Approfondimento 2.1

Non è questo il testo dove trattare esaurientemente queste tecniche semantiche. Ci accontenteremo di dare un semplice esempio delle tecniche basate sui sistemi di transizione per dare semantica ad un rudimentale linguaggio imperativo. Questa tecnica, chiamata SOS (Semantica Operazionale Strutturata, [80]) è molto versatile ed è stata effettivamente utilizzata per dare semantica ad alcuni linguaggi, tra i quali Standard ML, un influente linguaggio funzionale.

La Figura 16.1 riporta la grammatica di un semplice linguaggio. Per motivi di leggibilità (e di concisione della semantica) si è data una grammatica con infinite produzioni per i simboli non terminali *Num* e *Var*. La presenza esplicita di parentesi in corrispondenza ad ogni operatore binario elimina eventuali problemi di ambiguità. Introduciamo qualche notazione: indicheremo con n il generico *Num* (costante numerica); con X la generica *Var* (variabile); con a la generica *AExp* (espressione aritmetica); con b la generica *BExp* (espressione booleana, dove **tt** e **ff** indicano i valori vero e falso, rispettivamente); con c il generico *Com* (comando). Useremo al bisogno pedici per distinguere tra oggetti della stessa categoria sintattica (per esempio, avremo le *AExp* a_1, a_2 ecc.).

Stato La semantica di un comando nel nostro linguaggio è data con riferimento ad un semplice modello di memoria, che mantiene i valori delle *Var*. In questo modello, uno *stato* è una sequenza finita di coppie della forma (X, n) che possiamo leggere come "Nello stato corrente la variabile X ha valore n" (nel piccolo linguaggio che usiamo, il valore sarà sempre un valore intero, ma possiamo facilmente immaginare situazioni più complesse). Dato un comando c (cioè un albero di derivazione corretto secondo la grammatica della Figura 16.1), il suo stato di riferimento è dato da una sequenza di coppie che comprende tutte le *Var* che sono nominate in c. Indicheremo il generico stato con σ o τ, eventualmente con pedici.

Avremo bisogno di esprimere alcune operazioni sugli stati: la modifica di uno stato esistente e il recupero del valore di una variabile nello stato corrente. A tal fine, dato uno stato σ, una *Var* X ed un valore v, denotiamo con $\sigma[X \leftarrow v]$ quel nuovo stato che coincide in tutto con σ, salvo che ad X è associato il valore v (perdendo dunque l'associazione precedente). Dato uno stato σ ed una variabile X, indichiamo con $\sigma(X)$ il valore che σ associa a X; tale valore è indefinito se X non compare in σ (σ è dunque una funzione parziale).

Esempio 16.1 Fissato $\sigma = [(X, 3), (Y, 5)]$, si ha $\sigma[X \leftarrow 7] = [(X, 7), (Y, 5)]$. Si ha anche $\sigma(Y) = 5$ e $\sigma[X \leftarrow 7](X) = 7$; $\sigma(W)$ è indefinito.

$$
\begin{array}{lcl}
Num & ::= & 1 \mid 2 \mid 3 \mid \ldots \\
Var & ::= & X_1 \mid X_2 \mid X_3 \mid \ldots \\
 & & \\
AExp & ::= & Num \mid Var \mid (AExp + AExp) \mid (AExp - AExp) \\
BExp & ::= & \mathbf{tt} \mid \mathbf{ff} \mid (AExp == AExp) \mid \neg BExp \mid (BExp \wedge BExp) \\
 & & \\
Com & ::= & \mathbf{skip} \mid Var := AExp \mid Com; Com \mid \\
 & & \mathbf{if}\ BExp\ \mathbf{then}\ Com\ \mathbf{else}\ Com \mid \mathbf{while}\ BExp\ \mathbf{do}\ Com
\end{array}
$$

Figura 16.1 Sintassi di un semplice linguaggio imperativo.

Transizione La semantica operazionale strutturata può esser vista come un modo elegante per definire il funzionamento di una macchina astratta, senza entrare in alcun dettaglio relativo alla sua implementazione[4]. Tale funzionamento è espresso in termini di passi elementari di computazione della macchina astratta. Il modo formale con cui la semantica operazionale strutturata definisce il significato di un programma c è quello di una *transizione*, che esprime un passo della trasformazione (sullo stato e/o sul programma stesso) indotta dall'esecuzione del comando. La forma più semplice di una transizione è

$$\langle c, \sigma \rangle \to \tau,$$

dove c è un comando, σ è lo *stato di partenza*, e τ lo *stato di arrivo*. L'interpretazione che diamo a questa notazione è che, se iniziamo l'esecuzione di c nello stato σ, l'esecuzione termina (in un solo passo) nello stato τ. Per esempio, la transizione che definisce il comando **skip** è

$$\langle \mathbf{skip}, \sigma \rangle \to \sigma,$$

da cui si vede che si tratta del comando che non fa nulla: partendo da un qualsiasi stato σ, il comando **skip** termina lasciando inalterato lo stato.

In casi più complessi del comando **skip**, una situazione "terminale" sarà raggiunta non in un solo passo, bensì mediante piccoli passi, che trasformano via via lo stato (a partire da σ) e il comando c (eseguendone via via una parte fino a "consumarlo" tutto). Questi piccoli passi sono espressi da transizioni della forma

$$\langle c, \sigma \rangle \to \langle c', \sigma' \rangle.$$

Ad esempio, una delle transizioni che definiscono il comando condizionale sarà

$$\langle \mathbf{if}\ \mathtt{tt}\ \mathbf{then}\ c_1\ \mathbf{else}\ c_2, \sigma \rangle \to \langle c_1, \sigma \rangle,$$

[4] Usando una terminologia che riprenderemo negli ultimi capitoli del testo, possiamo dire che si tratta di una descrizione *dichiarativa* della macchina astratta del linguaggio.

$$\langle X, \sigma\rangle \to \langle \sigma(X), \sigma\rangle$$

$$\langle (n+m), \sigma\rangle \to \langle p, \sigma\rangle \quad \text{dove } p = n + m \qquad \langle (n-m), \sigma\rangle \to \langle p, \sigma\rangle \quad \text{dove } p = n - m \text{ e } n \geq m$$

$$\frac{\langle a_1, \sigma\rangle \to \langle a', \sigma\rangle}{\langle (a_1 + a_2), \sigma\rangle \to \langle (a' + a_2), \sigma\rangle} \qquad \frac{\langle a_2, \sigma\rangle \to \langle a'', \sigma\rangle}{\langle (a_1 + a_2), \sigma\rangle \to \langle (a_1 + a''), \sigma\rangle}$$

$$\frac{\langle a_1, \sigma\rangle \to \langle a', \sigma\rangle}{\langle (a_1 - a_2), \sigma\rangle \to \langle (a' - a_2), \sigma\rangle} \qquad \frac{\langle a_2, \sigma\rangle \to \langle a'', \sigma\rangle}{\langle (a_1 - a_2), \sigma\rangle \to \langle (a_1 - a''), \sigma\rangle}$$

Figura 16.2 Semantica delle espressioni aritmetiche.

che sta a significare che se la condizione booleana è vera allora deve essere eseguito il comando del ramo **then**. Alcune transizioni, infine, sono *condizionali*: se un certo comando c_1 ha una certa transizione, allora il comando c ha un'altra transizione. Le transizioni condizionali assumono la forma di una *regola*, espressa in forma frazionaria

$$\frac{\langle c_1, \sigma_1\rangle \to \langle c_1', \sigma_1'\rangle \quad \langle c_2, \sigma_2\rangle \to \langle c_2', \sigma_2'\rangle}{\langle c, \sigma\rangle \to \langle c', \sigma'\rangle}$$

Leggiamo questa regola nel modo seguente: se il comando c_1, partendo dallo stato σ_1, può fare un passo di computazione e trasformarsi nel comando c_1' nello stato σ_1', e se il comando c_2, partendo da σ_2, può fare un passo di computazione e trasformarsi nel comando c_2' nello stato σ_2', allora il comando c, partendo dallo stato σ, può fare un passo di computazione e trasformarsi nel comando c' nello stato σ'. È evidente che una specifica regola esprimerà alcune relazioni significative tra c, c_1 e c_2 (e gli stati relativi): in genere c_1 e c_2 saranno sottocomandi di c[5].

Semantica delle espressioni La Figura 16.2 riporta le regole per la semantica delle espressioni aritmetiche. Le prime tre regole sono regole terminali (cioè la configurazione a destra della freccia è in una forma alla quale non si possono applicare altre transizioni): nell'ordine esse definiscono la semantica di una variabile, della somma nel caso in cui entrambi gli addendi siano costanti, della sottrazione nel caso entrambi gli addendi siano costanti e il risultato sia un numero naturale. Si osservi che non viene data alcuna semantica ad espressioni quali "$5 - 7$". Il secondo gruppo di quattro regole è costituito da regole condizionali. La prima coppia definisce la semantica della somma ed esprime il fatto che per

[5] Al lettore attento non sfuggirà che quelle che abbiamo chiamato regole condizionali, sono in realtà regole induttive della definizione della relazione di transizione.

$$\langle (n == m), \sigma \rangle \to \langle \mathbf{tt}, \sigma \rangle \quad \text{se } n = m \qquad \langle (n == m), \sigma \rangle \to \langle \mathbf{ff}, \sigma \rangle \quad \text{se } n \neq m$$

$$\langle (bv_1 \wedge bv_2), \sigma \rangle \to \langle bv, \sigma \rangle \quad \text{dove } bv \text{ è l'}\textit{and}\text{ di } bv_1 \text{ e } bv_2$$

$$\langle \neg \mathbf{tt}, \sigma \rangle \to \langle \mathbf{ff}, \sigma \rangle \qquad \langle \neg \mathbf{ff}, \sigma \rangle \to \langle \mathbf{tt}, \sigma \rangle$$

$$\frac{\langle a_1, \sigma \rangle \to \langle a', \sigma \rangle}{\langle (a_1 == a_2), \sigma \rangle \to \langle (a' == a_2), \sigma \rangle} \qquad \frac{\langle a_2, \sigma \rangle \to \langle a'', \sigma \rangle}{\langle (a_1 == a_2), \sigma \rangle \to \langle (a_1 == a''), \sigma \rangle}$$

$$\frac{\langle b_1, \sigma \rangle \to \langle b', \sigma \rangle}{\langle (b_1 \wedge b_2), \sigma \rangle \to \langle (b' \wedge b_2), \sigma \rangle} \qquad \frac{\langle b_2, \sigma \rangle \to \langle b'', \sigma \rangle}{\langle (b_1 \wedge b_2), \sigma \rangle \to \langle (b_1 \wedge b''), \sigma \rangle}$$

$$\frac{\langle b, \sigma \rangle \to \langle b', \sigma \rangle}{\langle \neg b, \sigma \rangle \to \langle \neg b', \sigma \rangle}$$

Figura 16.3 Semantica delle espressioni logiche.

calcolare il valore di una somma, occorre valutare prima, separatamente, i due addendi. Si osservi come la semantica *non specifichi alcun ordine di valutazione tra gli addendi* di una somma. La sottrazione è trattata in modo analogo.

La Figura 16.3 è relativa alla semantica delle espressioni logiche e segue la stessa idea delle espressione aritmetiche. Si noti che in tale figura *bv* denota un valore booleano (**tt** o **ff**).

Semantica dei comandi Si osservi come lo stato σ rimanga sempre inalterato durante la valutazione di un'espressione, e come esso sia utilizzato solo per ottenere il valore di una variabile. Una situazione destinata a cambiare nella semantica dei comandi, descritta nella Figura 16.4. Si osservi, nella figura, come le regole siano sostanzialmente di due tipi: quelle che, per ogni comando, esprimono il passo di computazione vero e proprio relativo a quel comando (si tratta delle regole $(c1), (c2), (c4), (c6), (c7)$ e $(c9)$) e le altre, che servono "solo" a far progredire la computazione di un sottocomando (o di una sottoespressione). La descrizione semantica del nostro linguaggio è ora completa.

Computazioni Una *computazione* è una sequenza di transizioni che non può essere ulteriormente estesa con un'altra transizione. Inoltre, ogni transizione di una computazione deve essere permessa da qualche regola.

Esempio 16.2 Consideriamo il programma c seguente

$$X := 1;\ \textbf{while}\ \neg(X == 0)\ \textbf{do}\ X := (X - 1)$$

$$\langle \textbf{skip}, \sigma \rangle \to \sigma \quad (c1)$$

$$\langle X := n, \sigma \rangle \to \sigma[X \leftarrow n] \quad (c2) \qquad \frac{\langle a, \sigma \rangle \to \langle a', \sigma \rangle}{\langle X := a, \sigma \rangle \to \langle X := a', \sigma \rangle} \quad (c3)$$

$$\frac{\langle c_1, \sigma \rangle \to \sigma'}{\langle c_1; c_2, \sigma \rangle \to \langle c_2, \sigma' \rangle} \quad (c4) \qquad \frac{\langle c_1, \sigma \rangle \to \langle c_1', \sigma' \rangle}{\langle c_1; c_2, \sigma \rangle \to \langle c_1'; c_2, \sigma' \rangle} \quad (c5)$$

$$\langle \textbf{if } \mathtt{tt} \textbf{ then } c_1 \textbf{ else } c_2, \sigma \rangle \to \langle c_1, \sigma \rangle \quad (c6) \qquad \langle \textbf{if ff then } c_1 \textbf{ else } c_2, \sigma \rangle \to \langle c_2, \sigma \rangle \quad (c7)$$

$$\frac{\langle b, \sigma \rangle \to \langle b', \sigma \rangle}{\langle \textbf{if } b \textbf{ then } c_1 \textbf{ else } c_2, \sigma \rangle \to \langle \textbf{if } b' \textbf{ then } c_1 \textbf{ else } c_2, \sigma \rangle} \quad (c8)$$

$$\langle \textbf{while } b \textbf{ do } c, \sigma \rangle \to \langle \textbf{if } b \textbf{ then } c; \textbf{while } b \textbf{ do } c \textbf{ else skip}, \sigma \rangle \quad (c9)$$

Figura 16.4 Semantica dei comandi.

e fissiamo uno stato che comprenda tutte le variabili menzionate nel programma, per esempio $\sigma = [(X, 6)]$. Possiamo calcolare la computazione di c in σ come segue. Per abbreviare la notazione, indichiamo con c' il comando iterativo **while** $\neg(X == 0)$ **do** $X := (X - 1)$. Non è difficile vedere che la computazione generata da c è la seguente:

$$\begin{aligned}
&\langle c, \sigma \rangle \\
&\to \langle c', \sigma[X \leftarrow 1] \rangle \\
&\to \langle \textbf{if } \neg(X == 0) \textbf{ then } X := (X - 1); c' \textbf{ else skip}, \sigma[X \leftarrow 1] \rangle \\
&\to \langle \textbf{if } \neg(1 == 0) \textbf{ then } X := (X - 1); c' \textbf{ else skip}, \sigma[X \leftarrow 1] \rangle \\
&\to \langle \textbf{if } \neg \textbf{ff} \textbf{ then } X := (X - 1); c' \textbf{ else skip}, \sigma[X \leftarrow 1] \rangle \\
&\to \langle \textbf{if } \mathtt{tt} \textbf{ then } X := (X - 1); c' \textbf{ else skip}, \sigma[X \leftarrow 1] \rangle \\
&\to \langle X := (X - 1); c', \sigma[X \leftarrow 1] \rangle \\
&\to \langle X := (1 - 1); c', \sigma[X \leftarrow 1] \rangle \\
&\to \langle X := 0; c', \sigma[X \leftarrow 1] \rangle \\
&\to \langle c', \sigma[X \leftarrow 0] \rangle \\
&\to \langle \textbf{if } \neg(X == 0) \textbf{ then } X := (X - 1); c' \textbf{ else skip}, \sigma[X \leftarrow 0] \rangle \\
&\to \langle \textbf{if } \neg(0 == 0) \textbf{ then } X := (X - 1); c' \textbf{ else skip}, \sigma[X \leftarrow 0] \rangle \\
&\to \langle \textbf{if } \neg \mathtt{tt} \textbf{ then } X := (X - 1); c' \textbf{ else skip}, \sigma[X \leftarrow 0] \rangle \\
&\to \langle \textbf{if ff then } X := (X - 1); c' \textbf{ else skip}, \sigma[X \leftarrow 0] \rangle \\
&\to \langle \textbf{skip}, \sigma[X \leftarrow 0] \rangle \\
&\to \sigma[X \leftarrow 0]
\end{aligned}$$

La computazione dell'esempio appena discusso è una computazione *terminata*, nel senso che, dopo un certo numero di transizioni si è raggiunta una situazione nella quale nessun'altra transizione è possibile. Si osservi, tuttavia, che la definizione di computazione che abbiamo dato poc'anzi *non* richiede che una

computazione sia finita, ma solo che "non possa essere estesa". Vi è dunque la possibilità di computazioni infinite, come il seguente esempio mette in luce.

Esempio 16.3 Consideriamo il programma d seguente

$$X := 1;\ \textbf{while}\ (X == 1)\ \textbf{do}\ \ \textbf{skip}$$

e lo stato $\tau = [(X, 0)]$; sia d' il comando **while** $(X == 1)$ **do skip** . Si ha:

$$\begin{array}{l}
\langle d, \tau\rangle \\
\rightarrow \langle d', \tau[X \leftarrow 1]\rangle \\
\rightarrow \langle \textbf{if}\ (X == 1)\ \textbf{then}\ \ \textbf{skip}\ ; d'\ \textbf{else skip}, \tau[X \leftarrow 1]\rangle \\
\rightarrow \langle \textbf{if}\ (1 == 1)\ \textbf{then}\ \ \textbf{skip}\ ; d'\ \textbf{else skip}, \tau[X \leftarrow 1]\rangle \\
\rightarrow \langle \textbf{if}\ \mathtt{tt}\ \textbf{then}\ \ \textbf{skip}\ ; d'\ \textbf{else skip}, \tau[X \leftarrow 1]\rangle \\
\rightarrow \langle\ \textbf{skip}\ ; d', \tau[X \leftarrow 1]\rangle \\
\rightarrow \langle d', \tau[X \leftarrow 1]\rangle \\
\rightarrow \ldots
\end{array}$$

Esistono dunque due tipi di computazioni fondamentalmente diverse: quelle *finite* (dette anche terminanti), e quelle *divergenti*, cioè infinite (e corrispondenti in via intuitiva ad un comando in ciclo).