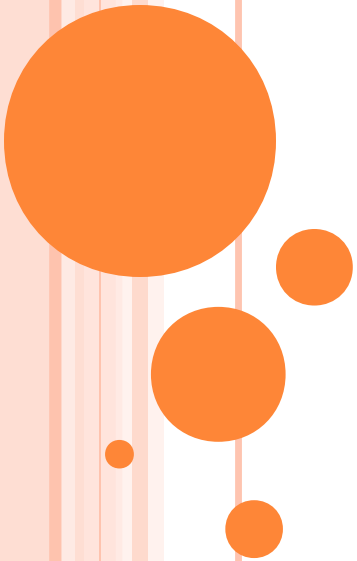

INTRODUZIONE ALLA LOGICA DI HOARE

Corso di Logica per la Programmazione

A.A. 2013/14

INTRODUZIONE

- Dall'inizio del corso ad ora abbiamo introdotto, un po' alla volta, un linguaggio logico sempre più ricco:
  - connettivi logici (Calcolo Proposizionale)
  - termini e quantificatori (Logica del Primo Ordine)
  - uguaglianza e disuguaglianze
  - insiemi e intervalli
  - quantificatori funzionali
- Per ognuna di queste estensioni abbiamo presentato:
  - la sintassi con grammatiche in BNF
  - la semantica (tabelle di verità, interpretazioni e modelli)
  - esempi di formalizzazione di enunciati
  - alcune leggi, e esempi di dimostrazioni

SUI LINGUAGGI DI PROGRAMMAZIONE

- In questa parte finale del corso sfruttiamo la logica introdotta per fornire una *semantica formale* di un semplice linguaggio di programmazione imperativo

  Perché?
- La presentazione di un *linguaggio di programmazione* di solito consiste nel dare la *sintassi* e una *semantica*
  - ***Sintassi*** spesso fornita con strumenti *formali*
    - es. grammatica in Backus-Naur Form (BNF)
  - ***Semantica*** spesso data in modo *informale*
    - tipicamente di stile operazionale
    - comprensibile per non esperti
    - ma lascia spazio a ambiguità
    - non sufficiente per applicazioni “critiche”

NECESSITÀ DI UNA SEMANTICA FORMALE

- A volte è necessario ***dimostrare*** proprietà relative a programmi
  - Software per controllo di centrali nucleari, di armamenti, di apparecchiature mediche, software/protocolli per e-banking, per gestione carte di credito, ...
- Sono state proposte varie *semantiche formali* tra cui:
  - **operazionale**, definendo struttura degli stati e transizioni di stato
  - **denotazionale**, con domini semantici e funzioni di interpretazione per i costrutti del linguaggio (come quella del C a PRL)
  - **assiomatica**, annotando un programma con asserzioni (formule) e poi dimostrandone la correttezza con proof rules.
- Noi vedremo la semantica assiomatica di Hoare [1969] - Dijkstra [1976]

UN ESEMPIO DI PROGRAMMA

- Assumiamo che **a** : array [0, n) of int
- Cosa calcola il seguente programma?

```
 x, c := 0, 0;
while x < n do
        if (a[x] > 0) then  c := c + 1   else  skip   fi ;
        x := x + 1
endw
```

- Il numero di elementi di **a** che sono maggiori di 0
- Tra poco saremo in grado di **dimostrarlo formalmente!**

COSA VEDREMO...

- Presentiamo dapprima le **espressioni** (a valori interi e booleani)
  - sintassi con grammatica BNF
  - semplice semantica in stile denotazionale
- Quindi presentiamo i **comandi** (skip, assegnamento, condizionale, iterazione (*while*))
  - sintassi con grammatica BNF
  - semantica operazionale *informale*
- Poi introduciamo le **Triple di Hoare** che permettono di annotare un programma con asserzioni
  - definiremo quando una tripla è **soddisfatta**
  - vedremo **regole di inferenza** per dimostrare che una tripla è soddisfatta (usando **induzione strutturale** sul programma)

IL PROGRAMMA "ANNOTATO"

{a : array [0, n) of int }

x, c := 0, 0;

{*Inv* : c = #{j | j ∈ [0, x) ∧ a[j] > 0 } ∧ x∈ [0, n] } {*t* : n - x}

while x < n **do**

    if (a[x] > 0) **then** c := c+1 **else skip** **fi** ;

    x := x + 1

endw

{*Inv* ∧ ~(x < n)}

{ c = #{j : j ∈ [0, n) ∧ a[j] > 0 } }

Saremo in grado di dimostrare che alla fine dell'esecuzione è vera l'ultima formula

LINGUAGGIO IMPERATIVO MINIMO

- **Espressioni (a valori interi o booleani)**:

  Exp ::= Const | Ide | (Exp) | Exp Op Exp | not Exp

  Op ::= + | - | * | div | mod |

  = | ≠ | < | ≤ | > | ≥ | or | and

  Const ::= Num | Bool Bool ::= *true* | *false*

  Num ::= 0 | -1 | 1 | ...

- **Comandi**:

  Com ::= **skip** | Ide_List := Exp_List | Com ; Com |

  if Exp **then** Com **else** Com **fi** |

  while Exp **do** Com **endw**

  Ide_List ::= Ide | Ide, Ide_List

  Exp_List ::= Exp | Exp, Exp_List

STATO DI UN PROGRAMMA

- Uno **stato** di un programma è una funzione da *identificatori* di variabili a *valori* (booleani e interi)

  $\sigma : \text{Ide} \rightarrow B \cup Z$

- Poiché in uno stato l'insieme delle variabili è finito, si può usare una rappresentazione estensionale. Esempio:

  $\sigma = \{\langle x, 18\rangle, \langle y, \text{true}\rangle, \langle z, -8\rangle\}$

- Lo stato, definito così, rappresenta *in modo astratto* lo stato della memoria usata dal programma.
- Non si possono modellare concetti come *aliasing* (due variabili che denotano la stessa cella di memoria) e *puntatori*.

VALORE DI UN'ESPRESSIONE

- Come visto, le espressioni possono contenere variabili
- Il ***valore*** di un'espressione dipende dal valore associato alle variabili, quindi dipende dallo stato.
- Definiamo la **funzione di interpretazione semantica**

$$E : \mathbf{Exp} \times (\mathbf{Ide} \rightarrow \mathbf{B} \cup \mathbf{Z}) \rightarrow \mathbf{B} \cup \mathbf{Z}$$

***E*(e,σ)** denota il valore dell'espressione **e** nello stato **σ**

$$[\, E : \mathbf{Exp} \times \mathbf{State} \rightarrow \mathbf{B} \cup \mathbf{Z} \,]$$

- Esempi (con σ = {<x, 18>, <y, true>, <z, -8>}):
  - ***E***(x + z, σ) = 10
  - ***E***(y, σ) = true

SEMANTICA DELLE ESPRESSIONI

- La funzione di interpretazione semantica ***E*** è definita in modo induttivo (sulla struttura delle espressioni) così:

$$
\begin{array}{rcll}
\mathcal{E}(true,\sigma) & = & \mathbf{tt} & \\
\mathcal{E}(false,\sigma) & = & \mathbf{ff} & \\
\mathcal{E}(n,\sigma) & = & \mathbf{n} & \text{se} \quad n \in \texttt{Num} \\
\mathcal{E}(x,\sigma) & = & \sigma(x) & \text{se} \quad x \in \texttt{Ide} \\
\mathcal{E}(E\ op\ E',\sigma) & = & \mathcal{E}(E,\sigma)\ \mathbf{op}\ \mathcal{E}(E',\sigma) & \text{se} \quad op \in \{+,-,div,mod,=,\neq,<,>,\leq,\geq\} \\
 & & & \quad \mathcal{E}(E,\sigma) \in \mathbb{Z} \ \text{ e } \ \mathcal{E}(E',\sigma) \in \mathbb{Z} \\
\mathcal{E}(E\ op\ E',\sigma) & = & \mathcal{E}(E,\sigma)\ \mathbf{op}\ \mathcal{E}(E',\sigma) & \text{se} \quad op \in \{and,or,=,\neq\} \\
 & & & \quad \mathcal{E}(E,\sigma) \in \mathbb{B} \ \text{ e } \ \mathcal{E}(E',\sigma) \in \mathbb{B} \\
\mathcal{E}(not\ E,\sigma) & = & \neg\mathcal{E}(E,\sigma) & \text{se} \quad \mathcal{E}(E,\sigma) \in \mathbb{B} \\
\mathcal{E}((E),\sigma) & = & \mathcal{E}(E,\sigma) &
\end{array}
$$

- Nota: ***E*** è una funzione **parziale**
  - es: ***E*****(**5 + true**, σ)** non è definita)

SIGNIFICATO INFORMALE DEI COMANDI

- L'esecuzione di un comando semplice tipicamente ha l'effetto di cambiare lo stato della memoria (si pensi ad un assegnamento singolo o multiplo)
- I comandi composti (sequenza, condizionale, iterazione) hanno "solo" il ruolo di controllare il flusso di esecuzione di comandi semplici. Naturalmente il loro effetto cambia al cambiare dello stato in cui vengono eseguiti.
- In generale, possiamo dire che l'esecuzione di un comando causa una transizione (un "passaggio") da uno stato (quello in cui inizia l'esecuzione del comando) ad un altro (quello in cui l'esecuzione termina).
- Un comando potrebbe non terminare, cioè potrebbe non arrivare ad uno stato finale. Per esempio:

  x := y + z nello stato $\sigma = \{\langle x, 5\rangle, \langle y, 3\rangle, \langle z, true\rangle\}$

  while true **do** skip **endw**

SIGNIFICATO INFORMALE DEI COMANDI

- L'esecuzione di **skip** a partire dallo stato σ porta nello stato σ
- L'esecuzione dell'assegnamento **x1, ..., xn := E1, ..., En** a partire dallo stato σ porta nello stato σ[*E(E1,σ)*/x1, ..., *E(En,σ)*/xn]
- L'esecuzione di **C;C'** a partire dallo stato σ porta nello stato σ' ottenuto eseguendo **C'** a partire dallo stato σ'' ottenuto dall'esecuzione di **C** nello stato σ.
- L'esecuzione di **if E then C else C' fi** a partire da σ porta nello stato σ' che si ottiene dall'esecuzione di **C** in σ, se *E(E,σ)= **tt**,* e dall'esecuzione di **C'** in σ, *se E(E,σ)= **ff***
- L'esecuzione del comando **while E do C endw** a partire da σ porta in σ se *E(E,σ)= **ff**,* altrimenti porta nello stato σ' ottenuto dall'esecuzione di **while E do C endw** a partire dallo stato σ'' ottenuto con l'esecuzione di **C** nello stato σ.

TRIPLE DI HOARE

- Una Tripla di Hoare ha la forma

  {Q} C {R}

  dove **C** è un **comando**, mentre **Q** e **R** sono **asserzioni**, ovvero formule ben formate in cui possono comparire le variabili dello stato
- Il dominio di interpretazione delle asserzioni è $\mathbf{B} \cup \mathbf{Z}$
- Significato intuitivo: la tripla **{Q} C {R}** è **soddisfatta** se

  *a partire da uno stato che soddisfi **Q**, l'esecuzione del comando **C** termina in uno stato che soddisfa **R***
- Es: $\{x > 1\}\ x := x + 1\ \{x > 2\}$

NOTAZIONE

- *free*(P) è l'insieme delle variabili libere nell'asserzione P

  p.e. *free*($x > y \wedge z \leq 1$) = {x, y, z}

- Sia P un'asserzione e σ uno stato. P^{σ} indica l'asserzione P istanziata sullo stato σ, ovvero in cui a tutte le variabili sono sostituiti i loro valori nello stato

- Poiché l'interpretazione del linguaggio delle asserzioni è fissata, dato uno stato si può valutare l'asserzione

  p.e. σ = {<x, 18>, <y, true>, <z, -8>}

  P = $(x > 0 \wedge z < 0)$

  P^{σ} = $(18 > 0 \wedge -8 < 0) \equiv T$

- Quindi uno stato determina univocamente un'interpretazione (nel senso del prim'ordine)

STATI COME INTERPRETAZIONI E MODELLI

- Scriviamo $\sigma \models P$ sse $P^{\sigma} \equiv T$, quindi se σ è un *modello* di P.
  In questo caso diciamo che "lo stato σ **soddisfa** l'asserzione P".
- Con {P} indichiamo l'insieme degli stati che soddisfano P
- Sia E una espressione, P un'asserzione. Valgono le seguenti proprietà:
  - Se per ogni $x \in free(E)$, $\sigma(x) = \sigma'(x)$ allora
    $E(E, \sigma) = E(E, \sigma')$
  - Se per ogni $x \in free(P)$, $\sigma(x) = \sigma'(x)$ allora
    $\sigma \models P$ se e solo se $\sigma' \models P$
  - Per ogni variabile x
    $\sigma[E(E, \sigma)/x] \models P$ se e solo se $\sigma \models P[E/x]$

GENERALITÀ SULLE TRIPLE DI HOARE

- Data la tripla di Hoare {Q} C {R}
  - Q è detta precondizione
  - R è detta postcondizione
  - La tripla è soddisfatta se:
    - **per ogni** stato σ che soddisfa Q:
      - L'esecuzione del comando C a partire dallo stato σ **termina** producendo uno stato σ'
      - Lo stato σ' soddisfa R

INTERPRETAZIONE DELLE TRIPLE DI HOARE (1)

- *Semantica*: dati Q e C, determinare R in modo che sia soddisfatta {Q} C {R} è un modo per descrivere il comportamento di C.

  Per esempio sia Q ≡ {x>10} e

  C ≡ **if** (x < 10) **then** x := 1 **else** x :=2 **fi**

  allora R ≡ {x = 2} descrive il comportamento del comando

- *Correttezza*: dati Q, C ed R, dimostrare che la tripla

  {Q} C {R} è soddisfatta corrisponde ad una dimostrazione di correttezza del comando C rispetto alla pre e alla postcondizione

INTERPRETAZIONE DELLE TRIPLE DI HOARE (2)

- *Specifica.* Data una precondizione Q e una postcondizione R determinare un comando C che soddisfa la tripla {Q} C {R} equivale a scrivere il programma che realizza le specifiche Q ed R.
- Per esempio le specifiche $Q \equiv \{x > 0\}$ e $R \equiv \{y = 2 \times x\}$ possono essere soddisfatte dal comando

  y := x * 2

  oppure dal comando

  y := x + x

  oppure dal comando

  y := x; y := y * 2

  oppure dal comando

  y := 10; x := 5 (!!!)

  ecc...

PROOF SYSTEM PER VERIFICA DI TRIPLE: ALCUNE REGOLE DI INFERENZA

(pre-post o conseguenza)

$$\frac{P \Rightarrow P' \quad \{P'\}\ C\ \{R'\} \quad R' \Rightarrow R}{\{P\}\ C\ \{R\}}$$

(pre)

$$\frac{P \Rightarrow P' \quad \{P'\}\ C\ \{R\}}{\{P\}\ C\ \{R\}}$$

(post)

$$\frac{\{P\}\ C\ \{R'\} \quad R' \Rightarrow R}{\{P\}\ C\ \{R\}}$$

- La correttezza di queste regole segue immediatamente dalla definizione di “tripla soddisfatta”

ASSIOMA PER SKIP

- **Assioma** per il comando vuoto:

$$\{P\}\ \textbf{skip}\ \{P\} \qquad \text{(SKIP)}$$

- Correttezza? Ovvia...

- Infatti ricordiamo il *significato informale:*
  - L'esecuzione di **skip** a partire dallo stato σ porta nello stato σ

ASSIOMA PER ASSEGNAMENTO SEMPLICE

- **Assioma** per l'assegnamento semplice

$$\{def(E) \wedge P[E/x]\} \quad x := E \quad \{P\} \qquad (ASS)$$

  dove *def*(E) è vera in uno stato σ se il valore di E in σ (cioè ***E***(E,σ)) è ben definito (vedi prossima pagina...)
- L'idea è che affinché l'asserzione P sia soddisfatta dopo l'assegnamento, la stessa P deve essere soddisfatta dallo stato precedente l'assegnamento quando all'identificatore di variabile è sostituita l'espressione
- La correttezza della regola si vede confrontando l'assioma con la semantica informale:
  - L'esecuzione dell'assegnamento **x := E** a partire dallo stato σ porta nello stato $\sigma[\boldsymbol{E}(E,\sigma)/x]$

DEFINIZIONE DI ESPRESSIONI

- La funzione *def(_)*, applicata a un'espressione *E,* restituisce una asserzione tale che $\sigma \models def(E)$ se esiste un v tale che $\boldsymbol{E}(E, \sigma) = v$.
- Serve per garantire che la valutazione di *E* sia definita e termini.

$$
\begin{array}{rcll}
def(c) & = & \mathbf{tt} & \text{se} \quad c \in \texttt{Num} \text{ o } c \in \texttt{Bool} \\
def(x) & = & \mathbf{tt} & \text{se} \quad x \in \texttt{Ide} \\
def(E \ op \ E') & = & def(E) \wedge def(E') & \text{se} \quad op \in \left\{ \begin{array}{l} +, -, <, >, \leq, \geq \\ =, \neq, and, or \end{array} \right\} \\
def(E \ op \ E') & = & def(E) \wedge def(E') \wedge E' \neq 0 & \text{se} \quad op \in \{div, mod\} \\
def(not \ E) & = & def(E) & \\
def((E)) & = & def(E) &
\end{array}
$$