano l'istruzione **GOTO** → difficile impementare l'uscita da una procedura
Introduzione delle **eccezioni**:

- definizione dei blocchi protetti → insiemi di eccezioni e relativi gestori
- il programma solleva un'eccezione → la computazione viene interrotta e si cerca un gestore
- gestore:
  - codice da eseguire soltanto in caso di eccezione relativa ad un blocco protetto
  - ricerca del gestore → terminazione dei comandi, uscita da cicli e deallocazione di RdA

Se l'eccezione non viene gestita nella procedura corrente → ri-sollevata nel punto di chiamata e propagata lungo la catena dinamica fino a quando non si raggiunge un gestore oppure il top level