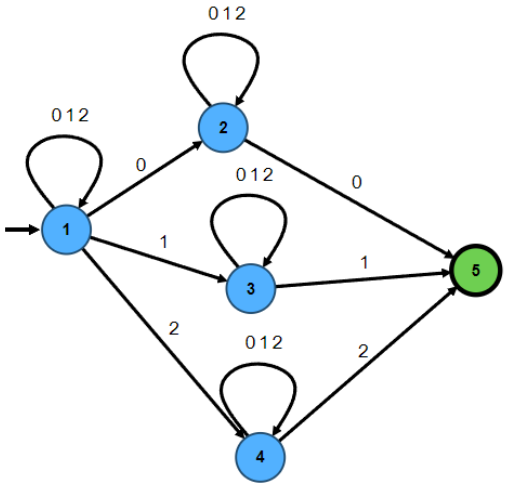
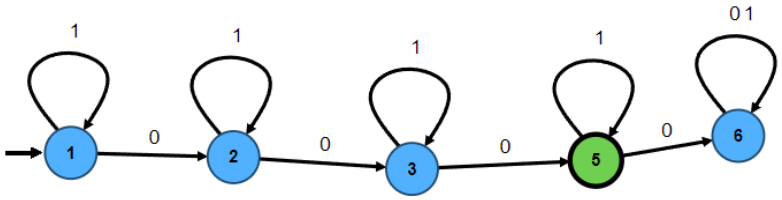
Primo set di esercizi Automata Tutor

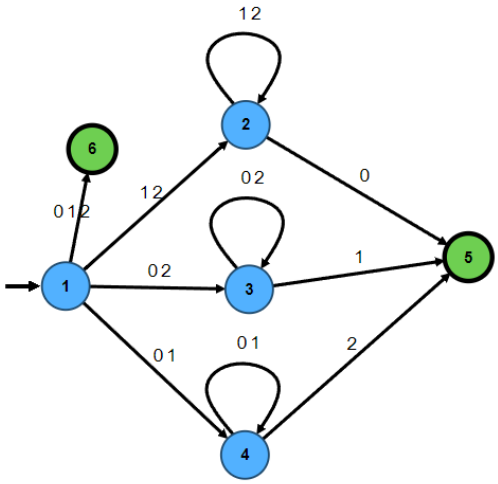
1) *Automa NFA con alfabeto {0,1,2} che ha come linguaggio:*
   la cifra finale sia comparsa in precedenza



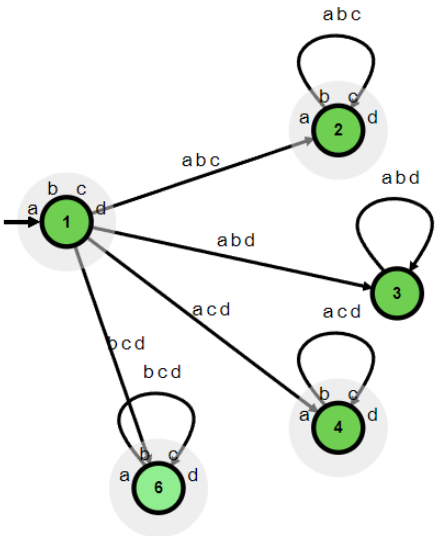
2) *Automa DFA con alfabeto {0, 1} che ha come linguaggio:*
   tutte e sole le stringhe che contengono esattamente tre zeri (anche non consecutivi)



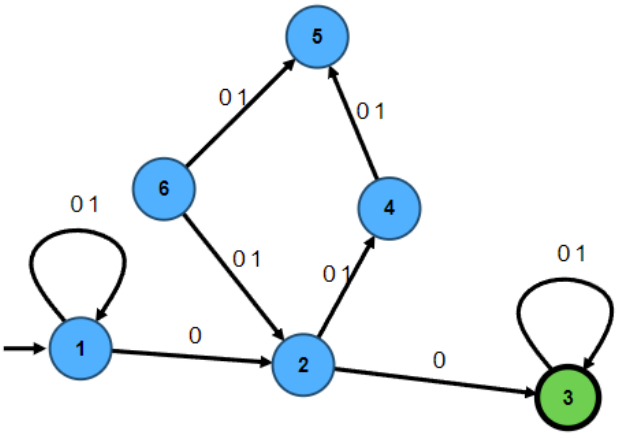
3) *Automa NFA con alfabeto {0,1,2} che ha come linguaggio le stringhe in cui:*
   la cifra finale non sia comparsa in precedenz



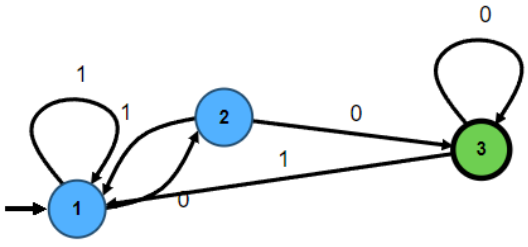
4) *Automa NFA con alfabeto {a, b, c, d} che ha come linguaggio le stringhe in cui:*
   uno dei simboli dell'alfabeto non compare mai



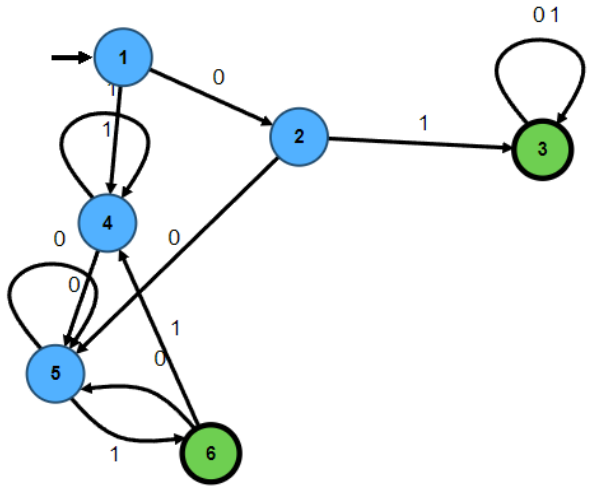
5) *Automa NFA con alfabeto {0, 1} che ha come linguaggio le stringhe in cui:*
esistono due 0 separati da un numero di posizioni multiplo di 4



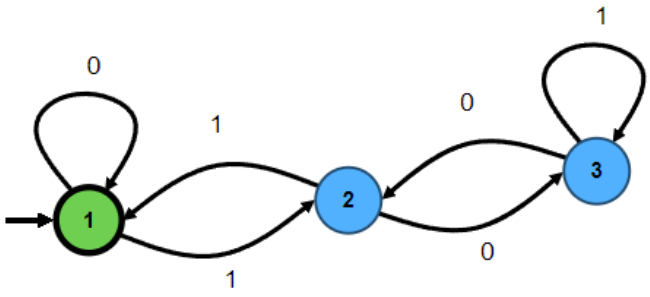
6) *Automa DFA con alfabeto {0, 1} che ha come linguaggio:*
tutte e sole le stringhe che terminano con 00



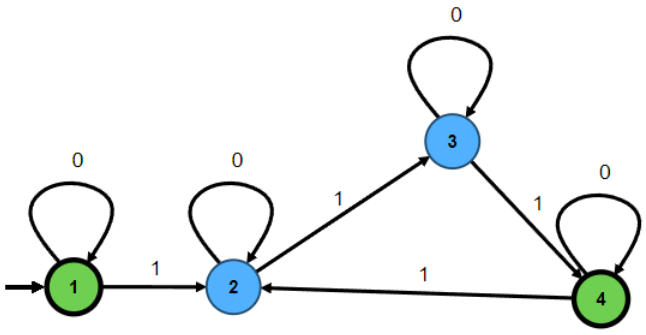
7) *Automa DFA con alfabeto {0, 1} che ha come linguaggio:*
tutte e sole le stringhe che cominciano o finiscono con 01 (o entrambe le cose)



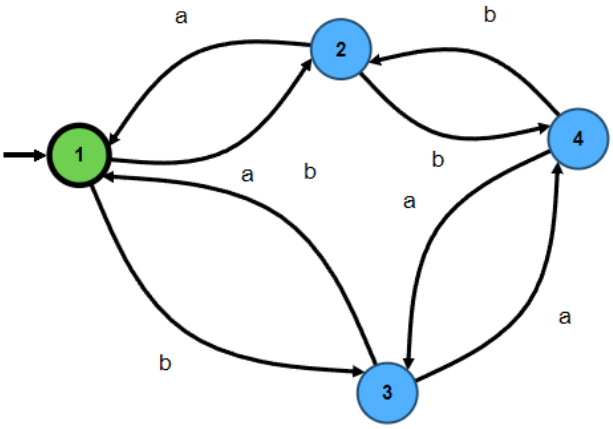
8) *Automa DFA con alfabeto {0, 1} che ha come linguaggio:*
tutte le stringhe che rappresentano la codifica binaria di un numero multiplo di 3. La stringa vuota non rappresenta nessun numero.



9) *Automa DFA con alfabeto {0, 1} che ha come linguaggio:*
tutte e sole le stringhe che un numero di 1 multiplo di 3



10) Automa DFA con alfabeto {0, 1} che ha come linguaggio:
tutte e sole le stringhe con un numero pari di "a" e di "b"



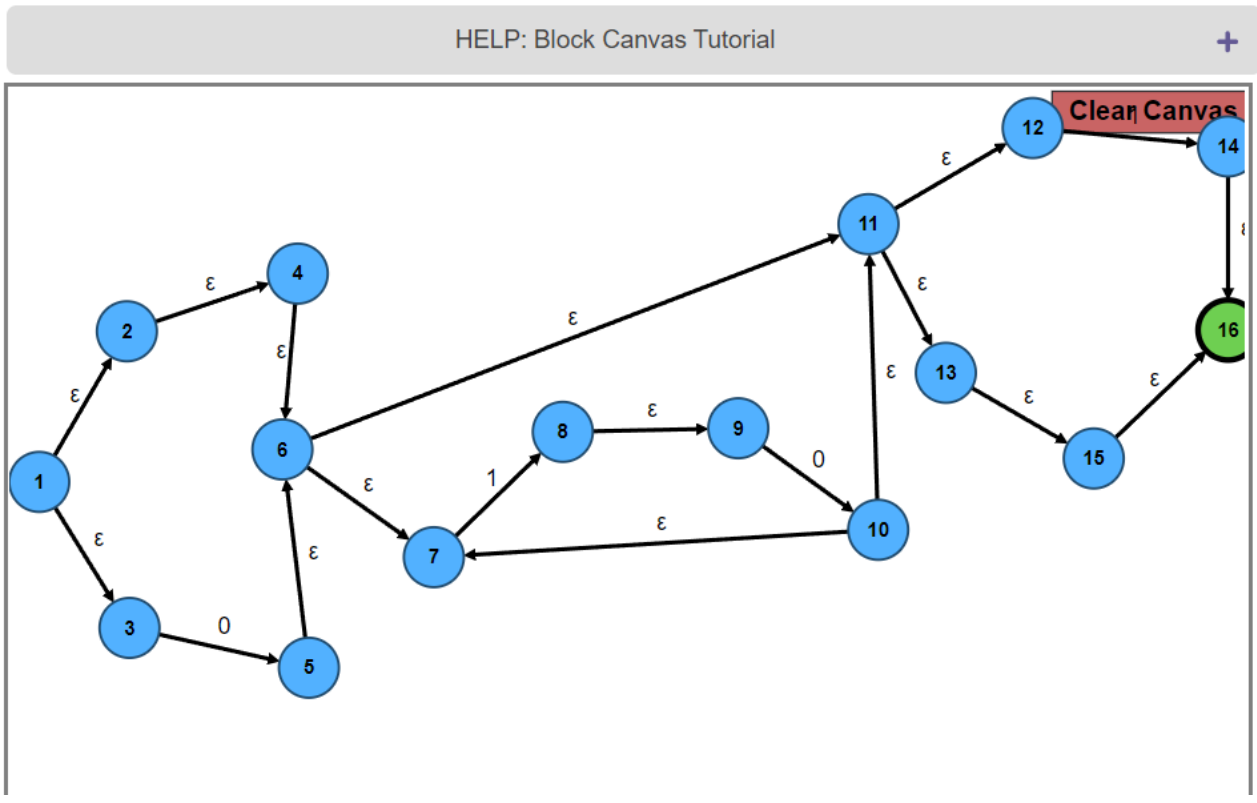
For the following regular expression:

$$(\epsilon|0)(10)^*(1|\epsilon)$$

Over the alphabet:

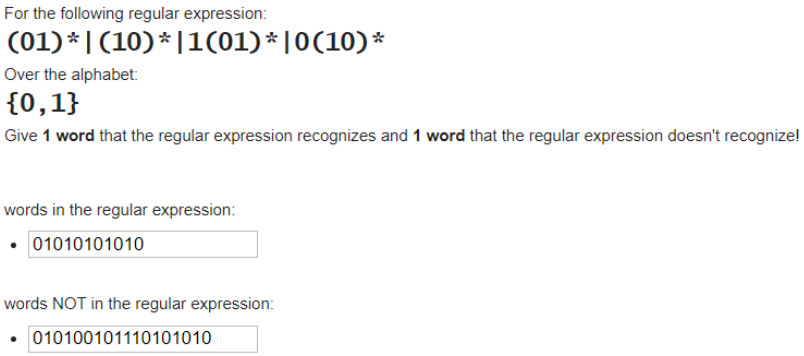
$$\{0,1\}$$

Give an epsilon-NFA that recognizes the same language.



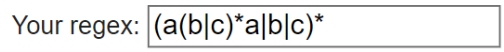
Secondo set di esercizi di Automata Tutor

1) Set di 0/1 alternati {alfabeto 0,1}



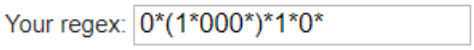
Ragionamento:
Chiede 0/1 alternati e si nota che la terza e quarta espressione *unite* (segno di unione per Automata è |)
sono sinonime della prima, da cui il risultato.

2) Tutte le stringhe che contengono un numero pari di “a” (alfabeto {a,b,c})



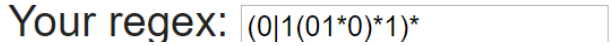
Ragionamento:
Posso scegliere “a” concatenato a b oppure c in tutte le combinazioni. A questo punto, posso concatenare un altro a oppure b oppure c, in tutte le combinazioni.

3) Tutte le stringhe che NON contengono la sottostringa 101 (alfabeto {0,1})



Ragionamento:
In poche parole, copre tutti i casi. Avendo la stringa 0 come iniziale, allora può essere seguita da un 1 e una serie di 0 (almeno uno se non di più).
Se la stringa è vuota (non è 0), passa al pezzo dopo, dove 1 può essere seguito da 0 oppure da 11. Anche i casi in cui una stringa cominci e finisca per 0 sono coperti dai casi esterni alle parentesi.
La parte tra parentesi ha bisogno di 3 zeri per separare tutti gli 1 (1*) dal fatto di avere tutti 0 (0*), quindi avendo anche 100 (con 1*, 0* e 0). Se fosse vuota, non avendo 1000 tra parentesi, accetterebbe anche 101 e non andrebbe più bene.

4) Tutte le stringhe che rappresentano una codifica binaria di un numero multiplo di 3 (alfabeto 0,1)



Ragionamento:
Dà 8/10, perché dice che non dovrebbe accettare anche la stringa vuota ma dopo numerose prove è l’unica soluzione che ho trovato. Si parta dall’automa, secondo me chiarisce piuttosto che provare alla cieca.
Si nota quindi l’unione iniziale di 0 ed 1. A questo punto si passa con un altro zero e successivamente con ciclo di 1 (1*) alla fine, avendo un altro 0 che torna indietro. Tutto questo è ripetuto più volte. Oltre a questo abbiamo un altro 1 che riconduce all’inizio, che è stato finale.
Da cui tutta sta roba.

5) Tutte le stringhe che contengono 4k + 1 occorrenze di “b” per “k” >= 0
Soluzione:
((a|c)*b(a|c)*b(a|c)*b(a|c)*b(a|c)*)* (a|c)*b(a|c)*
Ragionamento:
Provandole tutte, perché deve avere al minimo un solo b e al più almeno 1+4, la ripetizione di Kleene degli altri simboli è sensata tenerla ripetuta come si vede qui, quindi una per le 4 che sono ripetute di volta in volta. A quel punto, fuori vi sarà la b in più e due volte le espressioni a|c; per qualche ragione, il prof vuole entrambe le possibilità.

6) Tutte le stringhe w che contengono la sottostringa 101 (alfabeto 0,1)

Your regex: (0|1)*101(0|1)*

Ragionamento:
Questo è il più easy; basta proprio avere 101 e poi qualsiasi cifra prima e dopo e va bene.

7) Tutte le stringhe la cui lunghezza è multiplo di 3 (alfabeto a,b,c)

Your regex: ((a|b|c)(a|b|c)(a|b|c))*

Ragionamento:
Qui pure è abbastanza easy, di fatto il multiplo di 3 presuppone che tutto appaia tre volte, in termini appunto di a,b,c. Questo conduce alla soluzione.

Solve Regular expression to epsilon-NFA problem

For the following regular expression:

(ϵ|0)(10)*(1|ϵ)

Over the alphabet:

{0,1}

Give an epsilon-NFA that recognizes the same language.

In questo tipo di esercizi bisogna fare le cose a singoli pezzi:
si parte dall'unione ε|0 dopodiché attenzione alla concatenazione 10 e poi lo *.
Bisogna mettere una ε prima di 1, una ε dopo 1, si mette 0, dopodiché lo stato dopo va ad ε, quello stato torna ad 1 per fare lo * e lo stato prima di 1 (ε) va direttamente allo stato dopo lo 0.
Segue poi l'unione 1| ε realizzata come prima.

For the grammar G with the productions:

```
S -> A | I | E
I -> if cond then S
E -> if cond then S else S
A -> a
```

Find a **leftmost** derivation for the word:

if cond then if cond then a else s

HELP: Derivation Syntax +

```
S
I
if cond then S
if cond then E
if cond then if cond then S else S
if cond then if cond then A else S
if cond then if cond then a else S
if cond then if cond then a else s
```

Find a grammar that recognizes the following language:

a^i b^j c^k tali che i = j oppure j = k

```
S->X|Y
X->Xc|A|\eps
A->aAb|\eps
Y->aY|B|\eps
B->bBc|\eps
```

Find a grammar that recognizes the following language:

tutte le parole w che contengono un numero pari di 'a' e dispari di 'b'

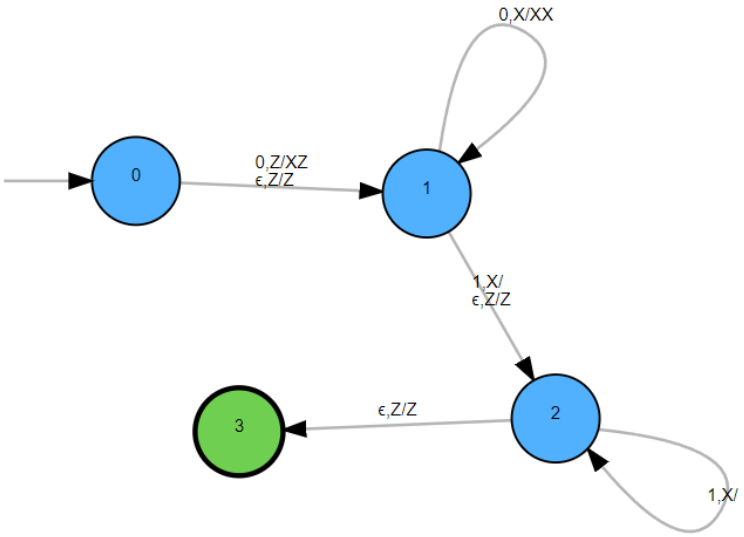
```
S->aT|bR
T->aS|bU
R->bS|aU|_
U->aR|bT
```

Construct a PDA that recognizes the following language:

{0^n 1^n | n >= 0}

Hint: To get the maximum number of points, use as few states and nonterminals as possible!

- **Alphabet:** {0, 1}
- **Stack alphabet (the first symbol is the initial one):** Z X apply
- **Acceptance condition:** final state
- **Deterministic (DPDA):** false



Find a grammar that recognizes the following language:

a^n b^n tale che n >= 0

HELP: Grammar S

```
S->aSb|\eps
```