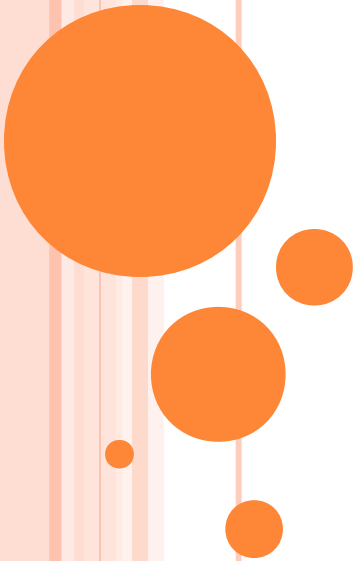

REGOLE DI INFERENZA PER TRIPLE DI HOARE: COMANDO ITERATIVO

Corso di Logica per la Programmazione

SEMANTICA INFORMALE DEL COMANDO ITERATIVO

- L'esecuzione del comando **while E do C endw** a partire da σ porta in σ se $E(E,\sigma) = \mathit{ff}$, 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.
- 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 è sempre vera (ciclo infinito), oppure
  - la guardia non è valutabile (*def*(**E**) è falso)
- Per vedere 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

$$\frac{P \Rightarrow Inv \wedge def(E) \qquad Inv \wedge \sim E \Rightarrow Q \qquad Inv \Rightarrow t \geq 0 \qquad \{Inv \wedge E\}\ C\ \{Inv \wedge def(E)\} \qquad \{Inv \wedge E \wedge t = V\}\ C\ \{t < V\}}{\{P\}\ \textbf{while}\ E\ \textbf{do}\ C\ \textbf{endw}\ \{Q\}}$$

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

ESEMPIO DI COMANDO ITERATIVO

- Usando come **invariante**

  $Inv : s = (\Sigma\, i : i \in [0, x).\ i) \ \wedge\ 0 \le x \wedge x \le n$

e come **funzione di terminazione**

  $t : n - x$

verificare la tripla nel riquadro a destra.

$\{s = 0 \ \wedge\ x = 0 \wedge n \ge 0\}$
while $x < n$ **do**
    x, s := x+1, s+x
endw
$\{s = (\Sigma\, i: i \in [0, n).\ i)\}$

- Per la **Regola per il Comando Iterativo** è sufficiente mostrare:

1) $s = 0 \wedge x = 0 \wedge n \ge 0 \Rightarrow def(x<n) \wedge s = (\Sigma\, i : i \in [0, x).\ i) \ \wedge\ 0 \le x \wedge x \le n$

2) $s = (\Sigma\, i : i \in [0, x).\ i) \ \wedge\ 0 \le x \wedge x \le n \wedge \sim(x < n) \Rightarrow s = (\Sigma\, i: i \in [0, n).\ i)$

3) $s = (\Sigma\, i : i \in [0, x).\ i) \ \wedge\ 0 \le x \wedge x \le n \ \Rightarrow n - x \ge 0$

4) $\{s = (\Sigma\, i : i \in [0, x).\ i) \ \wedge\ 0 \le x \wedge x \le n \wedge x < n\}$ x, s := x+1, s+x
   $\{s = (\Sigma\, i : i \in [0, x).\ i) \ \wedge\ 0 \le x \wedge x \le n \wedge def(x < n)\}$

5) $\{s = (\Sigma\, i : i \in [0, x).\ i) \ \wedge\ 0 \le x \wedge x \le n \wedge x < n \wedge n - x = V\}$
   x, s := x+1, s+x $\{n - x < V\}$

Esercizio: completare la dimostrazione

COMANDO DI INIZIALIZZAZIONE

- Spesso la precondizione di una tripla con un **while** non è sufficiente per soddisfare la condizione $P \Rightarrow Inv \wedge def(E)$
- In questo caso si può inserire un **comando di inizializzazione** C_I tale che $\{P\}\ C_I\ \{Inv \wedge def(E)\}$
- **Esempio. N**ella tripla vista, se la precondizione è solo $\{n \geq 0\}$, la 1) è falsa (invariante e *def*(x<n) non valgono).
- Possiamo renderla vera con un comando che inizializzi **x** e **s**.

$\{n \geq 0\}$ **??**

while x < n **do**

x, s := x+1, s+x

endw

$\{s = (\Sigma\ i: i \in [0, n).\ i)\}$

$\{n \geq 0\}$

x, s := 0, 0 ;

$\{s = 0 \wedge x = 0 \wedge n \geq 0\}$

while x < n **do**

x, s := x+1, s+x

endw

$\{s = (\Sigma\ i: i \in [0, n).\ i)\}$

PROGRAMMI ANNOTATI

- Invece di indicare solo pre- e post-condizioni di un programma, come a destra, è utile aggiungere altre annotazioni per facilitarne la comprensione.
- Per esempio, annotiamo il programma come sotto con invariante, funzione di terminazione e altre asserzioni. Questo rende esplicito cosa bisogna dimostare:

1) $s = 0 \wedge x = 0 \wedge n \geq 0 \Rightarrow def(x<n) \wedge Inv$

2) $Inv \wedge \sim(x < n) \Rightarrow s = (\Sigma\ i: i \in [0, n).\ i)$

3) $Inv \wedge x < n \Rightarrow n - x \geq 0$

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

5) $\{Inv \wedge x < n \wedge n-x = V\}$ x,s := x+1,s+x $\{n-x < V\}$

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

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

ESERCIZIO: SOMMA CON INCREMENTI UNITARI

- Si consideri il programma annotato che calcola in **z** la somma dei valori di **z** ed **n** usando incrementi unitari. Si noti l'uso di variabili di specifica.
- Dimostrarne la correttezza.
- Per la **Regola per il Comando Iterativo**, usando le annotazioni occorre dimostrare:

$\{z = A \wedge n = B \wedge B \geq 0\}$

$\{Inv : z+n = A+B \wedge n \geq 0\}\ \{t : n\}$

while not (n = 0) **do**

z := z+1; n:= n-1

endw

$\{Inv \wedge n = 0\}$

$\{z = A+B\}$

1) $z=A \wedge n=B \wedge B\geq 0 \Rightarrow Inv \wedge def(\text{not}(n=0))$

2) $Inv \wedge n = 0 \Rightarrow z = A+B$

3) [Condizione di Invarianza]

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

4) [Condizione di Terminazione] $Inv \Rightarrow n \geq 0$

5) [Cond. di Progresso] $\{Inv \wedge \text{not}(n=0) \wedge n=V\}$ z := z+1; n:= n-1 $\{n<V\}$

ESERCIZIO: Calcolo MCD

- Si consideri il seguente programma annotato

$\{x = A \ \wedge \ y = B \wedge \ A>0 \wedge B>0\}$

$\{Inv : x>0 \wedge y>0 \wedge mcd(A,B) = mcd(x,y)\} \ \{t : x+y\}$

while x <> y **do**

if x>y **then** x:= x-y; **else** y := y-x; **fi**

endw

$\{x = mcd(A,B)\}$

- Dimostrarne la correttezza, facendo uso delle seguenti note proprietà dell'operatore *mcd*:

$mcd\ (v,w) = v$ se v=w

$mcd\ (v,w) = mcd\ (v-w,w)$ se v>w

$mcd\ (v,w) = mcd\ (v,w-v)$ se v<w

ESERCIZIO: Calcolo del fattoriale

- Si consideri la seguente specifica

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

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

- Come determinare l'invariante e la funzione di terminazione?

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

{n>0}	x	f
f , x:= 1, 1;	1	1
{*Inv* : ?} {*t* : ?}	**2**	**1**
while x <= n **do**	3	2
f, x := f * x, x + 1;	4	6
endw	5	24
{*Inv* ∧ ~(x <= n)}	**6**	**120**
{f = n!}		

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

  f = (x - 1)!

- Scegliendo questa formula come invariante non riusciamo a dimostrare

  f = (x - 1)! ∧ ~(x <= n) ⇒ f = n!

```
{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)}
{f = n!}
```

- Aggiungendo x∈[0, n+1] in *Inv* abbiamo:

$$Inv \wedge \sim(x <= n) \Rightarrow f = n!$$

- Dimostrazione per esercizio

- 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:

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

- Partiamo dalla conseguenza (eliminando i def(...) che sono **T**):

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

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

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

$\equiv$ { def. fattoriale}

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

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

T

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