PROGRAMMAZIONE I (A,B) - a.a. 2017-18
I Verifica Intermedia – 3 Novembre 2017

Esercizio 1

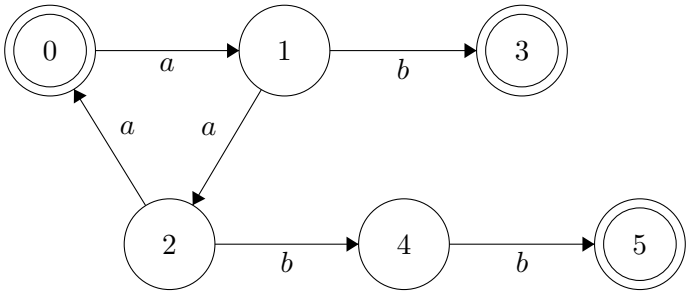
Dato il seguente linguaggio sull'alfabeto $\Sigma = \{a, b\}$

$$L = \{\ a^k b^m\ \mid\ k\%3 = m\ \ \wedge\ \ k \geq 0\ \}$$

si verifichi se è regolare o meno e si dia una grammatica che lo genera.

SOLUZIONE Il linguaggio è regolare. Si può infatti costruire un automa senza bisogno di "memorizzare" il numero delle a accettate, ma semplicemente ciclando tra tre stati corrispondenti ai tre risultati di $k\%3$. Da ognuno di questi stati partiranno transizioni per accettare 0, 1 o 2 simboli b rispettivamente.

Un possibile automa è il seguente:



La grammatica può essere definita traducendo l'automa come segue:

$$\begin{aligned} 0 &\to \epsilon \mid a1 \\ 1 &\to a2 \mid b \\ 2 &\to a0 \mid b4 \\ 4 &\to b \end{aligned}$$

In alternativa, un'altra possibile grammatica (più concisa) è la seguente:

$$S \to aaaS \mid \epsilon \mid ab \mid aabb$$

Esercizio 2

Data la seguente grammatica sull'alfabeto $\Sigma = \{a, b, c\}$

$$\begin{aligned} S &\to aSc \mid aAc \\ A &\to aAb \mid ab \end{aligned}$$

si dia il linguaggio generato dalla grammatica e si verifichi se tale linguaggio è regolare o meno.

SOLUZIONE Il linguaggio generato dalla grammatica è il seguente

$$L = \{\ a^k b^m c^p\ \mid\ k, m, p > 0\ \wedge\ k = m + p\ \}$$

Il linguaggio non è regolare. Per dimostrarlo usando il Pumping Lemma scegliamo, per un generico $n \in I\!N$, la stringa

$$w = a^{2n} b^n c^n$$

Con questa scelta, considerando tutte le possibili suddivisioni di w in xyz tali che $|xy| \leq n$ e $y \neq \epsilon$ abbiamo:

$$x = a^r$$
$$y = a^s$$
$$z = a^t b^n c^n$$

con $s > 0$, $r + s \leq n$ e $r + s + t = 2n$. A questo punto, considerando la stringa $xy^i z$ con $i = 0$ otteniamo $a^r a^t b^n c^n$, con $r + t < 2n$, che quindi non appartiene al linguaggio L.

Esercizio 3

Si scriva una funzione **C** che, dati due array a e b entrambi di dimensione dim, restituisca il valore di verità della seguente formula:

$$\forall i \in [0, dim - 1).\big(b[i] = \sum_{j \in [i+1, dim)} a[j]\big)$$

SOLUZIONE Seguendo pedissequamente la formula, definiamo innanzitutto una funzione ausiliaria che, dato un array a di dimensione dim e un intero $i \in [0, dim - 1)$, restituisce il valore di:

$$\sum_{j \in [i+1, dim)} a[j]$$

```
int sommasuccessivi(int a[], int dim, int i) {
  int somma = 0;
  for (int j=i+1; j<dim; j++)
    somma += a[j];
  return somma;
}
```

A questo punto, la formula richiesta dal testo dell'esercizio corrisponde alla seguente:

$$\forall i \in [0, dim - 1).\big(b[i] = sommasuccessivi(a, dim, i)\big)$$

il cui valore di verità può essere calcolato come segue:

```
int check(int a[], int b[], int dim) {
  int i = 0;
  int ok = 1;
  while (i<dim-1 && ok) {
    if (b[i] != sommasuccessivi(a,dim,i))
      ok=0;
    i++;
  }
  return ok;
}
```

Una soluzione alternativa, ma non modulare, prevede l'utilizzo di cicli annidati, come segue:

```
int check(int a[], int b[], int dim) {
  int i = 0;
  int ok = 1;
  while (i<dim-1 && ok) {
    int somma = 0;
    for (int j=i+1; j<dim; j++)
      somma += a[j];
    if (b[i] != somma)
      ok=0;
    i++;
  }
  return ok;
}
```

Soluzioni più efficienti possono essere basate sull'osservazione che in realtà abbiamo $b[i] = a[i+1] + a[i+2] + \cdots + a[dim-1]$ e $b[i+1] = a[i+2] + \cdots + a[dim-1]$. Quindi, i valori di a sommati per calcolare $b[i]$ sono in larga parte gli stessi usati per calcolare $b[i+1]$. Osservando che vale $b[i] = a[i+1] + b[i+1]$ possiamo definire una nuova soluzione come segue:

```
int check(int a[], int b[], int dim) {
  int i = 0;
  int ok = 1;
  while (i<dim-1 && ok) {
    if (b[i] != a[i+1] + b[i+1])
      ok = 0;
    i++;
  }
  return ok;
}
```

Seguendo lo stesso principio, la seguente funzione effettua il calcolo scandendo gli array al contrario e memorizzando la somma degli elementi in una variabile `somma` aggiornata ad ogni iterazione con l'aggiunta di un solo valore.

```
int check(int a[], int b[], int dim) {
  int i = dim-1;
  int somma = 0;
  int ok = 1;
  while (i>=0 && ok) {
    somma += a[i+1];
    if (b[i] != somma)
      ok = 0;
    i--;
  }
  return ok;
}
```

Esercizio 4

Si scriva una funzione **C** che, dato un array a di dimensione dim, restituisca il valore di verità della seguente formula:

$$\exists i \in [0, dim).\big(a[i] = \#\{\ j \mid j \in [0, dim)\ \wedge\ a[j] = a[i]\ \}\ \big)$$

dove $\#\{\ldots\}$ rappresenta il numero degli elementi (cardinalità) dell'insieme.

SOLUZIONE Modularmente, seguendo la struttura della formula possiamo definire una funzione per calcolare, dati a, dim e i, il valore di

$$\#\{\ j \mid j \in [0, dim)\ \wedge\ a[j] = a[i]\ \}$$

come segue:

```
int contauguali(int a[], int dim, int i) {
  int cont=0;
  for (int j=0; j<dim; j++)
    if (a[j]==a[i]) cont++;
  return cont;
}
```

A questo punto, la formula proposta dall'esercizio risulta essere equivalente alla seguente:

$$\exists i \in [0, dim).\big(a[i] = contauguali(a, dim, i)\ \big)$$

il cui valore di verità può essere calcolato come segue:

```
int check(int a[], int dim) {
  int i=0;
  int trovato=0;
  while (i<dim && !trovato) {
    if (a[i]==contauguali(a,dim,i))
      trovato=1;
    i++;
  }
  return trovato;
}
```

Una soluzione alternativa, non modulare, è la seguente:

```
int check(int a[], int dim) {
  int i=0;
  int trovato=0;
  while (i<dim && !trovato) {
    int cont=0;
    for (int j=0; j<dim; j++)
      if (a[j]==a[i]) cont++;
    if (a[i]==cont))
      trovato=1;
    i++;
  }
  return trovato;
}
```

Un'altra soluzione ancora, un po' più efficiente, può prevedere di interrompere il ciclo interno quando la variabile *cont* supera il valore di $a[i]$:

```
int check(int a[], int dim) {
  int i=0;
  int trovato=0;
  while (i<dim && !trovato) {
    int cont=0;
    int j=0;
    int superato=0;
    while (j<dim && !superato) {
      if (a[j]==a[i]) cont++;
      if (cont>a[i]) superato=1;
      j++;
    }
    if (a[i]==cont))
      trovato=1;
    i++;
  }
  return trovato;
}
```