Aniello Murano

Esercitazione sulla semantica operazionale di IMP e sulle tecniche di prova

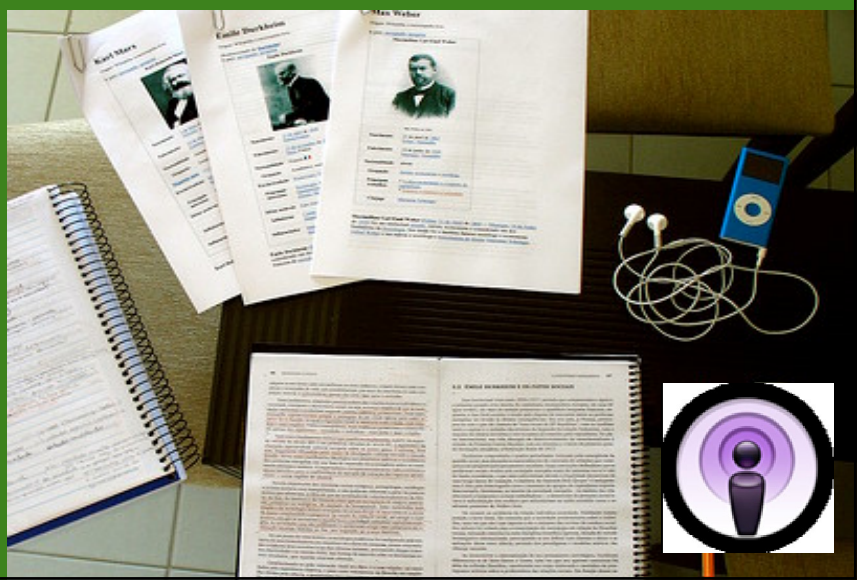
Lezione n.4
Parole chiave:
Esercitazione

Corso di Laurea:
Informatica

Codice:

Email Docente:
murano@ na.infn.it

A.A. 2008-2009



Argomenti di Esercitazione

- Seconda lezione: Sintassi e semantica operazionale del linguaggio imperativo IMP
  1. Utile per definire sintassi e semantica di nuovi operatori di un linguaggio
  2. Utile per provare l'equivalenza semantica di due elementi di un linguaggio
- Terza lezione: Tecniche di prova per induzione
  3. Utile per provare delle proprietà semantiche degli elementi di un linguaggio

1. Definizione di un nuovo operatore

- Supponiamo di voler estendere le espressioni aritmetiche di IMP includendo un nuovo operatore "^" con il significato di elevamento a potenza.
- La sintassi di Aexp di IMP verrà estesa includendo anche a_0^a_1 dove a_0 e a_1 sono elementi di Aexp. Dunque per Aexp avremo:
- a ::= n | X | a_0 + a_1 | a_0 - a_1 | a_0 x a_1 | a_0^ a_1.
- Valutazione del nuovo operatore

$$\frac{\langle a_0,\sigma\rangle \rightarrow n_0 \qquad \langle a_1,\sigma\rangle \rightarrow n_1}{\langle a_0 \hat{} a_1,\sigma\rangle \rightarrow n}$$

- dove n è uguale a n_0 elevato a n_1

2. Equivalenza semantica di due espressioni

- Utilizzando le regole di valutazione introdotte per le espressioni aritmetiche è possibile mostrare l'equivalenza di due espressioni
- Per esempio, è possibile mostrare formalmente che
- " y + y " equivale all'espressione " y * 2 ", cioè (y + y) equiv. (y * 2)
- Formalmente, si vuole provare che

  $\langle (y + y),\sigma\rangle \rightarrow m$ iff $\langle (y * 2),\sigma\rangle \rightarrow m$, per qualsiasi σ e m

"$\Rightarrow$" se $\langle (y + y), \sigma\rangle \rightarrow m$ allora deve esistere un albero di derivazione con y valutato m_0 e conclusione $\langle (y + y), \sigma\rangle \rightarrow m = m_0 + m_0$. Dunque m è due volte la valutazione di y. Allora possiamo definire un albero di derivazione per $\langle (y * 2), \sigma\rangle$ con conclusione $\langle (y * 2), \sigma\rangle \rightarrow m' = 2 \text{ x } m_0 = m_0 + m_0 = m$

"$\Leftarrow$" L'inverso del discorso precedente

2. Equivalenza semantica di due comandi

- Utilizzando le regole di valutazione introdotte per i comandi è possibile mostrare l'equivalenza di due comandi
- Per esempio, è possibile mostrare formalmente che

if y=3 then y:=y+1 else y:=y-1

equiv

if ¬(y=3) then y:=y-1 else y:=y+1

- Formalmente, si vuole provare che
  <if y=3 then y:=y+1 else y=y-1, σ> → σ′ iff <if ¬(y=3) then y:=y-1 else y:=y+1,σ> → σ′, per ogni scelta degli stati σ, σ′

"⇒" <if y=3 then y:=y+1 else y:=y-1, σ> → σ′ allora deve esistere un albero di derivazione in cui σ′ dipende dalla valutazione di y=3. Allora possiamo definire un albero di derivazione per <if ¬(y=3) then y:=y-1 else y:=y+1,σ> con conclusione <if ¬(y=3) then y:=y-1 else y:=y+1,σ> → σ′′ dove σ′′=σ′

"⇐" L'inverso del discorso precedente

3. $\rightarrow_{Bexp}$ è deterministica

- Usando l'induzione strutturale proviamo che
- P(b)=∀σ ϵ Loc e ∀ w,w′ ϵ {true,false}. <b,σ> → w & <b,σ> → w′ ⇒ w=w′
- Se **b** è **true/false**. Allora c'è una sola regola per true/false che si può applicare e dunque w=w′.
- Se **b** è $\mathbf{a_0 = a_1}$ oppure $\mathbf{a_0 \leq a_1}$ allora i valori w e w′ dipendono dalle regole di valutazione delle espressioni aritmetiche. Il risultato segue dal fatto che $\rightarrow_{Aexp}$ è deterministica.
- Se **b** è $\mathbf{b_0 \vee b_1}$ oppure $\mathbf{b_0 \wedge b_1}$ oppure $\neg b_0$, il risultato segue per ipotesi induttiva.

Esercizio

- Provare che $\rightarrow_{Bexp}$ è totale utilizzando il metodo di induzione strutturale
- Definire la semantica denotazionale del seguente comandi:
- Do c while b (con il significato classico in linguaggio C)
- Do c while b after c′ (con la variante al caso precedente che b viene valutato dopo aver eseguito c′ sullo stato ottenuto dall′esecuzione di c. Si noti che c′ viene eseguito solo per valutare b, ma nell′itgerazione c′ non ha effetto.)
- save (con il significato di fare un mirror di tutte le variavibli: per ogni variabile x, si usa x′ e si esegue x′:=x.
- restore (con il significato di eseguire per ogni variabile x il comando x = x′)
- Provare la seguente equivalenza:

do c1 while b after c2

equiv.

c1; save; c2; (while b do (restore; c1; save; c2)); restore