Schemi di programma: ricerca e verifica

- Molti problemi riguardano la ricerca di elementi in intervalli o la verifica di proprietà.
- Sviluppiamo schemi di programma dimostrabilmente corretti che realizzano la ricerca e la verifica.
- La soluzione di problemi concreti consiste poi nella sostituzione di alcuni parametri degli schemi con valori specifici dei problemi in esame.
- Distinguiamo due tipi di ricerca: ricerca certa e ricerca incerta.
  - **ricerca certa**: si vuole determinare il minimo elemento di un intervallo [a,b) per il quale vale una certa proprietà $\mathcal{P}$, sapendo che almeno un elemento dell'intervallo soddisfa $\mathcal{P}$.
  - **ricerca incerta**: si vuole determinare, se esiste, il minimo elemento di un intervallo [a,b) per il quale vale una certa proprietà $\mathcal{P}$.

Ricerca certa

- Intervallo di ricerca: [a,b)
- Proprietà: $\mathcal{P}(\cdot)$
- Ipotesi di certezza: $\exists$ i $\in$ [a,b) . $\mathcal{P}$(i)
- Stato finale:
  x = min { i$\in$ [a,b) | $\mathcal{P}$(i)}.
- Lo schema generale per risolvere il problema è il seguente:

```
int x;
x=a;
while (!P(x))
  x=x+1;
```

- Nota: l'estremo destro dell'intervallo non serve.
- Si assume che la proprietà $\mathcal{P}$ sia esprimibile nel linguaggio.

- Proprietà invariante del ciclo:
  $x \in [a, b) \wedge (\forall\, j \in [a,x).\ \neg\, \mathcal{P}(j))$
- In altre parole, tutti gli elementi che precedono il valore corrente di x non soddisfano la proprietà $\mathcal{P}$.
- Se il ciclo termina, all'uscita dal ciclo vale la congiunzione
  $x \in [a, b) \wedge (\forall\, j \in [a,x).\ \neg\, \mathcal{P}(j)) \ \wedge\ \mathcal{P}(x)$
  che implica esattamente quanto espresso dallo stato finale:
  $x = \min\ \{\, i \in [a,b) \mid \mathcal{P}(i)\}$
- Osserviamo che l'invariante vale banalmente alla prima iterazione, con x=a.
  $a \in [a, b) \wedge (\forall\, j \in [a,a).\ \neg\, \mathcal{P}(j))$

- Verifichiamo che la proprietà
  $x \in [a, b) \wedge (\forall\, j \in [a,x).\ \neg\, \mathcal{P}(j))$
  è invariante per il ciclo:

```
x=a;
while (!P(x))
  x=x+1;
```

- Sia S uno stato in cui valgono le seguenti proprietà (x^S indica il valore di x in S)
  1. $x^S \in [a, b) \wedge (\forall\, j \in [a,x^S).\ \neg\, \mathcal{P}(j))$
  2. $\neg\, \mathcal{P}(x^S)$

  ovvero uno stato prima di una nuova iterazione del ciclo.
- 1. e 2. implicano ovviamente
  $(\forall\, j \in [a,x^S+1).\ \neg\, \mathcal{P}(j))$
- Se riusciamo anche a dimostrare che
  $x^S + 1 \in [a, b)$
  abbiamo dimostrato che la proprietà è invariante, dal momento che $x^S + 1$ è proprio il valore di x dopo la nuova iterazione.

- Sappiamo:
  $x^S \in [a, b)$
- Supponiamo per assurdo
  $x^S + 1 \notin [a, b)$ ovvero $x^S + 1 = b$ (*)
- Abbiamo appena dimostrato
  $(\forall\, j \in [a, x^S+1).\ \neg\, \mathcal{P}(j))$
  che insieme con (*) implica
  $(\forall\, j \in [a, b).\ \neg\, \mathcal{P}(j))$
- Ciò contraddice l'ipotesi di certezza
  $\exists\, i \in [a, b)\ .\ \mathcal{P}(i)$
- Dunque, dopo la nuova iterazione vale ancora la proprietà invariante.

Funzione di terminazione: Tra le tante . . . $b-x$

Ricerca certa: esempio 1

- Calcolare la radice intera di un numero naturale.
- Si può esprimere come problema di ricerca certa:
  $\lfloor \sqrt{N} \rfloor = \min\{x \in [0, N+1) \mid x^2 \leq N < (x+1)^2\}$
- Dunque l'estremo sinistro dell'intervallo di ricerca, a nello schema, in questo caso è 0, mentre l'estremo destro, b nello schema, è N.
- Infine la proprietà $\mathcal{P}(x)$ dello schema è $N < (x+1)^2$

```
int x;
x=0;
while ((x+1)*(x+1) <= N)
  x=x+1;
```

Ricerca certa: esempio 2

- Determinare la posizione della prima occorrenza di un dato elemento in un array, sapendo che tale elemento vi occorrre almeno una volta.
- Indichiamo con vet l'array e con DIM la sua dimensione
- Vogliamo determinare:
  $x = \min\{i \in [0,DIM) \mid vet[i] = el\}$
- Possiamo istanziare lo schema come segue:

```
int x;
x=0;
while (vet[x]!=el)
  x=x+1;
```


Ricerca Incerta

- Si vuole determinare, se esiste, il minimo elemento di un intervallo [a,b) per il quale vale una certa proprietà $\mathcal{P}$.
- Perché lo schema di ricerca certa non va bene?

```
x=a;
while (!P(x))
  x=x+1;
```

- Se l'elemento non c'è si vanno ad esaminare valori di x che sono al di fuori dell'intervallo di ricerca e per i quali la la proprietà $\mathcal{P}$ potrebbe addirittura non essere definita (errore a tempo di esecuzione).

  Esempio: Nel caso della ricerca incerta di un elemento in un array di dimensione `DIM` si andrebbero ad esaminare elementi del tipo `vet[x]` con `x > DIM`.
- Abbiamo bisogno di modificare lo schema in modo che l'analisi degli elementi avvenga solo all'interno dell'intervallo di ricerca e che la ricerca venga interrotta una volta esaurito l'intervallo (e non individuato alcun elemento).

Ricerca incerta

- Intervallo di ricerca: [a,b)
- Proprietà: $\mathcal{P}(\cdot)$
- Stato finale: $x = \min\{i \in [a,b) \mid \mathcal{P}(i)\}$ *min* b

  $\Longrightarrow$ dobbiamo stabilire quale valore calcolare se nessun elemento dell'intervallo soddisfa $\mathcal{P}$: una buona scelta è il valore b, che sicuramente non fa parte dell'intervallo.

Ricerca incerta

- Utilizziamo una variabile booleana `trovato` che fa da sentinella

  $\Longrightarrow$ impone l'uscita dal ciclo non appena si individua un elemento che soddisfa la proprietà
- in congiunzione con la sentinella, la guardia del ciclo assicura che l'intervallo di ricerca non sia esaurito

```
int trovato = FALSE; /* inizialmente false */
int x=a;
while (!trovato && x<b)
  if (P(x))
    trovato = TRUE; /*x soddisfa P */
  else
    x=x+1;
```

- Si suppone che le costanti `TRUE` e `FALSE` siano state definite opportunamente, ad esempio mediante le direttive

```
#define FALSE 0
#define TRUE 1
```

- Anche in questo caso possiamo stabilire una proprietà invariante del ciclo, questa volta un po' più complicata:

$$x \in [a,b] \ \wedge \ (\forall\, j \in [a,x).\ \neg\mathcal{P}(j)) \ \wedge \ \text{trovato} \Rightarrow \mathcal{P}(x)$$

- È facile vedere che i valori iniziali di x e trovato soddisfano banalmente l'invariante
- Inoltre, al termine del ciclo abbiamo due casi:
  1. trovato $=$ TRUE $\wedge$ x<b: l'invariante e questa condizione implicano $x \in [a,b) \ \wedge \ (\forall\, j \in [a,x).\ \neg\mathcal{P}(j)) \ \wedge \ \mathcal{P}(x)$
  2. trovato=FALSE $\wedge\ x \geq b$: l'invariante e questa condizione implicano $x = b \ \wedge \ (\forall\, j \in [a,b).\ \neg\mathcal{P}(j))$
- Dunque possiamo controllare l'esito della ricerca analizzando il valore di trovato

- La dimostrazione formale di invarianza della proprietà vista è lasciata per esercizio
- **Funzione di terminazione**: anche in questo caso qualcosa del tipo

$$b - x$$

sembra ragionevole.
- Il problema (formale) è che in un solo caso il valore di x non cresce (e dunque b−x non decresce) strettamente.
- L'individuazione di una corretta funzione di terminazione è lasciata per esercizio.

Ricerca incerta: esempio

- Determinare la prima occorrenza di un elemento in un array.
- È un problema di ricerca incerta:
  $\min \{x \in [0,DIM) \mid vet[x] = el\}$ *min DIM*

```
int trovato = FALSE;
int x=0;
while (!trovato && x<DIM)
  if (vet[x]==el)
    trovato = TRUE;
  else
    x=x+1;
```


- Vi sono situazioni in cui la proprietà $\mathcal{P}$ della ricerca (certa o incerta) non è direttamente esprimibile nel linguaggio.
  Esempio: Determinare (se c'è) la posizione del primo elemento di un array di interi che è uguale alla somma degli elementi che lo precedono.
- Si tratta di un problema di ricerca incerta in cui
  1. l'intervallo [a,b] è [0,DIM]
  2. la proprietà $\mathcal{P}(x)$ è

$$vet[x] = \sum_{j=0}^{x-1} vet[j]$$

```
int trovato = FALSE;
int x=0;
while (!trovato && x<DIM)
  if (vet[x]==sum_{j=0}^{x-1} vet[j])
    trovato = TRUE;
  else
    x=x+1;
```

- In questi casi si utilizza la seguente tecnica:
  1. si rimpiazzano le espressioni "critiche" con variabili
  2. si impone l'uguaglianza tra le variabili cosi' introdotte e le corrispondenti espressioni "critiche", aggiungendo quanto necessario al corpo del ciclo per mantenere vere tali uguaglianze
- si noti che formalmente 2. corrisponde a rafforzare opportunamente l'invariante.
- Nell'esempio:

```
int trovato = FALSE;
int x=0;
int sommaPrecedenti = 0;
while (!trovato && x<DIM)
  if (vet[x]==sommaPrecedenti)
    trovato = TRUE;
  else
    { sommaPrecedenti = sommaPrecedenti + vet[x];
      x=x+1;      }
```


- Quale è l'invariante del ciclo cosi' ottenuto?

$$\text{x} \in [0, \text{DIM}] \wedge (\forall\, \text{j} \in [0,\text{x}).\ \text{vet[j]} \neq \sum_{\text{k}=0}^{\text{j}-1} \text{vet[k]}) \ \wedge$$

$$\text{trovato} \Rightarrow \text{vet[x]} = \sum_{\text{k}=0}^{\text{x}-1} \text{vet[k]} \wedge$$

$$\text{sommaPrecedenti} = \sum_{\text{k}=0}^{\text{x}-1} \text{vet[k]}$$

- L'ultimo congiunto rappresenta il significato della variabile introdotta per esprimere la proprietà di ricerca $\mathcal{P}$.

Verifica di una proprietà

- Vogliamo verificare che tutti gli elementi di un intervallo soddisfano una certa proprietà $\mathcal{P}$.
  1. Facciamo una ricerca incerta del minimo elemento dell'intervallo per il quale non vale la proprietà $\mathcal{P}$
  2. Se non troviamo tale minimo, la verifica ha esito positivo, altrimenti ha esito negativo.
- Lo schema generale per risolvere questo problema.

```
int trovato = FALSE;
int x=a;
while (!trovato && x<b)
  if (!P(x))
    trovato = TRUE;
  else
    x=x+1;
if (trovato)
  /* esito negativo */
else
  /* esito positivo */
```