Regular Languages and Automata: Problems for Week 1

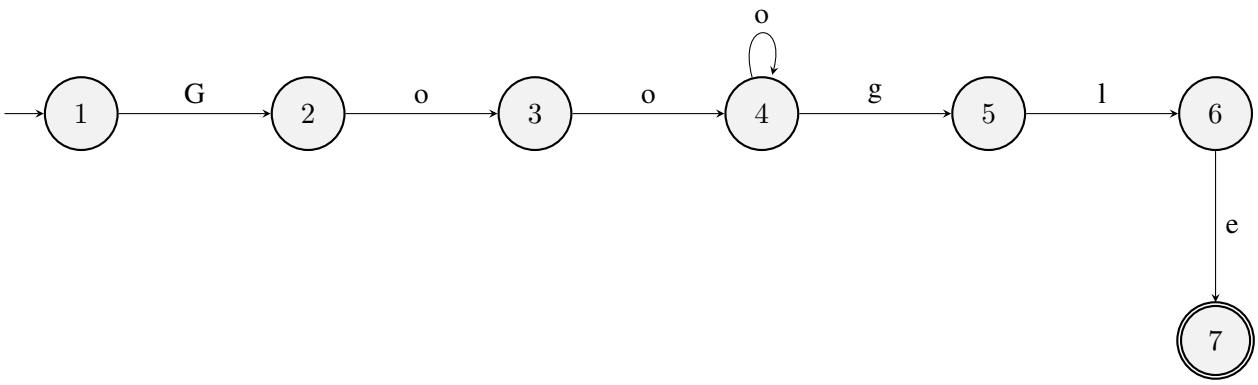
Note: when we ask for a DFA, we are happy for you to supply a partial DFA. Indeed that's usually better, because it's more efficient.

Exercise 1. *Give a regexp over the alphabet $\Sigma = \{a, b, c\}$ for the set of words in which "a" occurs precisely twice.*

Solution $(b|c)^*a(b|c)^*a(b|c)^*$

Exercise 2. *Build a DFA that checks whether a string is equal to "`Goo....gle`" with arbitrarily many `o`'s following the initial two.*

Solution

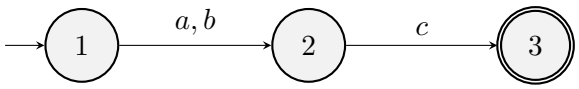


Exercise 3. *Design DFAs for the following regular expressions:*

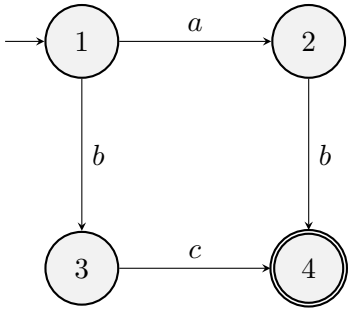
1. `(a|b)c`
2. `ab|bc`
3. `ab|ac` *(Careful! Remember that from any state there must be at most* one *transition labelled with a particular letter.)*
4. `c(a|b)*c`

Solution

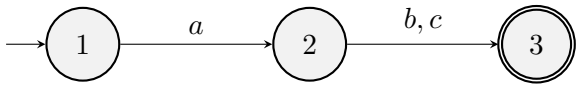
1.



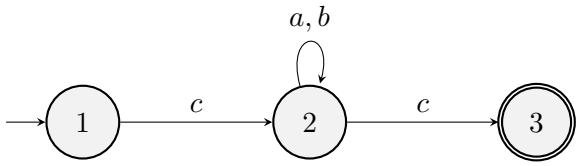
2.



3.



4.



Exercise 4. *An online shop requires users to provide a password during registration. Every password is a string of lowercase letters and digits. It must contain at least one letter and at least one digit, and it must be at least three characters long. Give a regular expression for passwords. You can use* `[a-z]`*, which matches any lowercase letter, and* `[0-9]`*, which matches any digit.*

Solution

One way of solving this problem is to divide it into cases. Clearly there are two cases, passwords beginning with a letter and passwords beginning with a digit. A password beginning with a letter consists of four parts:

- a letter
- then a (possibly empty) string of letters
- then a digit—i.e. the first digit appearing in the password
- then a (possibly empty) string of letters and digits.

The problem is that the second and fourth part can't both be empty because then the password would be only two characters long. So there are two acceptable sub-cases:

- passwords in which the second part is empty and the fourth part is not:

```
[a-z][0-9]([a-z]|[0-9])([a-z]|[0-9])*
```

- passwords in which the second part is not empty and the fourth part has any length:

```
[a-z][a-z][a-z]*[0-9]([a-z]|[0-9])*
```

Putting together these two sub-cases we obtain an expression for passwords beginning with a letter:

```
[a-z][0-9]([a-z]|[0-9])([a-z]|[0-9])*  |
[a-z][a-z][a-z]*[0-9]([a-z]|[0-9])*
```

In the same way, we obtain an expression for passwords beginning with a digit:

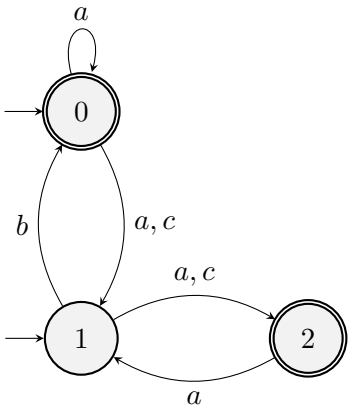
```
[0-9][a-z]([a-z]|[0-9])([a-z]|[0-9])*  |
[0-9][0-9][0-9]*[a-z]([a-z]|[0-9])*
```

Putting together the two kinds of passwords, we obtain:

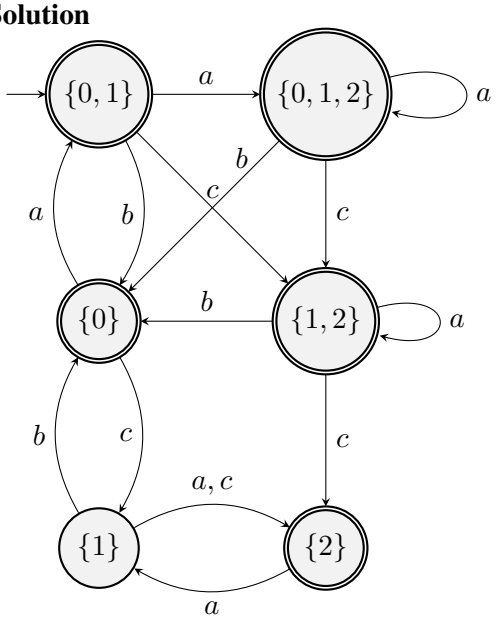
```
[a-z][0-9]([a-z]|[0-9])([a-z]|[0-9])*  |
[a-z][a-z][a-z]*[0-9]([a-z]|[0-9])*   |
[0-9][a-z]([a-z]|[0-9])([a-z]|[0-9])*   |
[0-9][0-9][0-9]*[a-z]([a-z]|[0-9])*
```

Of course this is just one solution to the question; there are many others.

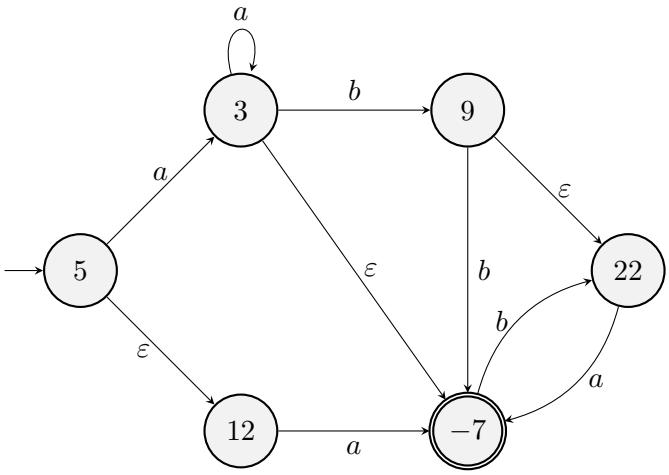
Exercise 5. *Determinize the following NFA.*



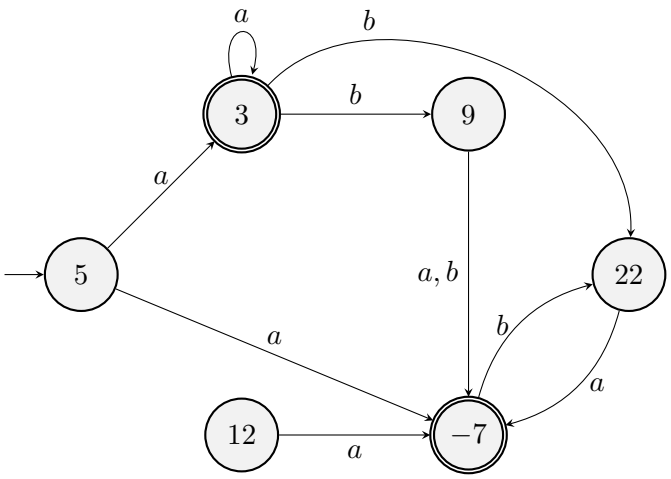
Solution



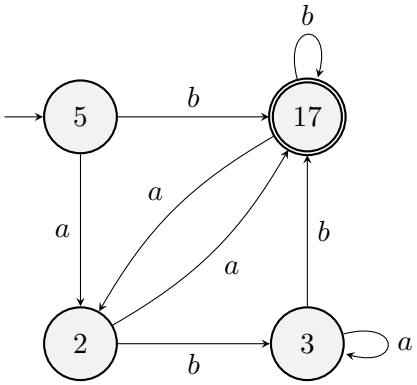
Exercise 6. *Remove ε-transitions from the following.*



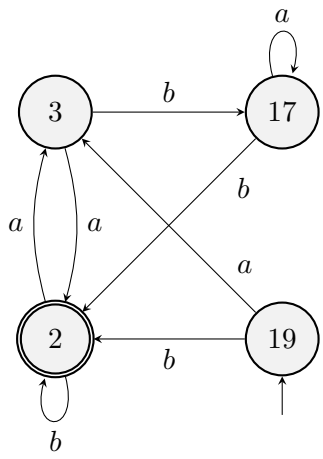
Solution



Exercise 7. *Let Automaton A be the following DFA:*



Let Automaton B be the following DFA:

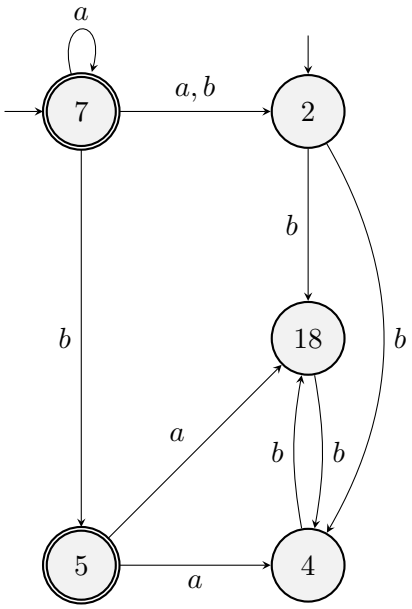


Give an isomorphism between these automata.

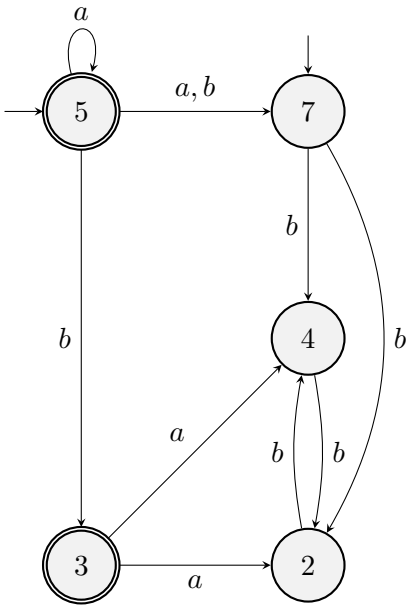
Solution

State of Automaton A	State of Automaton B
5	19
17	2
2	3
3	17

Exercise 8. *Let Automaton A be the following NFA.*



Let Automaton B be the following NFA.



Give two isomorphisms between these automata.

Solution

Here's an isomorphism:

State of Automaton A	State of Automaton B
7	5
2	7
5	3
18	4
4	2

Here's another isomorphism:

State of Automaton A	State of Automaton B
7	5
2	7
5	3
18	2
4	4