FONDAMENTI DI PROGRAMMAZIONE
ESERCITAZIONI
Corso di Laurea in MATEMATICA
a.a. 2014/15

Chiara Bodei

[1]Dipartimento di Informatica
email: chiara.bodei@unipi.it

Lo stato

- Insieme di associazioni tra nomi e valori.

  $x \rightsquigarrow 5$
- Nelle specifiche, si vogliono spesso indicare valori costanti ma generici. Utilizziamo le seguenti convenzioni:
  - usiamo lettere minuscole per i nomi simbolici utilizzati nello stato e negli algoritmi;
  - usiamo lettere maiuscole per indicare i valori generici.
- Laddove necessario, possiamo specificare condizioni al contorno sul dominio di appartenenza di tali valori.

  Esempio:

  $\{\text{naturale} \rightsquigarrow A, \text{base} \rightsquigarrow 2\}$ con $A \geq 0$

  Al nome simbolico naturale è associato un generico valore naturale ($A \geq 0$), mentre al nome simbolico base è associata il valore costante 2.

ATTENZIONE: all'interno degli algoritmi (in particolare nelle espressioni) non è possibile utilizzare esplicitamente le costanti generiche (proprio perché tali!).

Esempio: Area rettangolo
Specifica:
Stato iniziale: { base $\rightsquigarrow$ A, altezza $\rightsquigarrow$ B } con A>0 e B≥0
Stato finale: { area $\rightsquigarrow$ A*B}

Algoritmo: `area = base * altezza;` **OK!**

Algoritmo: `area = A * B;` **NO!**
L'algoritmo può solo manipolare i nomi simbolici. In questo caso, moltiplica qualsiasi valore associato a base a qualsiasi valore associato ad altezza e associa il risultato ad area.

Più precisamente

- Lo stato rappresenta il contenuto della memoria e il cambiamento dello stato si traduce nel cambiamento di alcune posizioni della memoria.
- Una posizione di memoria destinata a contenere dati, si identifica con una variabile. Il nome indica che i dati possono essere modificati nel corso dell'esecuzione.
- I nomi simbolici (ad es. base e altezza) sono quindi assimilabili alle variabili di programma. I valori ad essi associati possono cambiare.
- Per non confonderci, chiameremo variabili di specifica gli identificatori che servono per rappresentare valori generici (come A e B prima). Una volta fissato, il loro valore non cambia durante l'esecuzione dei comandi.

`area = A * B;` **NO!**
L'espressione non è lecita perché A e B rappresentano i generici valori di base e altezza nello stato iniziale.

Esempi

Esempio: Ordinare tre valori interi distinti tra loro

Stato iniziale: $\{x \rightsquigarrow A,\ y \rightsquigarrow B,\ z \rightsquigarrow C\}$ con A,B,C distinti tra loro

Stato finale: $\{\ x \rightsquigarrow \max(\{A,B,C\}),\ y \rightsquigarrow \max([\{A,B,C\} \setminus \{\max(\{A,B,C\})\}]),\ z \rightsquigarrow \min(\{A,B,C\})\}$

Algoritmo

Passo 1. "Ordiniamo" x e y, portandoci nello stato intermedio

Stato 1: $\{\ x \rightsquigarrow \max(A,B),\ y \rightsquigarrow \min(A,B),\ z \rightsquigarrow C\ \}$

Passo 2. "Ordiniamo" x e z, portandoci nello stato intermedio

Stato 2: $\{\ x \rightsquigarrow \max(\{A,B,C\}),\ y \rightsquigarrow \min(A,B),\ z \rightsquigarrow \min(\max(A,B),\ C)\ \}$

Passo 3. "Ordiniamo" y e z, portandoci nello stato finale desiderato

Soluzione nello pseudo-linguaggio

```
if (x < y)
{
    temp = x;
    x = y;
    y = temp;
}
  { x ⇝ max(A,B), y ⇝ min(A,B), z ⇝ C }
if (x < z)
{
    temp = x;
    x = z;
    z = temp;
}
  { x ⇝ max({A,B,C}), y ⇝ min(A,B), z ⇝ min(max(A,B), C) }
if (y < z)
{
    temp = y;
    y = z;
    z = temp;
}
  Stato finale
```

Possibile ottimizzazione

```
if (x < y)
{
    temp = x;
    x = y;
    y = temp;
}
  { x ⇝ max(A,B), y ⇝ min(A,B), z ⇝ C }
if (y < z)
{
    temp = y;
    y = z;
    z = temp;
  { x ⇝ max(A,B), y ⇝ C, z ⇝ min(A,B,C) }
  if (x < y)
      temp = x;
      x = y;
      y = temp;
  Stato finale }
  Stato finale
```


Esempio: Calcolo del fattoriale di un numero naturale.
Ricordiamo che:

$$n! = \begin{cases} 1 & \text{se } n = 0 \\ 1 \cdot 2 \cdot \ldots \cdot (n-1) \cdot n & \text{se } n > 0 \end{cases}$$

Specifica:

Stato iniziale: $\{n \rightsquigarrow N\}$ con $N \geq 0$
Stato finale: $\{n \rightsquigarrow N,\ fatt \rightsquigarrow N!\}$

- Attenzione: la specifica impone che il valore di n non deve cambiare!

Soluzione:

```
fatt = 1;
i = 1;
while (i <= n)
{
    fatt = fatt * i;
    i = i + 1;
}
```

- i viene chiamata variabile di controllo del ciclo, dato che l'esecuzione di ogni iterazione dipende dal valore di i, o contatore dato che viene incrementata di 1 alla fine di ogni iterazione

  $\Longrightarrow$ per ogni valore di i $\in$ [1,N] .....

- Ad ogni iterazione del ciclo, vale la seguente proprietà:

  $$\texttt{fatt} = \prod_{\texttt{j}=1}^{\texttt{i}-1} \texttt{j} = \text{(i-1)!} \qquad (\bullet)$$

- Al termine del ciclo, quando cioè i=N+1, la (•) diventa

  fatt = N!, ovvero lo stato finale desiderato.

Sequenze: nomi simbolici con indice

Consentiamo anche l'uso di nomi simbolici con indice, per rappresentare sequenze.

$$c[i] \rightsquigarrow 5$$

dove i è un valore naturale. Ad esempio:

$$\{i \rightsquigarrow K,\ c[0] \rightsquigarrow 0,\ \ldots,\ c[K-1] \rightsquigarrow 0\} \qquad K \geq 0$$

Nello stato sono presenti:

- un'associazione per il nome simbolico i, al quale è associato un valore naturale K.
- K associazioni per i nomi simbolici c[0], c[1], ..., c[K-1], a ciascuno dei quali è associato il valore 0.

All'interno degli algoritmi, consentiamo di riferire nomi simbolici con indice nella forma c[*exp*] dove *exp* deve essere una espressione a valori naturali.
Anche in questo caso *exp* non deve contenere valori generici (nell'esempio, non è lecito un assegnamento del tipo c[K-1] = 5, mentre è lecito c[i-1] = 5).

Esempio: Calcolare il numero di elementi pari in una sequenza data.
Stato iniziale: $\{dim \rightsquigarrow K, c[0] \rightsquigarrow V_0, \ldots, c[K\text{-}1] \rightsquigarrow V_{K-1}\}$ con $K > 0$
Stato finale: $\{count \rightsquigarrow \#\{\ j \mid j \in [0,K\text{-}1] \wedge\ V_j \text{ pari}\}\ \}$

- Dobbiamo calcolare

$$\sum_{\substack{0 \le j \le dim\text{-}1 \\ c[j] \text{ pari}}} 1$$

- Di nuovo, utilizziamo un ciclo controllato da una variabile di controllo i che assume tutti i valori nell'intervallo [0,dim-1].
- Facciamo in modo che, ad ogni iterazione, valga la seguente proprietà

$$count = \sum_{\substack{0 \le j \le i\text{-}1 \\ c[j] \text{ pari}}} 1$$

- Se, alla fine del ciclo, i=dim, abbiamo quanto desiderato e cioè

$$count = \sum_{\substack{0 \le j \le dim\text{-}1 \\ c[j] \text{ pari}}} 1$$

Soluzione:

```
count =  0;
i =  0;                          punto 1
while (i <= dim-1)
{
    if (c[i] % 2 == 0)
        count =  count + 1;
    i =  i + 1;
}
```

- Attenzione: l'istruzione i = i + 1 viene eseguita dopo l'istruzione if, indipendentemente dalla valutazione della condizione.
- Nello stato iniziale del while, al punto (1), count $\rightsquigarrow$ 0, i $\rightsquigarrow$ 0 e dunque

$$\sum_{\substack{0 \le j \le i\text{-}1 \\ c[j] \text{ pari}}} 1 = \sum_{\substack{1 \le j \le 0 \\ c[j] \text{ pari}}} 1 = 0 = count$$

- il corpo del while mantiene vera la proprietà

$$count = \sum_{\substack{0 \le j \le i\text{-}1 \\ c[j] \text{ pari}}} 1$$

Esempio: Calcolare il massimo di una sequenza data di interi.
Stato iniziale: $\{dim \rightsquigarrow K, c[0] \rightsquigarrow V_0, \ldots, c[K\text{-}1] \rightsquigarrow V_{K\text{-}1}\}$ con $K > 0$
Stato finale: $\{massimo \rightsquigarrow max(\{V_0, \ldots, V_{K\text{-}1}\})\}$

- Dobbiamo calcolare

$$max\{c[j] \mid j \in [0, dim\text{-}1]\}$$

- Di nuovo, utilizziamo un ciclo controllato da una variabile di controllo i che assume tutti i valori nell'intervallo [0,dim-1].
- Facciamo in modo che, ad ogni iterazione, valga la seguente proprietà

$$massimo = max\{c[j] \mid j \in [0, i\text{-}1]\}$$

Per ottenere questo ad ogni iterazione confrontiamo l'elemento corrente con il "massimo in carica".

- Se, alla fine del ciclo, i=dim, abbiamo quanto desiderato e cioè

$$massimo = max\{c[j] \mid j \in [0, dim\text{-}1]\}$$

Soluzione:

```
massimo = c[0];
i = 1;
while (i <= dim-1)
{          qui massimo = max{c[0], ..., c[i-1]}
    if (c[i] > massimo)
        massimo = c[i];
    i = i + 1;  anche qui massimo = max{c[0], ..., c[i-1]}
}
```

- Inizializzare la variabile massimo a 0 sarebbe sbagliato: non è detto che tutti i numeri della sequenza siano negativi.
- Attenzione: l'istruzione i = i + 1 viene eseguita dopo l'istruzione if, indipendentemente dalla valutazione della condizione. Comunque dobbiamo scorrere tutta la sequenza.
- Per calcolare il massimo di K elementi devo fare K-1 confronti, dato che sono proprio K-1 gli elementi che devono uscire perdenti.

Considerazioni generali

- In molti problemi è necessario operare allo stesso modo su tutti gli elementi di un intervallo dato

  per ogni $j \in [a,b]$ OP_j

  esempio: $\sum_{j=a}^{b} exp_j$

- in tutti questi casi si ricorre a cicli in cui una variabile di controllo permette di scandire tutto l'intervallo di interesse

```
i = a;
while (i <= b)
{
    ⟨operazione in funzione di i⟩
    i = i+1;
}
```


Esercizio Controllare se un valore dato appare in una sequenza data
Stato iniziale: $\{dim \rightsquigarrow K, val \rightsquigarrow V, c[0] \rightsquigarrow V_0, \ldots, c[K-1] \rightsquigarrow V_{K-1}\}$
Stato finale: $\{trovato \rightsquigarrow b = 1 \text{ se } \exists\ j \in [0,K-1] \wedge\ V_j = V, b = 0 \text{ altr.}\}$

- Di nuovo, utilizziamo un ciclo controllato da una variabile di controllo i che assume tutti i valori nell'intervallo [0,dim-1].
- Devo usare anche una variabile di controllo trovato che mi dica se l'elemento è stato trovato oppure no.
- Dal ciclo si esce quindi quando:
  - o ho appena trovato l'elemento cercato
  - o sono arrivato alla fine della sequenza.

```
i = 0; trovato = 0;
while ((i <= dim-1) && (trovato == 0))
{
    if (c[i] == val)
        trovato = 1;
    i = i + 1;
}
```

Caveat

- La doppia condizione ((i <= dim-1) && (trovato == 0)) corrisponde al fatto che si può uscire dal ciclo:
  - se sono arrivato alla fine della sequenza,
  - o se ho appena trovato l'elemento cercato.
- usare solo la prima non è scorretto **ma** risulta *inefficiente*. Ad es., in una sequenza di 10.000 elementi in cui il valore si trova nella prima posizione, non fermarsi appena lo si è trovato comporta che si facciano comunque tutti i 9999 confronti.
- uscire non appena si trova l'elemento cercato si può ottenere anche in altri modi, come ad esempio
  - usando una istruzione `break` o `return` che provocano l'uscita brutale dal ciclo
  - forzando il valore del contatore i ad assumere il valore dim per falsificare la prima condizione

  In entrambi i casi, viene pregiudicata la leggibilità del codice ed anche minata la prossibilità di fare analisi di correttezza. Per questo motivo, sono **fortemente sconsigliati**.