PROGRAMMAZIONE 1 e LABORATORIO (A,B) - a.a. 2014-2015
Prova scritta del 9/02/2015

SOLUZIONI PROPOSTE

Per ogni esercizio vengono proposte una o più soluzioni tra le tante possibili.

ESERCIZIO 1 (6 punti)
Dato l'alfabeto $\Lambda = \{a, b, c, d\}$ si definisca una grammatica libera che genera il seguente linguaggio:

$$\mathcal{L} = \left\{ ab^n c^k d^m \;\mid\; k, n, m > 0 \wedge m > n \right\}$$

Si dimostri, utilizzando il pumping lemma, che tale linguaggio non è regolare.

Soluzione

```
S ::= aB
B ::= bBd | bCDd
D ::= d | dD
C ::= c | cC
```

Per dimostrare che il linguaggio non è regolare utilizzando il pumping lemma mostriamo quanto segue:

$$\forall n \in \mathbb{N}.\ \exists w \in \mathcal{L}.\mid w \mid \geq n \wedge \forall x, y, z. \neg\big(w = xyz \ \wedge\ y \neq \epsilon \wedge\ \mid xy \mid \leq n \ \wedge\ (\forall i \in \mathbb{N}. xy^i z \in \mathcal{L})\big)$$

ovvero

$$\forall n \in \mathbb{N}.\ \exists w \in \mathcal{L}.\mid w \mid \geq n \wedge \forall x, y, z. \big(w = xyz \ \wedge\ y \neq \epsilon \wedge\ \mid xy \mid \leq n) \Rightarrow\ \neg(\forall i \in \mathbb{N}. xy^i z \in \mathcal{L})\big)$$

Sia allora $n \in \mathbb{N}$ e sia

$$w = ab^{n-1}c^k d^n$$

con $k > 0$. Chiaramente $w \in \mathcal{L}$ e $\mid w \mid \geq n$. Siano dunque x, y, z tali che:

(a) $w = xyz$

(b) $y \neq \epsilon$

(c) $\mid xy \mid \leq n$

Ragioniamo per casi:
(1) $x = \epsilon \ \wedge\ y = ab^h$, con $0 \leq h \leq n-1$ (e in questo caso $z = b^{(n-1-h)}c^k d^n$).
La stringa $xy^0z = z = b^{(n-1-h)}c^k d^n \notin \mathcal{L}$ c.v.d.

(2) $x = ab^h \ \wedge\ y = b^t$, con $t > 0$ e $h + t \leq n-1$ (e in questo caso $z = a^{(n-1-h-t)}c^k d^n$). Sia allora $s \in \mathbb{N}$ tale che $h + t \cdot s \geq n$
La stringa $xy^s z = ab^{h+t\cdot s}c^k d^n\$ \notin \mathcal{L}$ c.v.d.

ESERCIZIO 2 (6 punti)
Si definisca in CAML, senza utilizzare ricorsione esplicita, una funzione

```
pfirst : (‘a -> bool) -> ‘a list -> ‘a list
```

in modo che `(pfirst p lis)` restituisca una lista con gli stessi elementi di `lis`, in cui tutti gli elementi che soddisfano `p` precedono tutti quelli che non soddisfano `p`.

Soluzione

```
let pfirst p l =
   let nonp x = not (p x)
in
   (filter p l) @ (filter nonp l)
```

Una seconda soluzione si ottiene utilizzando la funzione di ordine superiore `foldr` come segue:

```
let pfirst p l =
    let f x y = if (p x) then x::y else y @ [x]
in
    foldr f [] l
```

ESERCIZIO 3 (6 punti)
Si scriva in `C` una procedura che, dati attraverso opportuni parametri una lista di interi ed un intero `n`, cancella dalla lista, se c'è, l'ultimo elemento che contiene `n`.
Si suppongano predefiniti i tipi

```
struct el {int info; struct el *next;};

typedef struct el ElementoDiLista;
typedef ElementoDiLista *ListaDiInteri;
```

Soluzione

```
void canc (ListaDiInteri *l, int n)
{
   if (*l != NULL)
   {
     ListaDiInteri prec=NULL, corr=*l, save=NULL;
     int trovato = 0;
     while (corr !=NULL)
       {
         if (corr->info == n)
            { save=prec;
              trovato = 1; }
         prec = corr;
         corr = corr->next;
       }
     if (trovato)
        {
          if (save==NULL)
           {corr =*l; *l=*l -> next;}
          else
           {corr = save->next; save->next=save-next->next;}
          free(corr);
        }
   }
}
```

ESERCIZIO 4 (6 punti)
Si scriva una funzione `C`

```
int check (int a[], int dim)
```

che, dato un array `a` e la sua dimensione `dim`, restituisce il valore di verità della seguente formula

$$\#\{i \mid i \in [0, dim-1) \wedge (\exists j \in [i+1, dim).\ a[i] = a[j])\} \geq 2$$

Si ricorda che, dato un insieme finito A, $\#A$ denota il numero di elementi di A.

Soluzione

```
int check (int a[], int dim)
{
   int conta=0;
   int i=0;
   while (i < dim-1 && conta<2)
   {
     int esiste = 0, j = i+1;
     while (j<dim && !esiste)
       if (a[i]==a[j])
         esiste = 1;
       else
         j=j+1;
     if (esiste) conta=conta+1;
     i = i+1;
   }
   return (conta>=2);
}
```

ESERCIZIO 5 (6 punti)
Una lista di coppie $(valore, numero)$, in cui *valore* compare nella lista al più una volta e $numero > 0$, può essere utilizzata per rappresentare un multinsieme (ovvero una struttura in cui gli elementi, contrariamente agli insiemi, possono essere ripetuti). Ad esempio la lista di tipo `(string * int) list`

```
[("abc", 1); ("x", 3); ("pq", 2)]
```

può rappresentare il multinsieme di stringhe

`{"x", "abc", "x", "pq", "pq", "x"}`.

Scrivere in `CAML` una funzione

```
canc : ('a * int) list -> 'a ->  ('a * int) list
```

in modo che `(canc m el)` restituisca una lista che rappresenta il multinsieme ottenuto da quello rappresentato da `m` cancellando **una sola occorrenza** di `el` (se quest'ultimo occorre in `m`).

Soluzione

```
let rec canc m el =
   match m with
        [] -> [] |
        (x,n)::ms when x=el -> if (n=0) then ms else (x,n-1)::ms |
        (x,n)::ms when x<>el -> (x,n):: (canc ms el);;
```