7 Procedure

Nei linguaggi di programmazione è di fondamentale importanza riuscire a scrivere dei programmi che siano facili da leggere e da modificare. Lo scopo di funzioni e procedure è quello di permettere la definizione di operatori e comandi di più alto livello di quelli forniti dal linguaggio. Tali definizioni, per essere di maggiore utilità, devono essere parametriche: la loro funzionalità deve essere definita non su oggetti reali, ma su parametri che, a seconda delle volte, verranno istanziati su oggetti diversi. A questo scopo la definizione di funzione o procedura è essenzialmente composta da due parti: una *intestazione* (detta anche *prototipo* o *header*) e un *corpo* (*body*). L'intestazione specifica il nome della funzione o procedura e il numero e il tipo dei parametri sui quali opererà ogni volta che verrà utilizzata (chiamata), mentre il corpo specifica le operazioni che saranno eseguite sugli argomenti al momento della chiamata.

È fondamentale avere sempre presente la distinzione tra una *dichiarazione* di funzione o procedura e la sua *chiamata*. La dichiarazione corrisponde alla definizione dell'operatore (nel caso di funzioni) o del comando (nel caso di procedure), mentre la chiamata corrisponde all'utilizzo di tale operatore o comando.

In queste note, per semplicità, non trattiamo funzioni ma solo procedure con un unico parametro. Per prima cosa estendiamo la sintassi delle dichiarazioni per inglobare le dichiarazioni e le chiamate di procedure.

Estensioni sintattiche

```
Dec ::= void Ide (Ptype Ide) Block
Com ::= Ide(Exp);
```

Data una dichiarazione del tipo `void p (T x) B` ed una corrispondente chiamata del tipo `p(E)`, `x` viene detto *parametro formale* della procedura, `B` viene detto il *corpo* della procedura e (il valore di) `E` viene detto il *parametro attuale* (dall'inglese *actual*, cioè *reale*, *effettivo*) della chiamata. Ad una stessa dichiarazione possono corrispondere ovviamente più chiamate con parametri attuali diversi.

Una dichiarazione di procedura specifica un comando di alto livello ed il suo effetto dovrà essere di introdurre nello stato una associazione tra il nome della procedura e tale operatore. Sintatticamente, la chiamata di tale operatore è un comando, la cui valutazione corrisponderà a valutarne il corpo, dopo aver opportunamente legato il parametro formale con il parametro attuale della chiamata.

Nel corpo della dichiarazione di procedura possono comparire degli identificatori che non sono locali alla dichiarazione stessa, sono cioè stati dichiarati al di fuori della procedura. Per tale motivo sarà necessario valutare il corpo in uno stato più ampio (*globale*) rispetto a quello locale alla procedura stessa. Consideriamo, per esempio, la seguente dichiarazione:

```
void foo (int x)
    {
      int z = x+1;
      y = y + z;
    }
```

È chiaro che `y` è una variabile *globale* per la procedura `foo`, poiché il nome `y` non compare né come parametro formale né all'interno delle dichiarazioni locali di `foo`. Dunque, al momento

della valutazione di una chiamata di `foo`, ad esempio `foo(5)`, occorrerà fare riferimento al valore della variabile `y`, ovvero ad uno stato *globale*.

Scoping statico e dinamico

Si impone fin da ora una scelta su tale stato globale. Ovviamente dovrà essere uno stato in cui la memoria e lo heap rispecchiano quelli presenti al momento della chiamata della procedura e non quella al momento della sua definizione: questo perché si vuole che la valutazione avvenga considerando il valore aggiornato delle variabili del programma. Se considerassimo, al momento della chiamata, la memoria e lo heap presenti al momento della dichiarazione, non verrebbero presi in considerazione tutti i cambiamenti che hanno avuto luogo successivamente alla dichiarazione stessa.

Per quanto riguarda l'ambiente da usare nella valutazione di una chiamata si hanno due possibili scelte:

- usare l'ambiente che si ha al momento della dichiarazione;
- usare l'ambiente che si ha al momento della chiamata.

Queste due scelte vengono indicate, rispettivamente, come *portata degli identificatori* (in inglese *scoping*) *statica* o *dinamica*. Informalmente, possono essere definite nel modo seguente:

- *scoping statico*: la valutazione di una chiamata di procedura usa, come ambiente globale, quello che si ha al momento della dichiarazione della procedura stessa. In questo modo si fissa, una volta per tutte, l'oggetto cui si riferisce un nome globale `y` usato nel corpo della procedura: esso è l'oggetto $\rho(\mathtt{y})$ associato a `y` nell'ambiente ρ in cui viene valutata la dichiarazione di procedura. Dunque, chiamate diverse della stessa procedura fanno riferimento agli stessi oggetti globali.
- *scoping dinamico*: la valutazione di una chiamata di procedura usa, come ambiente globale, quello che si ha al momento della chiamata. In questo modo, i nomi globali usati nel corpo della procedura possono avere significati diversi (alcuni nomi possono essere stati ridichiarati), ed in ogni caso questi nomi sono relativi all'ambiente che si ha quando la procedura viene chiamata.

Esempi

Consideriamo il seguente frammento di programma

```
{
  int y = 10;
  void foo (int x)  {y = y + x;}
  {
     int y = 150;
     foo(5);
  }
       ...
}
```

In regime di scoping statico il valore associato nello stato, dopo la chiamata di procedura, alla variabile `y` dichiarata nel blocco principale è `15`, mentre in regime di scoping dinamico è `10` in quanto, in quest'ultimo caso, la modifica sulla variabile globale `y` dovuta all'esecuzione del corpo di `foo` ha effetto sulla variabile `y` del blocco annidato.

Nel nostro linguaggio, la regola adottata è quella statica: il comando rappresentato da una procedura non usa, al momento della chiamata, l'ambiente corrente, bensì quello presente al momento della dichiarazione.

Estensioni semantiche: Scoping statico

Viste le considerazioni precedenti, siamo ora in grado di fornire la semantica formale di dichiarazione e chiamata di procedure. L'unico scopo della regola per la dichiarazione di procedura è di associare nell'ambiente, al nome della procedura, tutte le informazioni che serviranno al momento della valutazione di una chiamata. Queste informazioni sono:

- il corpo della procedura;
- il nome del parametro formale;
- l'ambiente di definizione (per gestire lo scoping statico).

Tutto ciò può essere fatto associando al nome della procedura una funzione semantica che consentirà, al momento della chiamata, di valutare il corpo della procedura, dati il valore del parametro attuale e la memoria e lo heap presenti al momento della chiamata. Formalmente:

$$\mathcal{S}em_d\ [\texttt{void p(T x) B}]\ \rho\ \mu\ \zeta = \langle \rho[{}^{psem}/_{p}]^{add}, \mu, \zeta \rangle$$

dove $psem\ a\ \mu'\ \zeta' = \langle \mu'', \zeta'' \rangle$

dove $\mathcal{S}em_c\ \texttt{B}\ \omega[{}^{\ell}/_{\texttt{x}}]^{add}.\rho\ \ \omega[{}^{a}/_{\ell}]^{add}.\mu'\ \ \zeta' = \langle \nu.\mu'', \zeta'' \rangle$

e $\ell = succloc\ \mu'$

Osserviamo che:

- l'oggetto semantico associato al nome della procedura è una funzione semantica *psem* che si aspetta tre argomenti:
  - un valore a, il parametro attuale
  - una memoria e uno heap, μ', ζ'
- la funzione *psem* non fa altro che valutare il corpo `B` della procedura in uno stato in cui
  - all'ambiente ρ corrente, quindi presente al momento della dichiarazione, è stato aggiunto un frame che contiene l'associazione tra il parametro formale `x` ed una nuova locazione ℓ
  - alla memoria μ', il parametro di *psem* che rappresenta la memoria al momento della chiamata, è stato aggiunto un frame che contiene l'associazione tra la locazione ℓ ed il parametro attuale a
  - lo heap ζ' è il parametro di *psem* che rappresenta lo heap al momento della chiamata.

Osserviamo che lo stato a partire dal quale la funzione *psem* andrà, di volta in volta, a valutare il corpo della procedura è quello che, a partire dallo stato $\langle \rho, \mu', \zeta' \rangle$, si otterrebbe dalla valutazione di un blocco in cui:

- viene dichiarata la variabile `x` ed inizializzata al valore v del parametro attuale;
- viene eseguito il comando `B`

Ciò mette anche in evidenza che il parametro formale è, a tutti gli effetti, trattato come una variabile.

L'estensione proposta richiede anche una estensione del codominio dei frame ambiente che, oltre alle locazioni, devono prevedere l'insieme degli oggetti semantici associati alle procedure (le funzioni cioè come la funzione *psem* della regola precedente). Detto $Proc$ l'insieme

$$Proc = \{psem \mid psem : \mathtt{Val} \to M \ \to \mathcal{H} \to M \ * \mathcal{H}\}$$

i frame ambiente sono ora funzioni

$$\varphi : \mathtt{Ide} \to (\mathtt{Loc}^{\mu} \cup Proc)_{\perp}$$

La semantica della chiamata di procedura consiste semplicemente nel valutare la funzione semantica associata al nome della procedura, alla quale vengono forniti i tre parametri necessari: il valore del parametro attuale, la memoria e lo heap presenti al momento della chiamata stessa. Formalmente:

$$\mathcal{S}em_c \ \mathtt{p(E)} \ \rho \ \mu \ \zeta = psem \ \ v \ \ \mu \ \ \zeta$$

dove $psem = \rho(\mathtt{p})$

e $v = \mathcal{S}em_e \ \mathtt{E} \ \rho \ \mu \ \zeta$

La stessa regola può essere scritta in maniera più compatta come segue:

$$\mathcal{S}em_c \ \mathtt{p(E)} \ \rho \ \mu \ \zeta = \rho(\mathtt{p}) \ (\mathcal{S}em_e \ \mathtt{E} \ \rho \ \mu \ \zeta) \ \ \mu \ \ \zeta$$

Scoping dinamico

Per completezza riportiamo di seguito le regole semantiche che si avrebbero nel nostro modello per trattare un regime di scoping dinamico.

$$\mathcal{S}em_d\ [\texttt{void p(T x) B}]\ \rho\ \mu\ \zeta = \langle \rho[^{psem}/_{p}]^{add}, \mu, \zeta \rangle$$

dove $psem\ a\ \rho'\ \mu'\ \zeta' = \langle \mu'', \zeta'' \rangle$

dove $\mathcal{S}em_c\ \texttt{B}\ \omega[^{\ell}/_{\texttt{x}}]^{add}.\rho'\ \ \omega[^{a}/_{\ell}]^{add}.\mu'\ \ \zeta' = \langle \nu.\mu'', \zeta'' \rangle$

e $\ell = succloc\ \mu'$

$$\mathcal{S}em_c\ \texttt{p(E)}\ \rho\ \mu\ \zeta = \rho(\texttt{p})\ (\mathcal{S}em_e\ \texttt{E}\ \rho\ \mu\ \zeta)\ \ \rho\ \ \mu\ \ \zeta$$

Passaggio dei parametri per valore e per indirizzo

Le regole semantiche che abbiamo visto, indipendentemente dal regime di scoping scelto, prevedono che il parametro formale di una procedura sia trattato come una variabile locale alla procedura stessa, che viene inizializzata al *valore* del parametro attuale. Una conseguenza di ciò è che eventuali modifiche del parametro formale, conseguenti ad esempio ad assegnamenti alla variabile stessa presenti nel corpo della procedura, non hanno ripercussioni sullo stato globale una volta terminata una chiamata della procedura stessa. Le uniche eventuali modifiche allo stato globale sono quelle che si posson ottenere a seguito dell'utilizzo di nomi globali e dunque, in regime di scoping statico, solo a variabili globali presenti al momento della dichiarazione della procedura stessa. Ciò contrasta con la natura delle procedure che dovrebbero corrispondere a comandi di alto livello e, come sappiamo, lo scopo dei comandi è proprio quello di modificare lo stato nelle sue componenti modificabili, memoria e heap.

Supponiamo ad esempio di voler definire un comando di alto livello il cui scopo è quello di aggiungere 2 al valore di una variabile intera. Una procedura del tipo

```
void add2 (int x) { x = x + 2; }
```

non fa al caso nostro. Se prendiamo ad esempio il seguente frammento di codice

```
void add2 (int x) { x = x + 2; }
int z = 20;
add2 (z);
...
```

è facile convincerci che il valore di `z`, al termine della chiamata della procedura, non ha subito alcuna modifica. Vediamo infatti l'evoluzione dello stato (trascurando la componente heap che

in questo esempio non ha alcun ruolo). nelle figure seguenti, indichiamo con $asem \in Proc$ la funzione semantica associata alla procedura `add2`, ovvero:

$$asem\ a\ \mu'\ \zeta' = \langle \mu'', \zeta'' \rangle$$

dove $\mathcal{S}em_c\ \{\texttt{x} = \texttt{x} + \texttt{2};\}\ \omega[^{\ell}/_{\texttt{x}}]^{add}.\rho_0\ \ \omega[^{a}/_{\ell}]^{add}.\mu'\ \ \zeta' = \langle \nu.\mu'', \zeta'' \rangle$
e $\ell = succloc\ \mu'$

in cui supponiamo per semplicità che l'ambiente ρ_0, al momento della dichiarazione di `add2` sia del tipo $\omega.\Omega$.

Lo stato dopo le due dichiarazioni è il seguente.

Ambiente

`z`	ℓ_z
`add2`	$asem$

Memoria

ℓ_z	20

e dunque l'esecuzione di `add2(z)` corrisponde a valutare il comando `x = x+2;` a partire dallo stato seguente, come si evince da una semplice analisi di $asem$:

Ambiente

`x`	ℓ_x

Memoria

ℓ_x	20
ℓ_z	20

La memoria risultante dall'esecuzione del corpo della procedura è

Memoria

ℓ_x	**22**
ℓ_z	20

da cui si evidenzia che l'effetto dell'assegnamento si ripercuote solo sulla locazione associata ad `x`, il parametro formale. La memoria risultante dall'esecuzione della procedura è quella precedente, una volta rimosso il frame in testa alla pila, ovvero

Memoria

ℓ_z	20

Per ovviare a questo inconveniente legato al passaggio dei parametri per valore, alcuni linguaggi di programmazione prevedono un nuovo meccanismo, detto passaggio *per variabile* o per *indirizzo*. Lo scopo del passaggio per variabile è fare in modo che, al momento della chiamata, il parametro formale e il parametro attuale siano di fatto la *stessa* variabile: in questo modo ogni modifica fatta nel corpo della procedura sul parametro formale è implicitamente fatta sul parametro attuale, contribuendo così alla modifica della memoria.

Nel nostro linguaggio, e nel linguaggio `C`, gli effetti del passaggio per riferimento si possono ottenere attraverso i puntatori, ovvero attraverso il passaggio *per valore* di un puntatore. Vediamolo attraverso l'esempio precedente rivisitato.

Consideriamo il seguente frammento di codice

```
void add2 (int *x) { *x = *x+2; }
int z = 20;
add2 (&z);
...
```

Notiamo che:

- il parametro formale della procedura è ora di tipo `int *` e dunque un puntatore ad una variabile intera;
- i riferimenti al parametro formale sono ora diventati riferimenti a `*x`, ovvero alla variabile *puntata* dal parametro formale
- nella chiamata di procedura il parametro attuale è l'indirizzo di `z`

Vediamo, come prima, l'evoluzione dello stato, indicando questa volta con $asem \in Proc$ la funzione semantica associata alla procedura `add2`, ovvero:

$$asem\ a\ \mu'\ \zeta' = \langle \mu'', \zeta'' \rangle$$

dove $\mathcal{S}em_c\ \{*\mathtt{x} = *\mathtt{x} + 2;\}\ \omega[^{\ell}/_{\mathtt{x}}]^{add}.\rho_0\ \ \omega[^{a}/_{\ell}]^{add}.\mu'\ \ \zeta' = \langle \nu.\mu'', \zeta'' \rangle$
e $\ell = succloc\ \mu'$

in cui, ancora, supponiamo per semplicità che l'ambiente ρ_0, al momento della dichiarazione di `add2` sia del tipo $\omega.\Omega$.

Lo stato dopo le due dichiarazioni è il seguente.

Ambiente

z	ℓ_z
add2	$asem$

Memoria

ℓ_z	20

Questa volta, l'esecuzione di `add2(&z)` corrisponde a valutare il comando `*x = *x + 2;` a partire dallo stato seguente:

Ambiente

x	ℓ_x

Memoria

ℓ_x	ℓ_z

ℓ_z	20

dal momento che il *valore* del parametro attuale (`&z`) è ora la locazione ℓ_z associata, nello stato presente al momento della chiamata, alla variabile `z`. La memoria risultante dall'esecuzione del corpo della procedura è dunque:

Memoria

ℓ_x	ℓ_z

ℓ_z	**22**

da cui si evidenzia che l'effetto dell'assegnamento si ripercuote questa volta sul valore della variabile `z`. La memoria risultante dall'esecuzione della procedura è quella precedente, una volta rimosso il frame in testa alla pila, ovvero

Memoria

ℓ_z	**22**

Dunque nel nostro linguaggio, come in `C`, la presenza del tipo puntatore consente di *simulare* il passaggio per variabile attraverso un passaggio *per valore* di un puntatore. Si noti che, mentre nel caso di parametri formali il cui tipo è un tipo semplice (categoria sintattica `Type`) il corrispondente parametro attuale in una chiamata può essere una generica espressione, nel caso di parametri formali di tipo puntatore (categoria sintatti `Ptype`) il parametro formale può essere solo una espressione il cui valore è una locazione. Nel nostro semplice linguaggio ciò corrisponde a parametri attuali del tipo `&x`, dove `x` è una variabile, o `p`, dove `p` è una variabile di tipo puntatore[4].

[4]Nel linguaggio `C` esistono altre espressioni (nell'aritmetica dei puntatori) il cui valore è una locazione.