APA Modulo 1 Lezione 3

Elena Zucca

13 marzo 2020

Correttezza di algoritmi iterativi

Semplificando:

```
while (B)
   C
```

Cosa significa "corretto"?

```
//Pre: precondizione
while (B)
   C
//Post: postcondizione
```

corretto = se vale *Pre* all'inizio, allora (termina e) vale *Post* alla fine

Come provo la correttezza?

invariante di ciclo

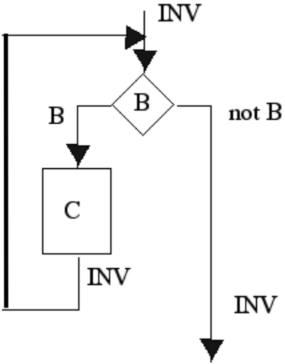
```
//Pre: Inv ∧ B
C
//Post: Inv
```

se devo eseguire il corpo e vale prima allora vale anche dopo

Quando un'invariante garantisce la correttezza del ciclo?

oltre alla proprietà di essere un'invariante:

- *Inv* vale all'inizio
- se valgono insieme *Inv* e $\neg$ *B* e allora vale *Post*



è facile vedere per induzione aritmetica sul numero di iterazioni che alla fine vale *Post*

La terminazione si prova a parte

- trovando una quantità t (funzione di terminazione) tale che, se devo eseguire il corpo, quindi se valgono Inv e B:
- viene decrementata strettamente eseguendo il corpo
- è limitata inferiormente

Esempio "didattico"

per ora trascuriamo la terminazione

```
//Pre: x = x0 ∧ y = y0
while (x!=0)
  x = x-1
  y = y+1
//Post: y = x0 + y0
```

Troviamo l'invariante che serve:

$x \geq 0$?

$x \leq x_0$?

$y \geq y_0$?

Soluzione: $x + y = x_0 + y_0$

Infatti:

```
//Pre: x = x_0 ∧ y = y_0
while (x!=0) \\Inv: x + y = x_0 + y_0
  x = x-1
  y = y+1
//Post: y = x_0 + y_0
```

- vale banalmente all'inizio
- se vale insieme a $\mathtt{x} = 0$, allora vale la postcondizione.
- se vale insieme a $\mathtt{x} \neq 0$ prima di eseguire il corpo, allora vale dopo

Per garantire la terminazione:

aggiungiamo una precondizione

```
//Pre: x = x₀ ∧ y = y₀ ∧ x ≥ 0
while (x!=0)
  x = x-1
  y = y+1
//Post: y = x₀ + y₀
```

L'invariante diventa:

$\mathtt{x} + \mathtt{y} = x_0 + y_0 \wedge \mathtt{x} \geq 0$

funzione di terminazione? il valore di `x`, infatti:

- viene decrementato a ogni passo
- assumendo *Inv* (quindi $\mathtt{x} \geq 0$), è limitato inferiormente da 0.

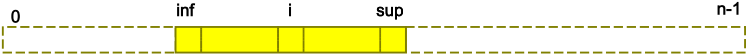
Metodologia

- l'invariante si ottiene in genere "approssimando" la postcondizione: in un passo generico ho risultato parziale
- come per l'induzione, serve non solo per provare la correttezza a posteriori, ma come guida allo sviluppo dell'algoritmo
- in molti algoritmi iterativi (vedremo esempi) l'idea iniziale è una buona invariante
- trovata l'invariante:
  - il corpo del ciclo deve preservarla e "avvicinarsi" alla soluzione
  - la condizione di controllo deve garantire la postcondizione in uscita
  - l'inizializzazione deve garantirla all'inizio

Esempio: ricerca binaria iterativa

idea per l'invariante: al passo generico il valore da cercare x, se presente, si trova nella porzione di array compresa fra due indici `inf` e `sup`:

$$x \in a[0..n-1] \Rightarrow x \in a[inf..sup]$$



Guidati da questa invariante:

Passo confronto x con l'elemento centrale $\texttt{a}[\texttt{mid}]$ della porzione $\texttt{a}[\texttt{inf}..\texttt{sup}]$, tre casi:

- $\texttt{x} < \texttt{a}[\texttt{mid}]$: `x`, se c'è, si trova nella porzione $\texttt{a}[\texttt{inf}..\texttt{mid} - 1]$, quindi `sup = mid-1`
- $\texttt{x} > \texttt{a}[\texttt{mid}]$: `x`, se c'è, si trova nella porzione $\texttt{a}[\texttt{mid} + 1..\texttt{sup}]$, quindi `inf = mid+1`
- $\texttt{x} = \texttt{a}[\texttt{mid}]$: ho trovato `x`, fine!

Condizione di controllo la porzione di array su cui effettuare la ricerca non deve essere vuota, quindi $\texttt{inf} \leq \texttt{sup}$

Inizializzazione inizialmente la porzione di array su cui effettuare la ricerca è l'intero array, quindi $\texttt{inf} = 0$ e $\texttt{sup} = n - 1$

Algoritmo completo

```
binary_search(x,a)
  inf = 0; sup = n-1 //Pre: inf = 0 ∧ sup = n − 1
  while (inf <= sup) //Inv: x ∈ a[0..n − 1] ⇒ x ∈ a[inf..sup]
    mid = (inf + sup)/2
    if (x < a[mid]) sup = mid-1
    else if (x > a[mid]) inf = mid+1
    else return true
  //Post: x ∉ a[0..n − 1]
  return false
```

Esempio: bandiera nazionale olandese

Problema proposto da Edsger Dijkstra

all'inizio (precondizione):

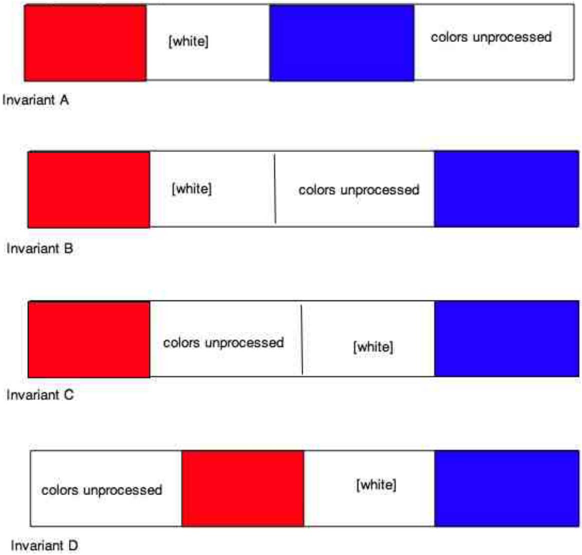
alla fine (postcondizione):

sequenza di n elementi rossi, bianchi e blu
operazioni che possiamo effettuare:

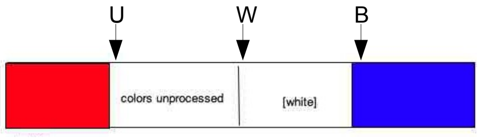
- `swap(i,j)` scambia di posto due elementi, per $1 \leq \mathtt{i}, \mathtt{j} \leq n$
- `colour(i)` restituisce `Red`, `White`, `Blue` per $1 \leq \mathtt{i} \leq n$

vogliamo inoltre esaminare ogni elemento *una volta sola*

Idea dell'invariante



Scegliamo versione C

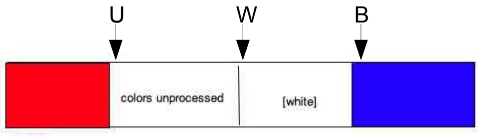


- U, W e B indici del primo elemento ignoto, bianco e blu
- quindi possiamo scrivere così l'invariante :

$$Red(1, U - 1) \wedge White(W, B - 1) \wedge Blue(B, n) \wedge U \leq W$$

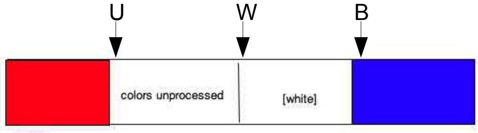
$Red(\mathtt{i}, \mathtt{j})$ = "rossi elementi da i a j compresi", analogamente $White(\mathtt{i}, \mathtt{j})$, $Blue(\mathtt{i}, \mathtt{j})$

A ogni passo



- considero ultimo elemento ignoto (se c'è) quindi `W-1`
- se è bianco: `W = W - 1`
- se è rosso: `swap(U, W-1); U = U + 1`
- se è blu: `swap(W-1, B-1); W = W-1; B = B-1`

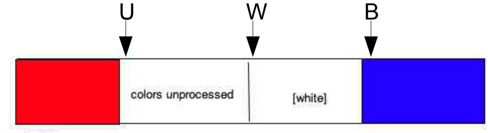
Quando è garantita la postcondizione?



- se non ci sono più ignoti, quindi $\mathtt{U} = \mathtt{W}$:
- infatti $\mathit{Red}(1, \mathtt{U} - 1) \land \mathit{White}(\mathtt{W}, \mathtt{B} - 1) \land \mathit{Blue}(\mathtt{B}, n) \land \mathtt{U} = \mathtt{W}$ diventa
- $\mathit{Red}(1, \mathtt{W} - 1) \land \mathit{White}(\mathtt{W}, \mathtt{B} - 1) \land \mathit{Blue}(\mathtt{B}, n)$

Precondizione?

deve garantire che valga *Inv* all'inizio



$$Red(1, \mathtt{U} - 1) \wedge White(\mathtt{W}, \mathtt{B} - 1) \wedge Blue(\mathtt{B}, n) \wedge \mathtt{U} \leq \mathtt{W}$$

- $\mathtt{U} = 1$
- $\mathtt{W} = \mathtt{B} = n + 1$

Algoritmo completo

```
//Pre: U = 1 ∧ W = B = n + 1
while (U<W) //Inv: Red(1,U − 1)∧White(W,B − 1) ∧ Blue(B,n) ∧ U ≤ W
    switch (Colour(W-1))
      case Red: swap(U, W-1); U= U + 1
      case White: W = W - 1
      case Blue: swap(W-1, B-1); W= W-1; B= B-1
//Post: Red(1,W − 1) ∧ White(W,B − 1) ∧ Blue(B,n)
```

Prova di correttezza:

- *Inv* vale all'inizio banalmente
- *Inv* $\wedge$ $\mathtt{U} \geq \mathtt{W}$ implica *Post* perché is ha $\mathtt{U} = \mathtt{W}$
- se vale *Inv* e $\mathtt{U} < \mathtt{W}$ eseguendo il corpo del ciclo vale ancora *Inv* (per casi)
- funzione di terminazione: $\mathtt{W} - \mathtt{U}$, infatti decresce eseguendo il corpo del ciclo in ognuno dei casi, e se vale la condizione di controllo è limitata inferiormente