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. La semantica operazionale fornisce l'**astrazione** dell'esecuzione di un interprete.

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**), ricorreremo 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 lo pseudo-codice)

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 è considerato un'espressione aritmetica (elementare).
- La regola $a ::= X$ ci dice che la variabile X 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.
- 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 **if** b **then** c_0 **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 **while** b **do** c.

Per indicare che due comandi sono equivalenti usiamo il simbolo $\equiv$.
Stiamo dando una definizione *induttiva* delle categorie introdotte.

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 σ : **Loc** $\rightarrow$ **N** da locazioni a numeri, per cui $\sigma(X)$ rappresenta il valore associato a X, ovvero il contenuto della locazione X nello stato σ
- La valutazione di un'espressione viene fatta dunque in un certo stato σ: abbiamo cioè coppie della forma $\langle a, \sigma \rangle$ in attesa di essere valutate, arrivando al valore n.

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

La semantica operazionale si dice **strutturata**, quando usa la sintassi per guidare il processo di valutazone.

Valutazione delle Espressioni Aritmetiche (cont.)

- $\langle n, \sigma \rangle \rightarrow n$
- $\langle x, \sigma \rangle \rightarrow \sigma(x)$
- $$\frac{\langle a_0, \sigma \rangle \rightarrow n_0 \quad \langle a_1, \sigma \rangle \rightarrow n_1 \quad \textbf{premessa}}{\langle a_0 + a_1, \sigma \rangle \rightarrow n_0 + n_1 \quad \textbf{conclusione}}$$
- $$\frac{\langle a_0, \sigma \rangle \rightarrow n_0 \quad \langle a_1, \sigma \rangle \rightarrow n_1}{\langle a_0 - a_1, \sigma \rangle \rightarrow n_0 - n_1}$$
- $$\frac{\langle a_0, \sigma \rangle \rightarrow n_0 \quad \langle a_1, \sigma \rangle \rightarrow n_1}{\langle a_0 \times a_1, \sigma \rangle \rightarrow n_0 \times n_1}$$

Nota che gli operatori in nero ($+$.-, $\times$) rappresentano simboli della sintassi, mentre i corrispondenti operatori in blu stanno per l'effettiva implementazione dell'operazione. Quindi, ad esempio $n_0 + n_1$ sta per la somma di n_0 e di n_1.

Valutazione delle Espressioni Booleane

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

Valutazione "lazy" di $\wedge$

- In C, la valutazione della congiunzione *and* è chiamata *lazy* o pigra, dato che inizialmente si rimanda la valutazione della seconda espressione e poi la si fa solo se necessario.
- In particolare, prima si controlla se la prima espressione è falsa, nel qual caso si evita di valutare la seconda parte dell'espressione, dato che la valutazione complessiva sarà comunque falsa.
- Questo tipo di valutazione può essere reso con le regole della semantica operazionale seguenti:

$$\frac{\langle b_0, \sigma \rangle \rightarrow \mathbf{false}}{\langle b_0 \wedge b_1, \sigma \rangle \rightarrow \mathbf{false}} \qquad \frac{\langle b_0, \sigma \rangle \rightarrow \mathbf{true} \quad \langle b_1, \sigma \rangle \rightarrow t}{\langle b_0 \wedge b_1, \sigma \rangle \rightarrow t}$$

Esecuzione dei comandi

L'esecuzione di un comando porta ad un nuovo stato: $\langle c, \sigma\rangle \to \sigma'$, ammesso che l'esecuzione termini!
Si suppone che lo *stato inziale* σ_0 abbia la proprietà che $\sigma_0(X) = 0$ per tutte le locazioni X.

- $\langle \textbf{skip}, \sigma\rangle \to \sigma$
- $\dfrac{\langle a, \sigma\rangle \to m}{\langle X = a, \sigma\rangle \to \sigma[m/X]}$ $\sigma[m/X]$ è lo stato dove m è associato a X^*
- $\dfrac{\langle c_0, \sigma\rangle \to \sigma'' \quad \langle c_1, \sigma''\rangle \to \sigma'}{\langle c_0; c_1, \sigma\rangle \to \sigma'}$ prima va eseguito il primo comando
- $\dfrac{\langle b, \sigma\rangle \to \textbf{true}/\textbf{false} \quad \langle c_i, \sigma\rangle \to \sigma_i}{\langle \textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1, \sigma\rangle \to \sigma_0/\sigma_1}$

$$^*\sigma[m/x](y) = \begin{cases} m & \text{if } y = x; \\ \sigma(y) & \text{if } y \neq x. \end{cases}$$

Valutazione dei comandi (cont.)

- $\dfrac{\langle b, \sigma\rangle \to \textbf{false}}{\langle \textbf{while } b \textbf{ do } c, \sigma\rangle \to \sigma}$
- $\dfrac{\langle b, \sigma\rangle \to \textbf{true} \langle c; \textbf{while } b \textbf{ do } c, \sigma\rangle \to \sigma'}{\langle \textbf{while } b \textbf{ do } c, \sigma\rangle \to \sigma'}$

Completezza e Correttezza

Vorremmo che la nostra semantica ci permettesse di dimostrare tutto ciò che è vero e niente di falso. La semantica è

- **completa** se può dimostrare ogni *asserzione* vera. Ad es., con la regola seguente, *non* potrei dimostrare che $\langle 5-3,\sigma\rangle \rightarrow 2$ (vero).

$$\frac{\langle a_0,\sigma\rangle \rightarrow n_0 \quad \langle a_1,\sigma\rangle \rightarrow 0}{\langle a_0 - a_1,\sigma\rangle \rightarrow n_0}$$

- **corretta** se ogni *asserzione* dimostrabile è vera. Ad es., con la regola seguente, *potrei* dimostrare che $\langle 5-3,\sigma\rangle \rightarrow 7$ (falso)

$$\frac{\langle a_0,\sigma\rangle \rightarrow n_0 \quad \langle a_1,\sigma\rangle \rightarrow n_1}{\langle a_0 - a_1,\sigma\rangle \rightarrow n_0+2}$$

Equivalenze

- Due espressioni aritmetiche a_0 e a_1 sono *equivalenti* $a_0 \sim a_1$ se e solo se

$$\forall n \in \mathbf{N}, \forall \sigma \in \Sigma. \langle a_0,\sigma\rangle \rightarrow n \Leftrightarrow \langle a_1,\sigma\rangle \rightarrow n$$

- Due espressioni booleane b_0 e b_1 sono *equivalenti* $b_0 \sim b_1$ se e solo se

$$\forall t, \forall \sigma \in \Sigma. \langle b_0,\sigma\rangle \rightarrow t \Leftrightarrow \langle b_1,\sigma\rangle \rightarrow t$$

- Due comandi c_0 e c_1 sono *equivalenti* $c_0 \sim c_1$ se e solo se

$$\forall \sigma, \sigma' \in \Sigma. \langle c_0,\sigma\rangle \rightarrow \sigma' \Leftrightarrow \langle c_1,\sigma\rangle \rightarrow \sigma'$$

Es. di dimostrazione

Proposizione
Sia $w \equiv$ **while** b **do** c. Allora

$$w \sim \textbf{if } b \textbf{ then } c; w \textbf{ else skip}$$

Dimostrazione
Si deve dimostrare che per ogni scelta di σ, σ':

$$\langle w, \sigma \rangle \to \sigma' \text{ sse } \langle \textbf{if } b \textbf{ then } c; w \textbf{ else skip}, \sigma \rangle \to \sigma'$$

Lo si fa usando le derivazioni delle regole

Dimostrazione $\Leftarrow$

Siano $\sigma, \sigma' \in \Sigma$ e si supponga che $\langle \textbf{if } b \textbf{ then } c; w \textbf{ else skip}, \sigma \rangle \to \sigma'$.
Allora deve esistere una derivazione con una delle seguenti forme

- $$\dfrac{\dfrac{\dots}{\langle b, \sigma \rangle \to \textbf{false}} \quad \dfrac{}{\langle \textbf{skip}, \sigma \rangle \to \sigma}}{\langle \textbf{if } b \textbf{ then } c; w \textbf{ else skip}, \sigma \rangle \to \sigma} \quad (1)$$
- $$\dfrac{\dfrac{\dots}{\langle b, \sigma \rangle \to \textbf{true}} \quad \dfrac{}{\langle c; w, \sigma \rangle \to \sigma'}}{\langle \textbf{if } b \textbf{ then } c; w \textbf{ else skip}, \sigma \rangle \to \sigma'} \quad (2)$$

Da ognuna si può partire per costruire la derivazione di $\langle w, \sigma \rangle$.

Dimostrazione $\Leftarrow$ (caso (2))

La seconda derivazione deve contenere, per qualche stato σ'', qualcosa del tipo

$$\dfrac{\dfrac{\ldots}{\langle c, \sigma\rangle \to \sigma''} \quad \dfrac{\ldots}{\langle w, \sigma''\rangle \to \sigma'}}{\langle c; w, \sigma\rangle \to \sigma'}$$

Combinando con il resto:

$$\dfrac{\dfrac{\ldots}{\langle b, \sigma\rangle \to \textbf{true}} \quad \dfrac{\ldots}{\langle c, \sigma\rangle \to \sigma''} \quad \dfrac{\ldots}{\langle w, \sigma''\rangle \to \sigma'}}{\langle w, \sigma\rangle \to \sigma'}$$

Analogamente per il caso (1). Si può quindi concludere che

$$\langle \textbf{if } b \textbf{ then } c; w \textbf{ else skip}, \sigma\rangle \to \sigma' \Rightarrow \langle w, \sigma\rangle \to \sigma'$$