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 $\equiv$.