Breve introduzione alla Semantica

- Per fornire la semantica formale a un linguaggio di programmazione serve un modello matematico, come base per comprendere e ragionare su come si comportano i programmi.
- Utile per vari tipi di analisi e verifiche, ma anche perché definire con precisione il significato delle costruzioni di programmi ci rende consapevoli di molti tipi di sottigliezze.
- Storicamente esistono tre approcci, non in opposizione tra loro: quelli della semantica operazionale, della semantica denotazionale e della semantica assiomatica.

Breve introduzione alla Semantica

- La **semantica operazionale** descrive il significato di un linguaggio di programmazione specificando come viene eseguito su una macchina astratta [Gordon, Plotkin]. Utile per l'implementazione. Approccio più concreto.
- La **semantica denotazionale** è una tecnica per definire il significato dei linguaggi di programmazione che utilizza i concetti matematici più astratti di ordini parziali completi, funzioni continue e punti fissi [Christopher Strachey, Dana Scott]. Approccio più astratto.
- La **semantica assiomatica** cerca di fissare il significato di un costrutto di programmazione fornendo regole di dimostrazione all'interno di una logica di programma. [R.W.Floyd, C.A.R.Hoare] Approccio adatto alla verifica di proprietà.

Breve introduzione alla Semantica Operazionale*

Quello che abbiamo visto può essere visto come un piccolo linguaggio **imperativo**, come quello che Winskel chiama **IMP**.

- Il comportamento di **IMP** viene descritto formalmente da un insieme di regole che specificano come valutare le espressioni e come eseguire i comandi.
- Le regole danno una *semantica operazionale* al linguaggio, operazionale proprio perché vicina all'implementazione del linguaggio. Inoltre offrono la base per semplici dimostrazioni di equivalenze tra comandi.
- La semantica operazionale, che fornisce un modello matematico, l'**astrazione**, dell'esecuzione di un interprete, viene usata per scrivere compilatori e interpreti ci descrive l'effetto di ogni comando sullo stato.

* Dal Capitolo 2 di "The Formal Semantics of Programming Languages: An Introduction" di Glynn Winskel. - Edizione italiana: "La semantica formale dei linguaggi di programmazione" di G. Winskel, F. Turini, P. Baldan e A. Bracciali

Categorie sintattiche

Cominciamo con l'introduzione delle categorie sintattiche del nostro linguaggio:

- numeri $\mathbf{N} \ni n$
- valori di verità $\mathbf{T} = \{\mathbf{true}, \mathbf{false}\}$
- locazioni $\mathbf{Loc} \ni X$ (*variabili* che possono essere modificate)
- espressioni aritmetiche $\mathbf{Aexp} \ni a$
- espressioni booleane $\mathbf{Bexp} \ni b$
- istruzioni o comandi $\mathbf{Com} \ni c$
- L'insieme degli *stati* Σ è dato dalle funzioni $\sigma : \mathbf{Loc} \to \mathbf{N}$, t.c. $\sigma(X)$ rappresenta il valore o il contenuto della locazione X nello stato σ.
- Adesso vedremo come costruire gli elementi di questi insiemi.

Categorie sintattiche, cont.

Per quanto riguarda la definizione delle ultime tre categorie sintattiche (**Aexp**,**Bexp**, e **Com**) daremo una *definizione induttiva*, ricorrendo ad apposite regole di formazione, con le quali esprimeremo cose del tipo seguente.

- Se a_0 e a_1 sono espressioni aritmetiche, lo è anche l'espressione composta $a_0 + a_1$.
- Analogamente, se b_0 e b_1 sono espressioni booleane, lo è anche l'espressione composta $b_0 \wedge b_1$.
- Analogamente, se c_0 e c_1 sono comandi, lo è anche il comando composto $c_0; c_1$.
- I simboli a_i, b_i e c_i sono usati per rappresentare una qualsiasi espressione aritmetica (espressione booleana, o comando). Più precisamente, si tratta di *metavariabili* che variano all'interno degli insiemi **Aexp**,**Bexp**, e **Com**.
- Diamo ora le regole di formazione delle espressioni (usando una notazione il più vicina possibile a quella già vista per il C)

Espressioni Aritmetiche

$$a ::= n | X | a_0 + a_1 | a_0 - a_1 | a_0 \times a_1$$

dove " :: =" deve essere letto come "può essere" e il simbolo "|" va inteso come "oppure".

- La regola $a ::= n$ ci dice che il numero $n \in \mathbf{N}$ è considerato un'espressione aritmetica (elementare).
- La regola $a ::= X$ ci dice che la variabile $X \in \mathbf{Loc}$ è considerata un'espressione aritmetica (elementare).
- La regola $a ::= a_0 + a_1$ ci dice che se a_0 e a_1 sono espressioni aritmetiche, allora lo è anche $a_0 + a_1$, dove le a_i sono meta-variabili. Ad es. se $a_0 = X$ and $a_1 = 5$ allora $X + 5$ è ancora un'espressione aritmetica.
- Le altre regole vanno interpretate in modo analogo.
- Per indicare che due espressioni sono equivalenti usiamo il simbolo $\equiv$.

Espressioni Booleane

$$b ::= \textbf{true} | \textbf{false} | a_0 == a_1 | a_0 <= a_1 | \neg b | b_0 \vee b_1 | b_0 \wedge b_1$$

- La regola $b ::= \textbf{true}$ ci dice che la costante **true** è considerata un'espressione booleana (elementare), analogamente per **false**.
- La regola $b ::= a_0 == a_1$ ci dice che se a_0 e a_1 sono espressioni aritmetiche, allora $a_0 == a_1$ è un'espressione booleana, analogamente per $a_0 <= a_1$.
- La regola $b ::= \neg b$ ci dice che se b è un'espressione booleana, lo è anche $\neg b$.
- La regola $b ::= b_0 \vee b_1$ ci dice che se b_0 e b_1 sono espressioni booleane, allora lo è anche $b_0 \vee b_1$
- Le altre regole vanno interpretate in modo analogo.
- Per indicare che due espressioni sono equivalenti usiamo il simbolo $\equiv$.

Comandi

$$c ::= \textbf{skip} | X = a | c_0; c_1 | \textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1 | \textbf{while } b \textbf{ do } c$$

- La regola $c ::= \textbf{skip}$ ci dice che **skip** è considerato un comando (elementare).
- La regola $c ::= X = a$ ci dice che se X è una variabile, ed a è un'espressione aritmetica, allora l'assegnamento $X = a$ è un comando.
- La regola $c ::= c_0; c_1$ ci dice che se c_0 e c_1 sono comandi, lo è anche il comando composto $c_0; c_1$.
- La regola $c ::= \textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1$ ci dice che se b è un'espressione booleana e c_0 e c_1 sono comandi, lo è anche il comando composto $\textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1$.
- La regola $c ::= \textbf{while } b \textbf{ do } c$ ci dice che se b è un'espressione booleana e c è un comando, lo è anche il comando composto $\textbf{while } b \textbf{ do } c$.
- Per indicare che due espressioni sono equivalenti usiamo il simbolo $\equiv$.

Valutazione delle Espressioni Aritmetiche

- Il significato di un'espressione **dipende** dal valore delle variabili, ovvero da quello che chiamiamo **stato**
- L'insieme degli stati Σ consiste nelle funzioni $\sigma : \mathbf{Loc} \to \mathbf{N}$ da locazioni a numeri, per cui $\sigma(X)$ rappresenta il valore associato a X, ovvero il contenuto della locazione X nello stato σ (nella nostra notazione corrisponde all'associazione $X \rightsquigarrow \sigma(X)$)
- La valutazione di un'espressione viene fatta dunque in un certo stato σ: abbiamo cioè *configurazioni* (coppie) della forma $\langle a, \sigma \rangle$ in attesa di essere valutate, arrivando al valore n.

$$\langle a, \sigma \rangle \to n$$

  Vedremo un insieme di regole da applicare per riscrivere le configurazioni fino a quando non raggiungiamo il valore finale n.
- La semantica operazionale si dice **strutturata**, quando usa la sintassi per guidare il processo di valutazione.

Regole di inferenza logica

- Regola di inferenza $\frac{\mathbf{A_1}, ..., \mathbf{A_n}}{\mathbf{B}}$: se sono vere le affermazioni sopra la riga (premesse), allora è vera l'affermazione sotto la riga (conclusione). Equivale a $A_1 \wedge ... \wedge A_n \Rightarrow B$
- $$\frac{\text{piove} \ \wedge \ \text{non ho l'ombrello}}{\text{mi bagno}}$$
- Un'inferenza è logicamente valida (le premesse implicano logicamente la conclusione) quando è impossibile che la conclusione sia falsa se le premesse sono vere. Altrimenti non è valida.
- $A \Rightarrow B$ è sempre vero, tranne nel caso in cui A è vero e B è falso.
- $A \Rightarrow B$ equivale a $\neg A \vee B$.

Regole di inferenza logica (cont.)

- Regola di inferenza $\frac{\mathbf{A_1}, ..., \mathbf{A_n}}{\mathbf{B}}$: se sono vere le affermazioni sopra la riga (premesse), allora è vera l'affermazione sotto la riga (conclusione). Equivale a $A_1 \wedge ... \wedge A_n \Rightarrow B$
- Assioma $\frac{-}{\mathbf{B}}$: regola di inferenza senza premesse, B quindi la conclusione è sempre vera
- Una derivazione di B prende la forma di un albero (detto di dimostrazione o derivazione) che può essere una istanza di un assioma $\frac{-}{\mathbf{B}}$ o includere le derivazioni delle premesse di B: $\frac{\frac{\mathbf{D_1}, ..., \mathbf{D_n}}{\mathbf{A_1}}, ..., \frac{\mathbf{E_1}, ..., \mathbf{E_n}}{\mathbf{A_n}}}{\mathbf{B}}$: la radice è l'asserzione da dimostrare, le foglie sono assiomi e i nodi intermedi sono ottenuti applicando regole di inferenza

Valutazione delle Espressioni Aritmetiche (cont.)

- La valutazione di un'espressione non richiede di modificare la memoria.
- $\langle n, \sigma \rangle \to n$ **assioma** (premessa vuota)
- $\langle x, \sigma \rangle \to \sigma(x)$
- $$\frac{\langle a_0, \sigma \rangle \to n_0 \quad \langle a_1, \sigma \rangle \to n_1}{\langle a_0 + a_1, \sigma \rangle \to n_0 + n_1}$$ **premessa** / **conclusione**
- $$\frac{\langle a_0, \sigma \rangle \to n_0 \quad \langle a_1, \sigma \rangle \to n_1}{\langle a_0 - a_1, \sigma \rangle \to n_0 - n_1}$$
- $$\frac{\langle a_0, \sigma \rangle \to n_0 \quad \langle a_1, \sigma \rangle \to n_1}{\langle a_0 \times a_1, \sigma \rangle \to n_0 \times n_1}$$
- N.B. Gli operatori in nero ($+, -, \times$) rappresentano simboli della sintassi, mentre i corrispondenti operatori in blu l'effettiva implementazione dell'operazione. Ad es. $n_0 + n_1$ sta per la somma di n_0 e di n_1.

Valutazione delle Espressioni Aritmetiche (esempio)

Consideriamo la valutazione dell'espressione aritmetica
$a_0 = (X + 5) + (7 + 9)$ nello stato σ_0, nel quale $\sigma_0(X) = 0$

$$\dfrac{\dfrac{\dfrac{}{\langle X, \sigma_0\rangle \to 0} \quad \dfrac{}{\langle 5, \sigma_0\rangle \to 5}}{\langle (X + 5), \sigma_0\rangle \to 5} \quad \dfrac{\dfrac{}{\langle 7, \sigma_0\rangle \to 7} \quad \dfrac{}{\langle 9, \sigma_0\rangle \to 9}}{\langle (7 + 9), \sigma_0\rangle \to 16}}{\langle (X + 5) + (7 + 9), \sigma_0\rangle \to 21}$$

La valutazione di $\langle a_0, \sigma_0\rangle$ dipende da un albero di valutazioni.

Valutazione delle Espressioni Booleane

- $\langle \mathbf{true}, \sigma \rangle \to \mathbf{true} \qquad \langle \mathbf{false}, \sigma \rangle \to \mathbf{false}$
- $$\frac{\langle a_0, \sigma \rangle \to n_0 \quad \langle a_1, \sigma \rangle \to n_1}{\langle a_0 == a_1, \sigma \rangle \to \mathbf{true}/\mathbf{false}} \quad \text{se } n_0 = n_1 / n_0 \neq n_1$$
- $$\frac{\langle a_0, \sigma \rangle \to n_0 \quad \langle a_1, \sigma \rangle \to n_1}{\langle a_0 <= a_1, \sigma \rangle \to \mathbf{true}/\mathbf{false}} \quad \text{se } n_0 \leq n_1 / n_0 \not\leq n_1$$
- $$\frac{\langle b, \sigma \rangle \to \mathbf{true}}{\langle \neg b, \sigma \rangle \to \mathbf{false}} \qquad \frac{\langle b, \sigma \rangle \to \mathbf{false}}{\langle \neg b, \sigma \rangle \to \mathbf{true}}$$
- $$\frac{\langle b_0, \sigma \rangle \to t_0 \quad \langle b_1, \sigma \rangle \to t_1}{\langle b_0 \vee b_1, \sigma \rangle \to t} \quad \text{con } t \text{ a seconda dei valori } t_0 \text{ e } t_1$$
- $$\frac{\langle b_0, \sigma \rangle \to t_0 \quad \langle b_1, \sigma \rangle \to t_1}{\langle b_0 \wedge b_1, \sigma \rangle \to t} \quad \text{con } t \text{ a seconda dei valori } t_0 \text{ e } t_1$$