

Correttezza algoritmi iterativi / 08-03 / 16-03

Idea:

// Precondizione

while (β)

C

// β espressione booleana (senza side-effect)

// C comando che modifica lo stato

non modifica
lo stato
!!

// Postcondizione

Esempio

// Precondizione $x = x_0 \wedge y = y_0$ $x_0, y_0 \in \mathbb{N}$

while ($x \neq 0$)

$x = x - 1$

$y = y + 1$

// Postcondizione: $x = 0 \wedge y = x_0 + y_0$

Correttezza rispetto a Pre/Post

= perche' da uno stato che soddisfa Pre
alla fine dell'esecuzione vole Post

Per verificarla, devo trovare un' espressione
detta INVARIANTE DI CICLO (Inv):

- vole all'inizio ($\text{Pre} \xrightarrow{\text{Inv}} \text{Inv}$)
- alla fine del ciclo garantisce Post ($\text{Inv} \wedge \neg \beta \Rightarrow \text{Post}$)
- Inv si preserva ad ogni iterazione

ovvero se vole Inv c B :

// Inv $\lambda \boxed{B}$ → significa che entra nel ciclo

C

// Inv

garantiscono che vole

// Precondizione

while (B)

C

// Postcondizione

Esempio

// Precondizione $x = x_0 \wedge y = y_0$ $x_0, y_0 \in \mathbb{N}$

while ($x \neq 0$)

$x = x - 1$

$y = y + 1$

// Postcondizione : $x = 0 \wedge y = x_0 + y_0$

Inv : $x + y = x_0 + y_0$ la somma dei due e' invariante

1) Pre \Rightarrow Inv $x = x_0 \wedge y = y_0 \Rightarrow x + y = x_0 + y_0$

2) Inv $\wedge x \neq 0 \stackrel{\text{def}}{\Rightarrow} x = 0 \wedge y = x_0 + y_0$

3) // $x + y = x_0 + y_0 \wedge x \neq 0$

C

\Rightarrow necessaria perché $x \in \mathbb{N}$ e se diventa negativo l'algoritmo si blocca

// $x + y = x_0 + y_0$

Vero che si preserva: tolgono una parte e somma l'oltre

Non e' Inv:

- $x \leq x_0 \wedge y \geq y_0$ vale all'inizio, \therefore preserva
ma non dice nulla sul fatto
che Post parla della somma

Supponendo ora $x_0, y_0 \in \mathbb{Z}$, l'algoritmo puo' non
terminare

Due nozioni di correttezza:

- **PARZIALE** se prima vale Pre e il ciclo, allora vale Post
 \sim
while(β) C
- **TOTALE** se prima vale Pre allora il ciclo termina
e vale Post

Nell'ultimo caso ($x_0, y_0 \in \mathbb{Z}$)

abbiamo una correttezza parziale. Obiettivo e' quella totale

Esempio

// Precondizione $x = x_0 \wedge y = y_0 \wedge x_0 \geq 0$ $x_0, y_0 \in \mathbb{N}$

while ($x \neq 0$)

$x = x - 1$

$y = y + 1$

// Postcondizione: $x = 0 \wedge y = x_0 + y_0$

| Inv: $x + y = x_0 + y_0$

$\wedge x_0 \geq 0$

Per provare anche la terminazione delle chiamate

VALORE

Inv è l'"funzione di terminazione"

Nell'esempio x deve essere mai limitata inferiormente.

Vediamo dunque

- 1) Inv vede all'inizio Pre \Rightarrow Inv
- 2) In uscita INV garantisce Post $inv \wedge \neg B \Rightarrow post$
- 3) Si preserva $\| inv \wedge B$
 C
 $\| inv$
- 4) A ogni iterazione il valore di t deve essere salvalmente
- 5) t è limitata inferiormente se vale Inv $\wedge B$
x già stato inserito

Esempio binay search iterativo

binary-search (x, a)

inf = 0 ; sup = n - 1

while (inf ≤ sup)

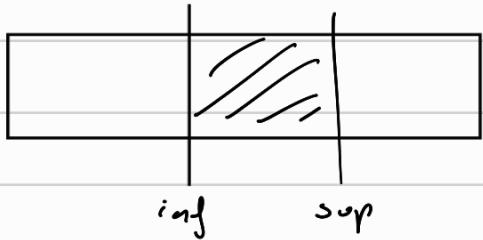
mid = (inf + sup) / 2

if ($x > a[mid]$)

inf = mid + 1

// Precondizione

inf = 0 \wedge sup = n - 1



else if ($x < \alpha[mid]$)

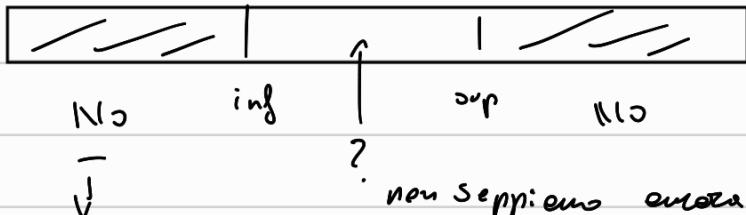
$sup = mid - 1$

else return true

→ Postcondizione : $x \notin \alpha[0, n-1]$

return false;

Iniz



Cose succede in un momento qualsiasi:

qui invece abbiamo i risultati dei confronti più
già ridotta la sequenza
dopo: confronti falliti

→ $x \notin \alpha[0, inf-1] \wedge x \in \alpha[sup+1, n-1]$

oppure $x \in \alpha[0, n-1] \Leftrightarrow x \in \alpha[inf, sup]$

funzione t di terminazione $\neq sup - inf$ (decrease scopus)

inv $x \notin \alpha[0, inf-1] \wedge x \notin \alpha[sup+1, n-1]$

t $sup - inf$

① Vole ell'inv? $inf = 0, sup = n-1$ s: (intervalli vuoti)

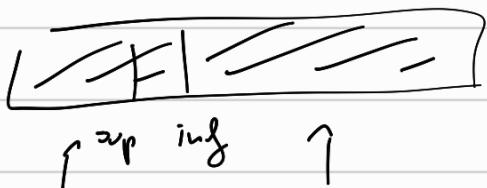


2) Garantisce Post?

Vole → ($\inf \leq \sup$) alla fine → $\inf > \sup$

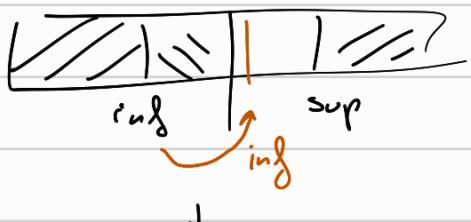
$\inf \geq \sup + 1$

(nella pratica =)



l'el non sta nelle due posizioni → Post garantito

3) Si preserva?



$x > \infty [\text{mid}]$

ma l'vv persiste, non è contenuto nelle posizioni esterne x

4) Decrese? Sì sup-inf

5) Limite inferiore?

$$\inf \leq \sup + 1$$

informalmente mi fermo quando

\inf supera \sup

(o viceversa)

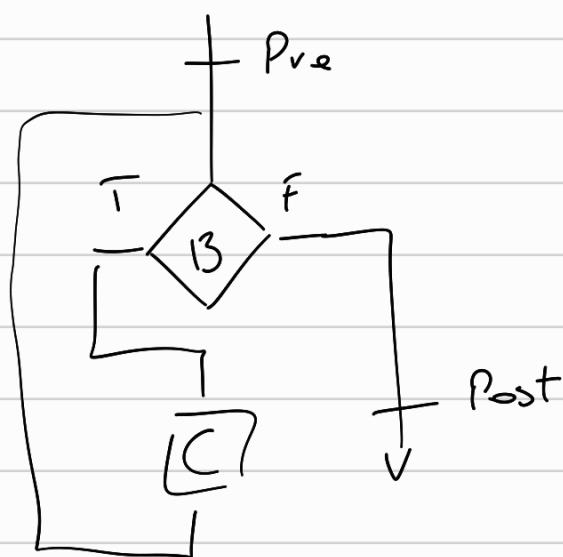
(ultimo giro)

$$\inf = \sup + 1$$

Note: si stipula un **CONTRATTO** fra programmatore e cliente / user
 → client garantisce Pv (input sensato)
 → programmatore garantisce Post (oggetto)

14-03

Proviamo che 1, 2, 3, 4, 5 implica while (B) C corretto rispetto
a Pre Post



- Supponiamo termini:
 - esegue C n volte $n \geq 0$
 - $n=0 \quad \text{INV} \wedge \neg B \rightarrow \text{Post}$
 - $n > 0 \quad \text{almeno 1 volta}$
 - $\| \text{ INV} \wedge \neg B$
 - C
 - $\| \text{ INV} \quad \text{e poi faccio } n-1 \text{ volte}$

- Potrebbe non terminare?

$\text{INV } C, \quad \text{INV } C \dots \quad \text{INV } C \dots \quad \text{sarebbe terminare}$
 $\wedge B \quad \wedge \neg B \quad \wedge \neg B$

$$t(s_0) > t(s_1) > t(s_i) >$$

Non esiste una cosa del genere, perché limitato inferiormente

Esempio - Bendicere nazionale olandese

red	white	blue
-----	-------	------

Dati una sequenza mescolata

B | R | N | R | R

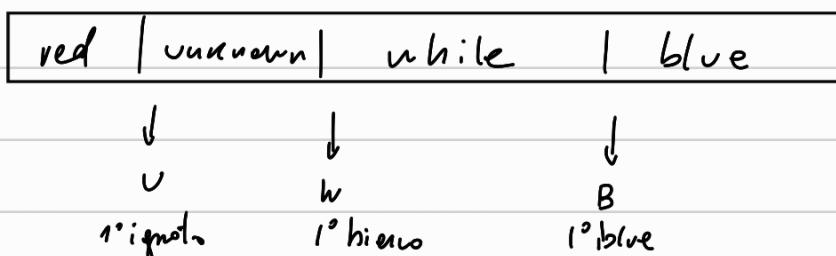
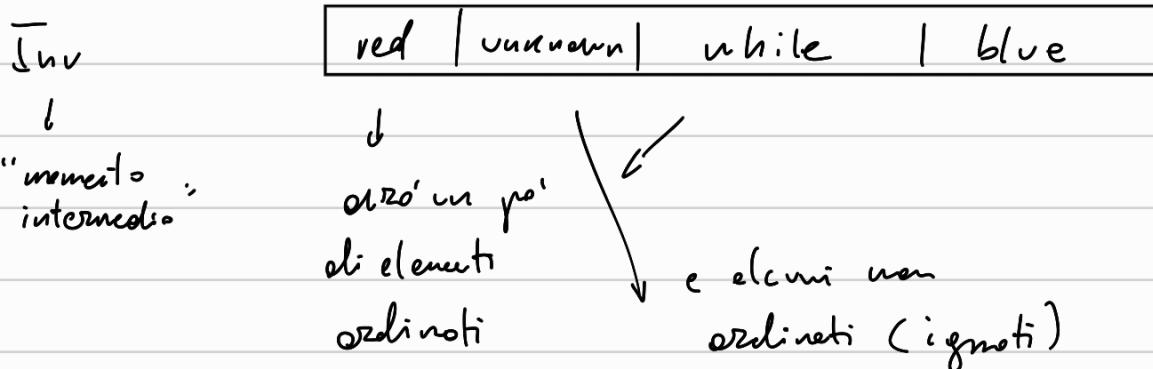
1..n

(es: vuole ordinare: prime tutti red, poi white, poi blue)

- $\text{Color}(i) \quad 1 \leq i \leq n$ restituisce il colore
- $\text{swap}(i, j) \quad 1 \leq i, j \leq n$ scambia

Pre: elementi disordinati

Post: elementi ordinati



Inv

$\forall i \ 1 \dots v-1 \ \text{Color}(i) = \text{red} \quad = \text{Red} \ (1, v-1) \ \text{per obbligazione}$

stesso caso x white e blue

$\rightarrow \text{Red} \ (1, v-1) \wedge \text{White} \ (w, \beta-1) \wedge \text{Blue} \ (\beta, n)$
 $v \leq w$

while ($v \leq w$) "finché ci sono ignoti"

case Color($w-1$)

Red : swap($w-1, v$); $v = v+1$

White : $w = w-1$

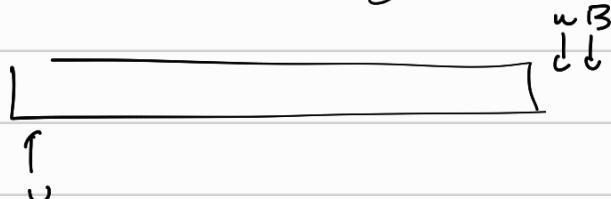
Blue : swap($w-1, B-1$); $w = w-1$; $B = B-1$

J

perché l'ultimo bianco
è l'ave intesta

① Pre \rightarrow inv $P_{\text{vc}} \equiv v=1 \wedge w=n+1 \wedge B=n+1$
(per far tornare i conti se White($n+1, n+1-1$) = \neq)

l'invertente ci guida: non so se alla fine ho un blu o bianco



② Inv $\wedge \neg B \rightarrow$ Post

$v=w$ Red($1, w-1$) \wedge White($w, B-1$) \wedge Blue(B, n)

(avendo non ha più ignoti)

③ si preserva (per costruzione)

④ termina, e ogni passo $w-v$ decresce: $t = \text{n° ignoti} = w-v$

⑤ $w-v$ limitato? In generale no ma per Inv è limitato
perché vale $\text{INV} \wedge B$: $v \leq w$ (cioè no sotto)

Si puo' notare come ogni elemento viene guardato una volta sola