

# Logica per la Programmazione

## Lezione 13

- ▶ Sistema di Dimostrazioni per le Triple di Hoare
- ▶ Comando Iterativo

## Semantica informale del Comando Iterativo

- ▶ Come per gli altri comandi (assegnamento, sequenza di comandi, skip, condizionale) introdurremo una regola di inferenza per il comando iterativo sulla base della sua semantica informale:
- ▶ L'esecuzione del comando **while E do C endw** a partire da  $\sigma$  porta in  $\sigma$  se  $\mathcal{E}(E, sg) = \mathbf{ff}$ , altrimenti porta nello stato  $\sigma'$  ottenuto dall'esecuzione di **while E do C endw** a partire dallo stato  $\sigma''$  ottenuto con l'esecuzione di *C* nello stato  $\sigma$ .
- ▶ In altre parole, l'esecuzione inizia valutando la *guardia E*. Se il valore è **ff** l'esecuzione del **while** termina senza modificare lo stato. Altrimenti si esegue il *corpo C*, e nello stato risultante si riesegue l'intero comando **while**.

## Semantica informale del Comando Iterativo

- ▶ Quindi l'esecuzione del while comporta l'esecuzione del comando  $C$  un certo numero di volte, non determinabile a priori. Inoltre l'esecuzione potrebbe non portare ad un stato definito se
  - ▶ la guardia non è valutabile ( $def(E)$  è falso), oppure
  - ▶ la guardia è sempre vera (ciclo infinito)
  
- ▶ **Attenzione:** Per una spiegazione approfondita di come si può arrivare alla regola di inferenza presentata di seguito, si veda il Paragrafo 4.7 della dispensa sulle Triple di Hoare

## Regola per il Comando Iterativo

$$\frac{
 \begin{array}{l}
 \{Inv \wedge E \wedge t = V\} C \{t < V\} \quad Inv \Rightarrow t \geq 0 \\
 P \Rightarrow Inv \wedge def(E) \quad \{Inv \wedge E\} C \{Inv \wedge def(E)\} \quad Inv \wedge \neg E \Rightarrow Q
 \end{array}
 }{
 (WHILE) \quad \{P\} \text{ while } E \text{ do } C \text{ endw } \{Q\}
 }$$

- ▶  $t$  è chiamata **funzione di terminazione**
- ▶  $Inv$  è chiamata **invariante**
- ▶  $V$  è una **variabile di specifica**: denota un generico valore, non utilizzabile e non modificabile nel programma
- ▶  $\{Inv \wedge E \wedge t = V\} C \{t < V\}$  è l'**ipotesi di progresso**
- ▶  $(Inv \Rightarrow t \geq 0)$  è l'**ipotesi di terminazione**
- ▶  $\{Inv \wedge E\} C \{Inv \wedge def(E)\}$  è l'**ipotesi di invarianza**

## Regola per il Comando Iterativo: alcuni commenti

$$\{Inv \wedge E \wedge t = V\} C \{t < V\}$$

$$Inv \Rightarrow t \geq 0$$

$$P \Rightarrow Inv \wedge def(E)$$

$$\{Inv \wedge E\} C \{Inv \wedge def(E)\}$$

$$Inv \wedge \neg E \Rightarrow Q$$

(WHILE)  $\{P\}$  **while**  $E$  **do**  $C$  **endw**  $\{Q\}$

Le condizioni della regola formano due gruppi:

- ▶ Le ipotesi di **terminazione** e di **progresso**, insieme, garantiscono che il comando iterativo termini. Infatti
  - ▶ ad ogni iterazione il valore di  $t$  decresce di almeno un'unità, visto che  $t$  assume valori interi
  - ▶  $t$  non può assumere valori negativi durante l'esecuzione del ciclo, perché l'invariante è sempre vera.
- ▶ Le altre tre condizioni consentono di dimostrare la postcondizione  $Q$  sfruttando ipotesi presenti nell'invariante (e indirettamente nella precondizione  $P$ ), e la negazione delle guardia

## Esempio di Comando Iterativo

Usando come **invariante**  $Inv : s = (\sum i : i \in [0, x] . i) \wedge 0 \leq x \wedge x \leq n$   
 e come **funzione di terminazione**  $t : n - x$  verificare la tripla

```

{ s = 0 ∧ x = 0 ∧ n ≥ 0 }
while x < n do
  x, s := x + 1, s + x
endw
{ s = (∑ i : i ∈ [0, n] . i) }
  
```

Per la Regola per il Comando Iterativo è sufficiente mostrare:

[Pre]  $s = 0 \wedge x = 0 \wedge n \geq 0 \Rightarrow def(x < n) \wedge Inv$

[Post]  $Inv \wedge \neg(x < n) \Rightarrow s = (\sum i : i \in [0, n] . i)$

[Term]  $Inv \Rightarrow n - x \geq 0$

[Inv]  $\{ Inv \wedge x < n \} x, s := x + 1, s + x \{ Inv \wedge def(x < n) \}$

[Prog]  $\{ Inv \wedge x < n \wedge n - x = V \} x, s := x + 1, s + x \{ n - x < V \}$

**Esercizio:** completare la dimostrazione

## Comando di Inizializzazione

- ▶ Spesso la preconditione di una tripla con un while non è sufficiente per soddisfare la condizione  $[Pre] P \Rightarrow Inv \wedge def(E)$
- ▶ In questo caso si può inserire un **comando di inizializzazione**  $C_I$  tale che  $\{P\} C_I \{Inv \wedge def(E)\}$
- ▶ **Esempio.** Nella tripla vista, se la preconditione è solo  $\{n \geq 0\}$ , la  $[Pre]$  è falsa (invariante e  $def(x < n)$  non valgono).
- ▶ Possiamo renderla vera con un comando che inicializzi  $x$  e  $s$ .

```

{ n ≥ 0 }??
while x < n do
x, s := x + 1, s + x
endw
{ s = (∑ i : i ∈ [0, n] . i) }

```

```

{ n ≥ 0 }
x, s := 0, 0 ;
{ s = 0 ∧ x = 0 ∧ n ≥ 0 }
while x < n do
x, s := x + 1, s + x
endw
{ s = (∑ i : i ∈ [0, n] . i) }

```

## Programmi Annotati

- ▶ Si possono aggiungere annotazioni al programma per rendere esplicito ciò che si deve dimostrare, p.es. *invariante* e *funzione di terminazione* di un ciclo, o *asserzione* in un punto arbitrario del programma.
- ▶ Per esempio, annotiamo il programma a sinistra come mostrato a destra:

```

{ n ≥ 0 }
x, s := 0, 0;
while x < n do
x, s := x + 1, s + x
endw
{ s = (∑ i : i ∈ [0, n) . i) }
  
```

```

{ n ≥ 0 }
x, s := 0, 0;
{ P : s = 0 ∧ x = 0 ∧ n ≥ 0 }
{ Inv : s = (∑ i : i ∈ [0, x) . i) ∧ 0 ≤ x ∧ x ≤ n }
{ t : n - x }
while x < n do
x, s := x + 1, s + x
endw
{ s = (∑ i : i ∈ [0, n) . i) }
  
```

- ▶ Le annotazioni forniscono, in questo caso, le info necessarie per applicare le regole della sequenza e del comando iterativo.



## Esempio di Programma Annotato: formule da dimostrare

```

{ $n \geq 0$ }
 $x, s := 0, 0;$ 
{ $P : s = 0 \wedge x = 0 \wedge n \geq 0$ }
{ $Inv : s = (\sum i : i \in [0, x]. i) \wedge 0 \leq x \wedge x \leq n$ }
{ $t : n - x$ }
while  $x < n$  do
 $x, s := x + 1, s + x$ 
endw
{ $s = (\sum i : i \in [0, n]. i)$ }

```

[ASS]  $\{n \geq 0\} \ x, s := 0, 0 \ \{P : s = 0 \wedge x = 0 \wedge n \geq 0\}$

[Pre]  $s = 0 \wedge x = 0 \wedge n \geq 0 \Rightarrow \text{def}(x < n) \wedge \text{Inv}$

[Post]  $\text{Inv} \wedge \neg(x < n) \Rightarrow s = (\sum i : i \in [0, n]. i)$

[Term]  $\text{Inv} \Rightarrow n - x \geq 0$

[Inv]  $\{\text{Inv} \wedge x < n\} \ x, s := x + 1, s + x \ \{\text{Inv} \wedge \text{def}(x < n)\}$

[Prog]  $\{\text{Inv} \wedge x < n \wedge n - x = V\} \ x, s := x + 1, s + x \ \{n - x < V\}$

## Esercizio: Somma con Incrementi

```

{z = A ∧ n = B ∧ B ≥ 0}
{Inv : z + n = A + B ∧ n ≥ 0}{t : n}
while not (n = 0) do
z := z + 1; n := n - 1
endw
{Inv ∧ n = 0}
{z = A + B}

```

Il programma calcola  
in **z** la somma dei valori iniziali di **z** ed **n**  
usando incrementi unitari

Per verificare la tripla, per la Regola per il Comando Iterativo è sufficiente dimostrare:

$$[\text{Pre}] \quad z = A \wedge n = B \wedge B \geq 0 \Rightarrow \text{Inv} \wedge \text{def}(\text{not}(n = 0))$$

$$[\text{Post}] \quad \text{Inv} \wedge n = 0 \Rightarrow z = A + B$$

$$[\text{Inv}] \quad \{\text{Inv} \wedge \text{not}(n = 0)\} \quad z := z + 1; \\ n := n - 1 \quad \{\text{Inv} \wedge \text{def}(\text{not}(n = 0))\}$$

$$[\text{Term}] \quad \text{Inv} \Rightarrow n \geq 0$$

$$[\text{Prog}] \quad \{\text{Inv} \wedge \text{not}(n = 0) \wedge n = V\} \quad z := z + 1; n := n - 1 \quad \{n < V\}$$

## Esercizio: Calcolo MCD

```

{x = A ∧ y = B ∧ A > 0 ∧ B > 0}
{Inv : x > 0 ∧ y > 0 ∧ mcd(A, B) = mcd(x, y)}{t : x + y}
while (x <> y) do
if x > y then x := x - y else y := y - x fi
endw
{x = mcd(A, B)}

```

Dimostrare la correttezza del programma annotato, facendo uso delle seguenti note proprietà dell'operatore *mcd*:

$$mcd(v, w) = \begin{cases} v & \text{se } v = w \\ mcd(v - w, w) & \text{se } v > w \\ mcd(v, w - v) & \text{se } v < w \end{cases}$$

## Esercizio: Calcolo del Fattoriale

- ▶ Si consideri la seguente specifica:

$$\{n > 0\} C \{f = n!\}$$

- ▶ Si chiede quindi di scrivere un comando  $C$  che calcoli il fattoriale di un numero  $n$  maggiore di zero
- ▶ Proponiamo come soluzione il seguente programma:

```

{ $n > 0$ }
 $f, x := 1, 1;$ 
while ( $x \leq n$ ) do
 $f, x := f * x, x + 1;$ 
endw
{ $Inv \wedge \neg(x \leq n)$ }
  
```

- ▶ Dobbiamo verificare che la tripla sia soddisfatta
- ▶ Ma come possiamo determinare l'invariante e la funzione di terminazione?

## Esercizio: Calcolo del Fattoriale (2)

- ▶ Eseguiamo manualmente il programma (es: per  $n = 5$ ):

```

{ n > 0 }
  f, x := 1, 1;
  { Inv :? } { t :? }
  while ( x <= n ) do
    f, x := f * x, x + 1;
  endw
  { Inv ∧ ¬( x <= n ) }
  { f = n! }
  
```

x	f
1	1
2	1
3	2
4	6
5	24
<b>6</b>	<b>120</b>

- ▶ Osserviamo che, ad ogni iterazione, i valori di  $x$  e  $f$  sono legati dalla seguente relazione

$$f = (x - 1)!$$

- ▶ Tuttavia, scegliendo  $Inv = f = (x - 1)!$  non riusciamo a dimostrare

$$\text{[Post]} \quad Inv \wedge \neg E \Rightarrow Q \quad \equiv$$

$$f = (x - 1)! \wedge \neg(x \leq n) \Rightarrow f = n!$$

## Esercizio: Calcolo del Fattoriale (3)

- ▶ Aggiungendo  $x \in [0, n + 1]$  in *Inv* siamo in grado di dimostrare:

$$[\text{Post}] \quad \text{Inv} \wedge \neg(x \leq n) \Rightarrow f = n!$$

- ▶ Aggiungiamo l'invariante alla tripla:

```

{ n > 0 }
f, x := 1, 1;
{ Inv : f = (x - 1)! ∧ x ∈ [0, n + 1] } { t : ? }
while (x ≤ n) do
  f, x := f * x, x + 1;
endw
{ Inv ∧ ¬(x ≤ n) }
  
```

- ▶ Possiamo ora verificare la tripla...

## Esercizio: Calcolo del Fattoriale (4)

- ▶ Ipotesi di invarianza

$$\{f = (x-1)! \wedge x \in [0, n+1)\} f, x := f * x, x+1 \{f = (x-1)! \wedge x \in [0, n+1]\}$$

- ▶ Per la regola dell'assegnamento multiplo, basta dimostrare:

$$f = (x-1)! \wedge x \in [0, n+1) \Rightarrow \text{def}(f * x) \wedge \text{def}(x+1) \wedge f * x = x! \wedge x+1 \in [0, n+1]$$

- ▶ Partiamo dalla conseguenza (eliminando i  $\text{def}(\dots)$  che sono **T**):

$$f * x = x! \wedge x + 1 \in [0, n + 1]$$

$$\equiv \{\text{Ip: } f = (x - 1)!\}$$

$$(x - 1)! * x = x! \wedge x + 1 \in [0, n + 1]$$

$$\equiv \{\text{def. fattoriale}\}$$

$$x + 1 \in [0, n + 1]$$

$$\equiv \{\text{Ip: } x \in [0, n + 1), \text{ calcolo}\}$$

**T**

- ▶ La funzione di terminazione (che è molto semplice) è lasciata per esercizio.