Aniello Murano

Semantica denotazionale di IMP

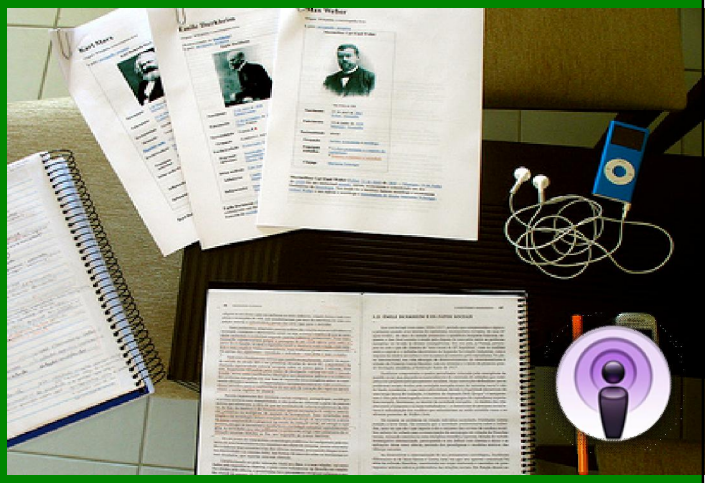
Lezione n. 7
Parole chiave:
Semantica denotazionale

Corso di Laurea:
Informatica

Codice:

Email Docente:
murano@ na.infn.it

A.A. 2008-2009



Introduzione alla VII lezione

- Nella prima parte del corso abbiamo introdotto il linguaggio imperativo IMP
- Per questo linguaggio, abbiamo introdotto una semantica operazionale e abbiamo mostrato che questa semantica si basa su regole di derivazione. In particolare:
  - Ø È legata alla sintassi
  - Ø Interpreta i programmi come relazioni di transizione.
- Sebbene **relativamente semplice** e molto efficiente per valutare certe proprietà di un linguaggio quali il **determinismo**, l'utilizzo di tali regole comporta alcune limitazioni:
  - Ø Mostra soltanto come possono essere utilizzati gli elementi di un linguaggio, ma non il oro significato intrinseco.
  - Ø La dipendenza dalla sintassi rende difficile il confronto di programmi scritti in linguaggi di programmazione differenti

Esempio

- Supponiamo di voler valutare l'equivalenza della funzione "quadrato di un numero" espressa nelle seguenti notazioni

Algol:
```
integer procedure square(x);
integer x;
begin square := x * x end;
```

Pascal:
```
function square (x:integer) :
integer;
begin square := x * x end;
```

StdC/C++/Java:
```
int square(int x)
{ return (x * x); }
```

ML97:
```
fun square x = x * x;
fun square (x:int) = x * x;
val square = fn x => x * x;
```

Teoria degli insiemi:
$\{(x,y) \mid 8\ x,y\ 2\ N : y = x^2\}$

Algebra
$f : N \to N;\ f(x) = x^2;$

- Utilizzando la sola semantica operazionale questa valutazione non è possibile

Un po' di storia

- Negli anni sessanta, Christopher Strachey e Dana Scott riuscirono a superare le limitazioni della semantica operazionale introducendo una semantica più astratta basata sull'utilizzo di **funzioni** semantiche (chiamate denotazioni) come modelli matematici di rappresentazione (del significato) dei programmi.
- L'idea di base utilizzata in questa semantica (denotazionale) può essere mostrata con un esempio:
- In IMP, due comandi c_0 e c_1 sono equivalenti (c_0 » c_1) sse

$$8\sigma,\sigma' . \langle c_0,\sigma\rangle \ ! \ \sigma' \ , \ \langle c_1,\sigma\rangle \ ! \ \sigma'$$

- In modo equivalente, possiamo dire che c_0 » c_1 sse

$$\{(\sigma,\sigma') \mid \langle c_0,\sigma\rangle \ ! \ \sigma'\} = \{(\sigma,\sigma') \mid \langle c_1,\sigma\rangle \ ! \ \sigma'\}$$

- Cioè sse c_0 e c_1 definiscono la stessa **funzione parziale** sugli stati. Dunque una valutazione di equivalenza tra linguaggi viene rimandata alla valutazione di equivalenza tra oggetti matematici

Semantica denotazionale

- L'obiettivo della semantica denotazionale è quello di esprimere il significato di un programma in termini di funzioni semantiche (funzioni matematiche).
- In pratica, ad ogni frase del linguaggio viene associata una denotazione (significato) come funzione delle denotazioni delle sue sottofrasi.
- La semantica denotazionale cerca di catturare il significato interno di un programma piuttosto che la strategia di implementazione. Per questo motivo è più astratta di quella operazionale ed è indipendente dalla macchina su cui si lavora.

Semantica denotazionale di IMP

- In generale, per definire la ***semantica*** di un oggetto di un linguaggio definito da una certa ***sintassi*** utilizzeremo la notazione « ¬ nel seguente modo:

  « <sintassi>¬ = denotazione di <sintassi>

- dove le parentesi « ¬ sono chiamate parentesi semantiche e « <sintassi>¬ va letto come la "denotazione di <sintassi>".
- Sostanzialmente le parentesi « ¬ sono una funzione matematica.
- Nelle prossime diapositive mostgriamo la semantica (Scott-Strackey) denotazionale di IMP

Semantica operazionale di Aexp

- Ricordiamo la sintassi delle espressioni aritmetiche Aexp di IMP

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

- La semantica denotazionale per a 2 Aexp è una funzione

  A«a¬ : Σ ! N o equivalentemente A : Aexp ! (Σ ! N)

- La denotazione di a (semantica denotazionale A«a¬ per a) è una relazione tra stati e numeri ed è definita per induzione sulla struttura dell'espressione nel modo seguente:
  - Ø A«n¬ = {(σ, n) | σ 2 Σ} A«X¬ = {(σ, σ(X)) | σ 2 Σ}
  - Ø $A«a_0 + a_1¬ = \{(\sigma, n_0 + n_1) \mid (\sigma, n_0)\ 2\ A«a_0¬\ \&\ (\sigma, n_1)\ 2\ A«a_1¬\}$
  - Ø $A«a_0 - a_1¬ = \{(\sigma, n_0 - n_1) \mid (\sigma, n_0)\ 2\ A«a_0¬\ \&\ (\sigma, n_1)\ 2\ A«a_1¬\}$
  - Ø $A«a_0 * a_1¬ = \{(\sigma, n_0 * n_1) \mid (\sigma, n_0)\ 2\ A«a_0¬\ \&\ (\sigma, n_1)\ 2\ A«a_1¬\}$

- Si noti che le funzioni "Σ ! N " sono rappresentate (in modo equivalente) in termini di insiemi di coppie "stato, numero" e che i simboli "+", "-" e "*" nella parte sinistra delle uguaglianze sono simboli sintattici e nella parte destra sono operatori

Esempi di valutazione di Aexp

- Per qualsiasi stato σ, la semantica denotazionale di "3+5" è

  A«3 + 5¬σ = A«3¬σ + A«5¬σ = 3+5 =8

- dove A«a¬σ semplifica A«a¬(σ).
- Sia σ lo stato in cui la locazione X vale "2". La semantica denotazionale di "X*3" è

  A«X*3¬σ = A«X¬σ * A«3¬σ = σ(X)*3 =6

- E se avessimo avuto anche la divisione (a_0/a_1) in Aexp, come può essere definita la semantica denotazionale per a_0/a_1?
- Bisogna tener presente quando il denominatore è zero:
  - Ø $A«a_0/a_1¬\sigma$ è indefinito se $A«a_1¬\sigma = 0$.
  - Ø Altrimenti è il quoziente intero della divisione tra $A«a_0¬\sigma$ e $A«a_1¬\sigma$

λ–calculus

- Un modo alternativo di rappresentare funzioni è dato dalla notazione λ
- In pratica, la notazione λx.e denota la funzione che mappa x in e.
- Per esempio la funzione successione può esser scritta come

  λx.x+1
- Utilizzando questa notazione, è possibile riscrivere la semantica delle espressioni aritmetiche di IMP nel seguente modo
  - A«n¬ = λσ.n
  - A«X¬ = λσ.σ(X)
  - A«a_0 + a_1¬ = {(σ, n) | σ 2 Σ Æ n = A«a_0¬σ + A«a_1¬σ}
  - A«a_0 - a_1¬ = {(σ, n) | σ 2 Σ Æ n = A«a_0¬σ - A«a_1¬σ}
  - A«a_0 * a_1¬ = {(σ, n) | σ 2 Σ Æ n = A«a_0¬σ * A«a_1¬σ}

Intermezzo: Background del λ-calculus

- Developed in 1930's by Alonzo Church.
- Subsequently studied (and still studied) by many people in logic and computer science.
- Considered the "testbed" for procedural and functional languages.
  - Simple.
  - Powerful.
  - Easy to extend with features of interest.
- "Whatever the next 700 languages turn out to be, they will surely be variants of lambda calculus." (Landin '66)

Intermezzo: Alonzo Church

- PhD from Princeton University, 1927
  - Advisor was Oswald Veblen who also advised R. L. Moore of UT Austin
- Studied with David Hilbert, Paul Bernays & L.E.J. Brouwer in Germany
- Many of his PhD students are "founding fathers" of theoretical computer science
  - Stephen Kleene, 1934 (recursive function theory)
  - J. Barkley Rosser, 1934 (Church-Rosser theorem)
  - Alan Turing, 1938 (computational logic & computability)
  - Martin Davis, 1950 (logic & computability theory)
  - J. Hartley Rogers, 1952 (recursive function theory)
  - Michael Rabin, 1956 (probabilistic algorithms – Turing award)
  - Dana Scott, 1958 (prob. algorithms, domain theory – Turing award)
  - Raymond Smullyan, 1959 (logic, tableau proof, formal systems)

Intermezzo
The λ-calculus in Computer Science

- A formal notation, theory, and model of computation
- Church's thesis: λ-calculus & Turing Machines describe the same set of objects, i.e., effectively computable functions
  - equivalence was proved by **Stephen Kleene**
- Foundation for the functional style of programming
  - Lisp, Scheme, ISWIM, ML, Miranda™, Haskell, …
- Notation for Scott-Strachey denotational semantics.
- It can be used to abstractly represent numbers, booleans, predicates, functions, variables, block scopes, expressions, ordered pairs, lists, records & recursion, ecc.

Denotazione di Bexp (1)

- Ricordiamo la sintassi delle espressioni booleane Bexp di IMP

  b ::= true | false | $a_0 = a_1$ | $a_0 \cdot a_1$ | : b| $b_0 \text{Æ} b_1$ | $b_0 \text{Ç} b_1$
- La semantica denotazionale per b 2 Bexp è una funzione

  B«b¬ : Σ ! {true, false}
- La funzione semantica per le espressioni booleane è definita in termini delle operazioni logiche di congiunzione (Æ), disgiunzione (Ç) e negazione (:)
- La denotazione di b 2 Bexp (semantica denotazionale A«b¬ per b) è una relazione tra stati e valori di verità, definita per induzione sulla struttura dell'espressione booleane nel modo seguente:

Denotazione di Bexp (2)

- B«true¬ = {(σ, true) | σ 2 Σ}
- B«false¬ = {(σ, false) | σ 2 Σ}
- B«$a_0 = a_1$¬ = {(σ, true) | σ 2 Σ & A«a_0¬σ = A«a_1¬σ} [{(σ, false) | σ 2 Σ & A«a_0¬σ ≠ A«a_1¬σ}
- B«$a_0 \cdot a_1$¬ = {(σ, true) | σ 2 Σ & A«a_0¬σ · A«a_1¬σ} [{(σ, false) | σ 2 Σ & A«a_0¬σ £ A«a_1¬σ}
- B«: b¬ = {(σ, : t) | σ 2 Σ & (σ, t) 2 B«b¬}
- B«b_0 Æ b_1¬ = {(σ, t_0 Æ t_1) | σ 2 Σ & (σ, t_0) 2 B«b_0¬ & (σ, t_1) 2 B«b_1¬}
- B«b_0 Ç b_1¬ = {(σ, t_0 Ç t_1) | σ 2 Σ & (σ, t_0) 2 B«b_0¬ & (σ, t_1) 2 B«b_1¬}
- Si noti che i simboli "true", "false", ":", "Æ" e "Ç" nella parte sinistra delle uguaglianze sono simboli sintattici mentre nella parte destra sono operatori. Inoltre, le funzioni "Σ ! {true, false}" sono state indicate come insiemi di coppie "stato, valore di verità"

Esempi di valutazione di Bexp

- Per ogni stato σ, la semantica denotazionale di "true Ç false" è
  B«true Ç false ¬σ = B«true¬σ Ç B«false¬σ = true Ç false = true
- dove B«b¬σ semplifica B«b¬(σ).
- Sia σ lo stato in cui la locazione X vale "2" e Y vale "3". La semantica denotazionale di "X+3 = Y+2" è
  B«X+3 = Y+2 ¬σ = true perché A«X+3¬σ = 5 = A«Y+2¬σ

Denotazione di Com (1)

- Ricordiamo la sintassi dei comandi Com di IMP
  c ::= skip | X:=a | c_0;c_1 | if b then c_0 else c_1 | while b do c
- La semantica denotazionale per c 2 Com è una funzione
  C«c¬ : Σ Σ
- La denotazione di c **2** Com (semantica denotazionale C«c¬ per c) è una relazione tra stati, definita per induzione sulla struttura dei comandi.
- Si noti che la funzione di valutazione è parziale C«c¬ : Σ Σ in quanto su alcuni comandi la funzione può non essere definita (per esempio sui loop infiniti)

Denotazione di Com (2)

- C«skip¬ = {(σ, σ) | σ 2 Σ}
- C«X:=a ¬ = {(σ, σ[n/X]) | σ 2 Σ & A«a_0¬σ = n }
- C«c_0;c_1¬ = C«c_1¬ ± C«c_0¬ (si noti l'inversione in accordo alla regola di composizione)
- C«if b then c_0 else c_1¬ = {(σ, σ') | B«b¬σ = true & (σ, σ') 2 C«c_0¬} [{(σ, σ') | B«b¬σ = false & (σ, σ') 2 C«c_1¬}
- Particolarmente difficile è invece la valutazione del comando while per il quale è necessario utilizzare i concetti matematici del punto fisso introdotti nella lezione precedente. Questa parte e' rimandata alla prossima lezione

Semantica Assiomatica

- Per motivi di tempo, in questo corso si è deciso di non trattare la semantica assiomatica
- La semantica assiomatica e' un approccio basato sulla logica matematica per provare la correttezza di programmi
- Una logica assiomatica consiste di:
  - Un linguaggio per fare asserzioni sui programmi
  - Regole per verificare le asserzioni.
- Alcuni tipi classici di asserzioni:
  - Questo programma termina.
  - Le variabili x e y hanno lo stesso valore durante l'esecuzione di un un programma ogniqualvolta z vale 0.
  - Tutti gli accessi ad un array sono all'interno dei bound dell'array.
- Alcuni linguaggi tipici per le asserzioni sono :
  - La logica del primo ordine.
  - Altre logiche (e.g., logiche temporali).

Applicazioni della semantica assiomatica

- Alcune applicazioni:
  - Ø Guida alla progettazione e alla realizzazione di software.
  - Ø Prova di correttezza di software (algoritmi) e hardware.
  - Ø Documentazione di programmi e interfacce
- L'obiettivo di definire e provare ogni cosa formalmente (circa i programmi) non e' ancora stato realizzato (almeno non ancora).
- Dijkstra una volta disse: " Il testing dei programmi puo' essere usato per mostrare la presenza di un errore, ma non per mostrare la sua assenza".
- Hoare disse: "Quindi la pratica di provare al correttezza di programmi sembrerebbe portare alla soluzione di tre dei piu' intricati problemi nella programmazione e cioè:, **reliability**, **documentazione** e **compatibilità**. Ad ogni modo, la verifica dei programmi, almeno per ora, sarà difficile anche per programmatori di alto calibro e potrà essere applicata solo alla progettazione di programmi piuttosto semplici".