PROGRAMMAZIONE I (A,B) - a.a. 2017-18
II appello – 6 Febbraio 2018

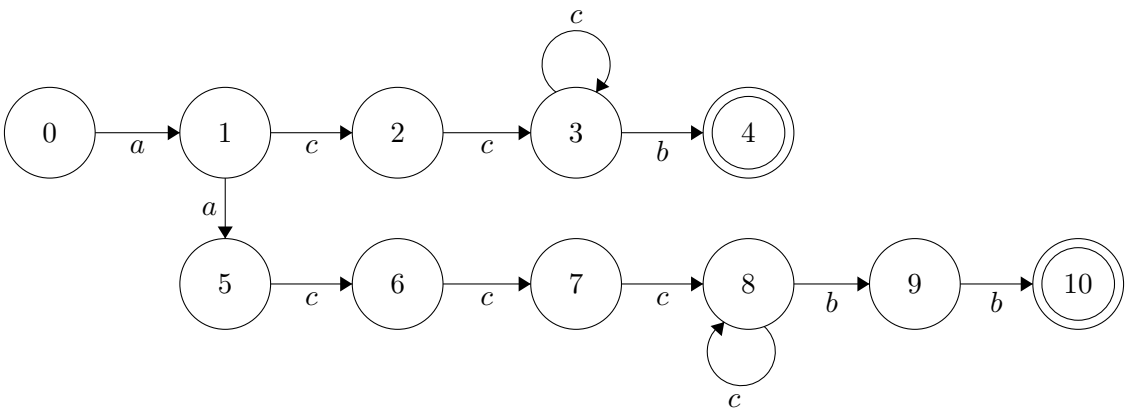
Esercizio 1

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

$$L = \{a^n c^m b^n \mid 0 < n < 3 \ \wedge \ n < m\}$$

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

SOLUZIONE Il linguaggio è regolare. Abbiamo infatti che n può assumere solo i valori 1 o 2, quindi un insieme finito di valori diversi che possono corrispondere ad un insieme finito di stati di un automa. Un automa che lo accetta è il seguente:



Una grammatica per il linguaggio può essere definita traducendo l'automa a stati finiti oppure, più semplicemente, come segue:

$$S \to acCb \mid aaccCbb$$
$$C \to c \mid cC$$

Esercizio 2

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

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

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

SOLUZIONE La formula chiede di verificare se esiste un elemento di a che è presente tante volte in a quante in b.

Modularmente, seguendo la struttura della formula possiamo definire una funzione per calcolare, dato un generico array c di dimensione dim, e un valore x

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

come segue:

```
int conta(int c[], int dim, int x) {
  int cont = 0;
  for (int i=0; i<dim; dim++)
    if (c[i]==x) cont++;;
  return cont;
}
```

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

$$\exists i \in [0, dima).\big(conta(a, dima, a[i]) = conta(b, dimb, a[i])\big)$$

che possiamo verificare come segue:

```
int check(int a[], int b[], int dima, int dimb) {
  int trovato=0;
  int i=0;
  while (i<dima && !trovato) {
    if (conta(a,dima,a[i]==conta(b,dimb,a[i])
      trovato=1;
    else
      i++;
  }
  return trovato;
}
```

Esercizio 3

Si definisca in CAML una funzione ricorsiva in modo esplicito

`zerouno : int list -> bool`

che, data una lista di interi, restituisce `true` se la lista contiene solo occorrenze di 0 e 1, e nella lista ci sono tanti 0 quanti 1. La funzione restituisce `false` altrimenti.

SOLUZIONE Una possibile soluzione è la seguente: Una possibile soluzione consiste nel definire due funzioni ausiliare: una che verifica se la lista contiene solo occorrenze di 0 e 1, e l'altra che, dato un valore n, conta quante volte esso è presente nella lista.

```
let zerouno lis =
  let rec solo01 l =
    match l with
        [] -> true
      | x::xs -> if x<>0 && x<>1 then false
                 else solo01 xs
  in
  let rec conta l n =
    match l with
        [] -> 0
      | x::xs -> if x=n then 1 + conta xs n
                        else conta xs n
  in
    solo01 lis && conta lis 0 = conta lis 1;;
```

Esercizio 4

Si definisca in CAML, senza usare la ricorsione esplicita, una funzione

`zerouno_ordinati : int list -> bool`

che, data una lista di interi, restituisce `true` se la lista contiene solo occorrenze di 0 e 1 ed è ordinata in modo non decrescente (prima tutti gli 0 e poi tutti gli 1). La funzione restituisce `false` altrimenti.

SOLUZIONE Una possibile soluzione è la seguente, in cui la funzione ausiliaria passata all `foldr` utilizza due booleani `ok` e `trovato`:

```
let zerouno_ordinati lis =
  let f x (ok,trovato) =
      if ((not ok) || (x<>0 && x<>1)) then (false,trovato)
      else if x=1 && not trovato then (ok,trovato)
           else if x=1 && trovato then (false,trovato)
                else (ok,true)
  in
    let (a,b) = foldr f (true,false) lis
  in
    a;;
```

L'idea è che `trovato` parte `false` e, nello scandire la lista dalla coda alla testa, diventa `true` non appena viene incontrato uno 0 (da quel momento in poi tutti gli elementi incontrati dovranno essere 0). Invece, `ok` parte `true` e diventa `false` se viene incontrato un valore diverso da 0 o 1, o se viene incontrato un 1 quando `trovato` è `true`.

Una soluzione più semplice, ma meno efficiente, consiste nell'utilizzare due volte la `foldr`, una volta per controllare che tutti gli elementi siano 0 o 1, e una seconda per controllare che gli elementi siano ordinati.