Recitation 4: Regular Expressions and DFAs

COP3402 FALL 2015 – ARYA POURTABATABAIE

FROM EURIPIDES MONTAGNE, FALL 2014

Regular Expressions: Syntax

Operation	Regular Expression	Yes	No
Concatenation	aabaab	aabaab	every other string
Logical Or	aa | baab	aa baab	every other string
Replication	ab*a	aa aba abbba	ε ab ababa
Grouping	a(a|b)aab	aaaab abaab	every other string
	(ab)*a	a aba ababa	ε aa abbba

Regular Expressions: Examples

Regular Expression	Yes	No
`a* | (a*ba*ba*ba*)` multiple of three b's	ε bbb aaa abbbaaa bbbaababbaa	b bb abbaaaa baabbbaa
`a | a(a|b)*a` begins and ends with a	a aba aa abbaabba	ε ab ba
`(a|b)* abba (a|b)*` contains the substring abba	abba bbabbabb abbaabba	ε abb bbaaba

Regular Expressions: Exercise 1

Let $\sum$ be a finite set of symbols

$\sum = \{10, 11\}$

$\sum^* = ?$

Regular Expressions: Exercise 1

Let ∑ be a finite set of symbols

∑ = {10, 11}

∑* = {ϵ, 10, 11, 1010, 1011, 1110, 1111, …}

Regular Expressions: Exercise 2

Let $\Sigma=\{0,1\}$ be a finite set of symbols

Let L1 and L2 be sets of strings from Σ^*

L1L2 is the set $\{xy \mid x \in L1 \text{ and } y \in L2\}$

L1 = {10, 1}, L2 = {011, 11}

L1L2 = ?

Regular Expressions: Exercise 2

Let $\Sigma=\{0,1\}$ be a finite set of symbols

Let L1 and L2 be sets of strings from Σ^*

L1L2 is the set $\{xy \mid x \in L1 \text{ and } y \in L2\}$

L1 = {10, 1}, L2 = {011, 11}

L1L2 = {10011, 1011, 111}

Regular Expressions: Exercise 3

Write regular expressions for:

All strings of 0’s and 1’s

All strings of 0’s and 1’s with at least 2 consecutive 0’s

All strings of 0’s and 1’s beginning with 1 and *not* having two consecutive 0’s

Regular Expressions: Exercise 3

Write regular expressions for:

All strings of 0’s and 1’s

(0|1)*

All strings of 0’s and 1’s with at least 2 consecutive 0’s

All strings of 0’s and 1’s beginning with 1 and *not* having two consecutive 0’s

Regular Expressions: Exercise 3

Write regular expressions for:

All strings of 0's and 1's

(0|1)*

All strings of 0's and 1's with at least 2 consecutive 0's

(0|1)*00(0|1)*

All strings of 0's and 1's beginning with 1 and *not* having two consecutive 0's

Regular Expressions: Exercise 3

Write regular expressions for:

All strings of 0’s and 1’s

    (0|1)*

All strings of 0’s and 1’s with at least 2 consecutive 0’s

    (0|1)*00(0|1)*

All strings of 0’s and 1’s beginning with 1 and *not* having two consecutive 0’s

    (1|10)+

Regular Expressions: Exercise 4

Characterize the strings matched by these patterns:

(0|1)*011

0*1*2*

00*11*22*

Regular Expressions: Exercise 4

Characterize the strings matched by these patterns:

(0|1)*011

All strings of 0's and 1's ending in 011

0*1*2*

00*11*22*

Regular Expressions: Exercise 4

Characterize the strings matched by these patterns:

(0|1)*011

All strings of 0's and 1's ending in 011

0*1*2*

Any number of 0's followed by any number of 1's followed by any number of 2's

00*11*22*

Regular Expressions: Exercise 4

Characterize the strings matched by these patterns:

(0|1)*011

All strings of 0's and 1's ending in 011

0*1*2*

Any number of 0's followed by any number of 1's followed by any number of 2's

00*11*22*

The same, but with at least one of each digit

Regular Expressions in Practice

Regular expressions are a fairly standard tool built in to most modern programming languages.

- Java
- Perl
- Python
- C#

They're also heavily used in the UNIX environment in general.

- Variations are even available in some Microsoft programs – ever used Word wildcard search?

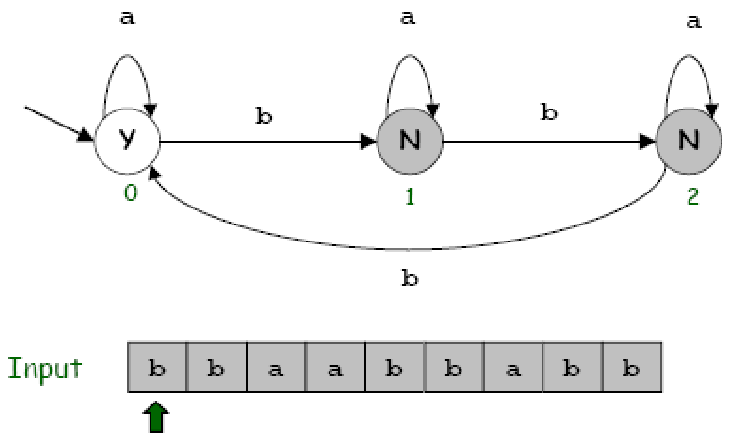
But they're not in C.

So how do you start thinking this way about string parsing?

Deterministic Finite Automata

Simple *machines* with a finite number of *states*.

- Begin in the *start* state
- Read the first input *symbol*
- *Move* to a new state, depending on the current state and input symbol
- Repeat until the last input symbol is read
- Accept or reject the string depending on the label of the last state



DFAs and REs: Duality

A regular expression is a concise way to *describe* a set of strings.

- In other words – a language!
- Languages that can be described by regular expressions are called *regular languages.*

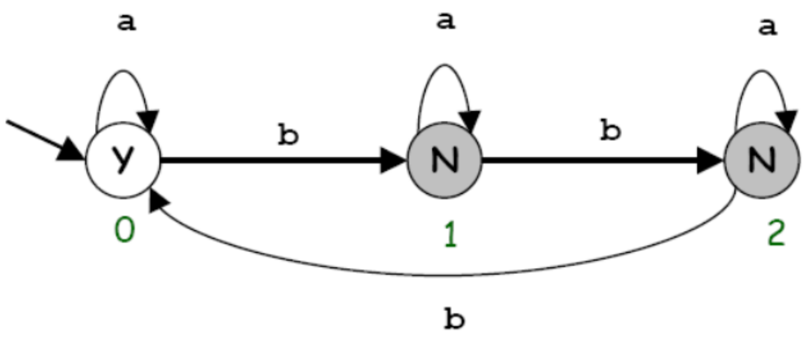
A DFA is a machine to *recognize* whether a given string is *in* a given set.

It can be demonstrated that:

- For any set of strings described by a regular expression, there is a DFA that will recognize it.
- For any set of strings recognized by a DFA, there is a regular expression that will describe it.

Example of Duality

DETERMINISTIC FINITE AUTOMATON



REGULAR EXPRESSION

(a*ba*ba*ba*)* a*

Duality in Practice

To match against a regular expression pattern:

- Build a DFA corresponding to the regular expression
  - (DFAs are *really* easy to build)
- Run the DFA on the input string

You will use similar techniques in your parser, and probably even your scanner.

- *Don't overthink this right now*

Limitations

There are such things as *non-regular* languages – languages that we cannot describe with a regular language, and (therefore) cannot recognize with a DFA.

Some examples:

- The set of all bit strings with equal number of 0s and 1s
- The set of all decimal strings that represent prime numbers
- Many, many more

DFA Conversion: Exercise 1

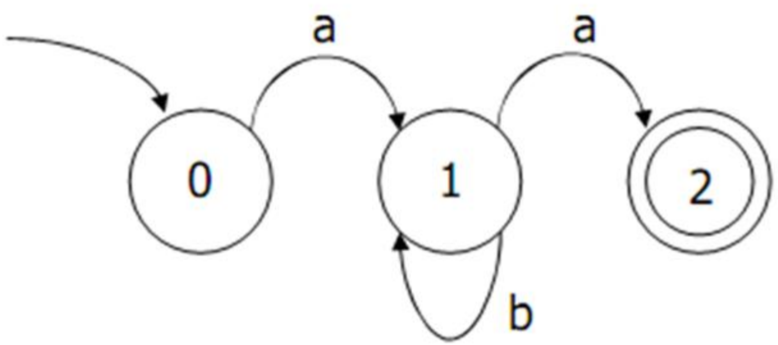
Create a DFA that accepts the strings in the language denoted by regular expression:

ab*a

DFA Conversion: Exercise 1

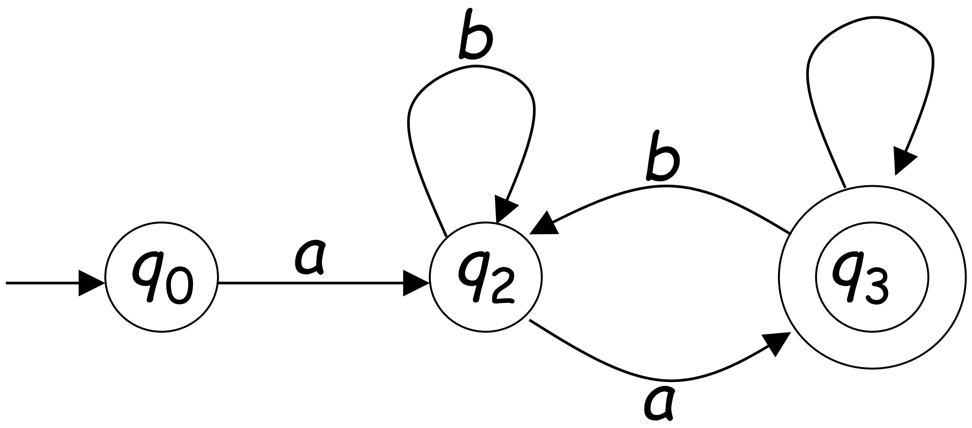
Create a DFA that accepts the strings in the language denoted by regular expression:

ab*a



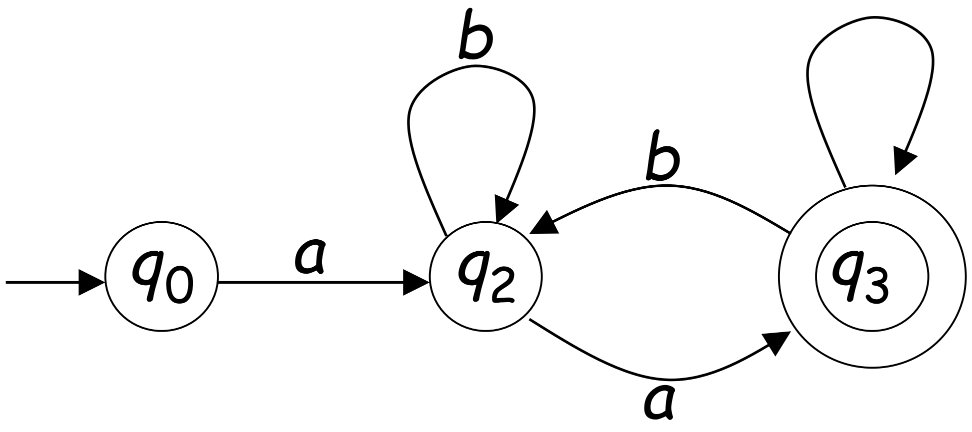
DFA Conversion: Exercise 2

Write the RE for this automata:



DFA Conversion: Exercise 2

Write the RE for this automata:



a(a|b)*aa*

DFA to RE by State Elimination

There is a systematic way to convert DFAs to REs.

- (And you guessed it, it involves divide-and-conquer.)

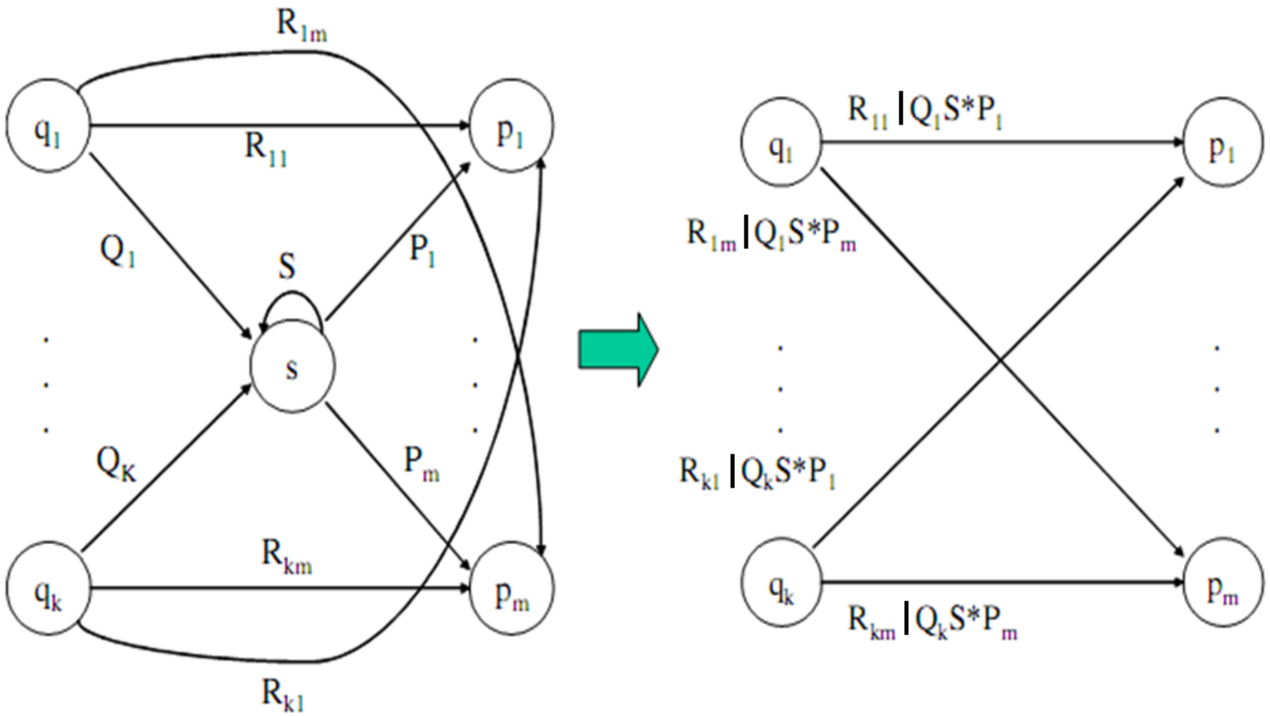
It's called *state elimination*.

- Eliminate states of the automaton, and replace their *edges* with regular expressions that include the behavior of the eliminated states.
- Eventually we get down to two nodes, and we're basically done.

State Elimination Example

Consider the figure below, which shows the elimination of State s.

- The labels on all the edges are regular expressions.
- To remove s, we must make labels from each q_i to p_1 up to p_m that describe the paths we could have taken through s.



The State Elimination Process

Starting with *intermediate* states then continuing to *accepting* states, we apply the state elimination process to produce equivalent automata with regular expression labels on the edges.

The end result will be a one- or two- state automaton with a start state and accepting state.

State Elimination Results

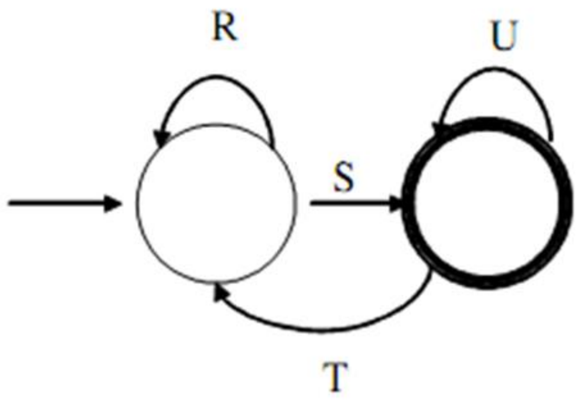
If the start state is *not* an accepting state, we will end up with an automaton that looks like this, for some regular expressions R, S, T and U.

We can always describe this automaton as:

(R | SU*T)*SU*

…copying in the subexpressions as necessary.

- We may not always need R, T or U.
- We will always need S for any expression worth converting.

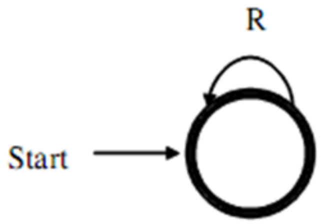


State Elimination Results 2

If the start state *is* an accepting state, we will have an automaton that looks like this.

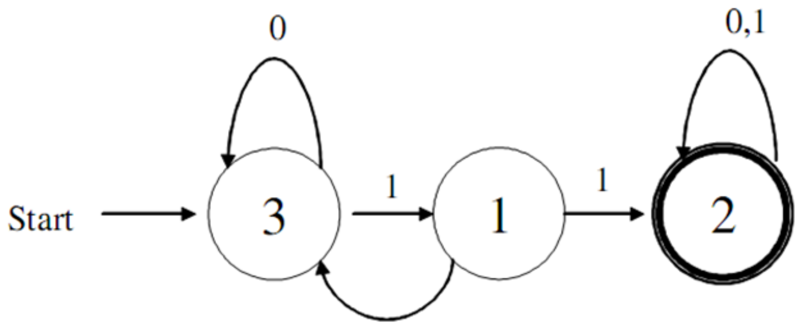
This simply becomes:

R*



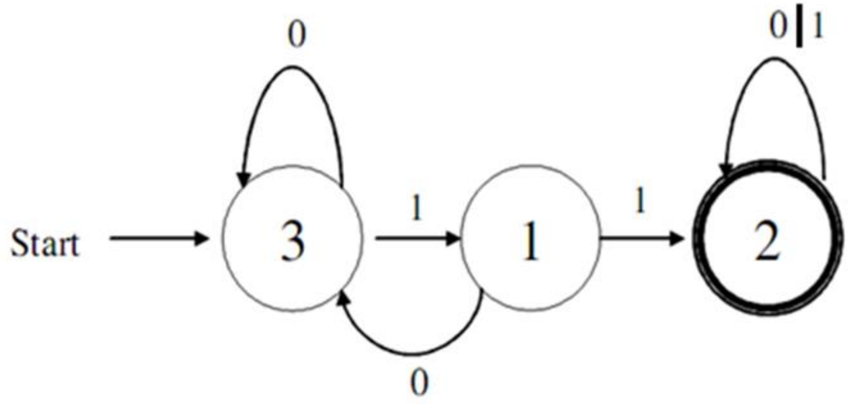
State Elimination Exercise 1

Convert this DFA to a regular expression.



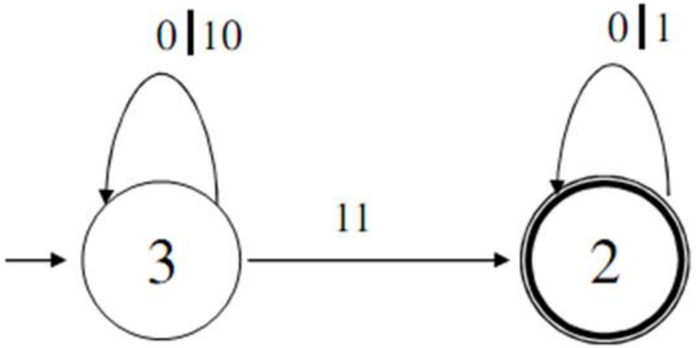
State Elimination Exercise 1

First, express the edges as regular expressions.



State Elimination Exercise 1

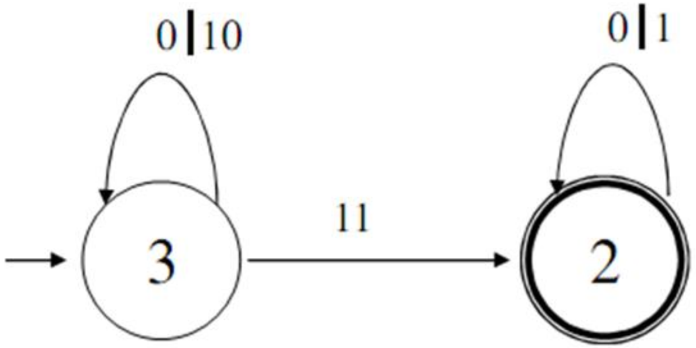
Now, eliminate State 1.



State Elimination Exercise 1

Now, eliminate State 1.

Answer: (0|10)*11(0|1)*



Multiple Accepting States

With more than one accepting state, it gets trickier.

We have to repeat the process (or at least some of the process) for *each accepting state*.

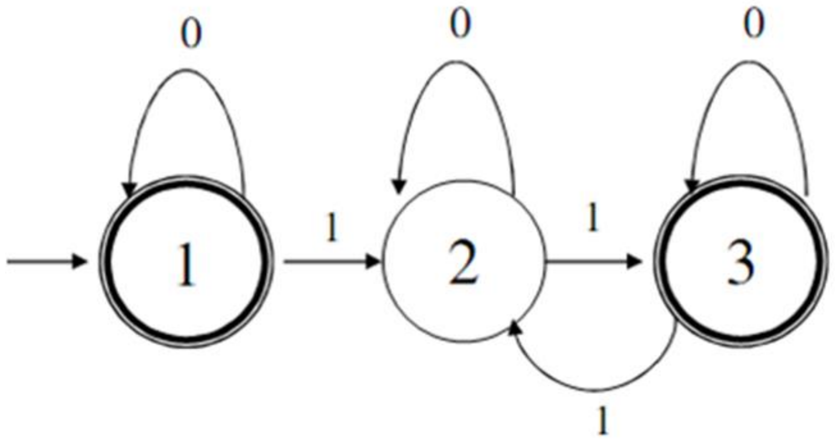
So if we have *n* accepting states, then for each accepting state s_i, ($i \in [1, \ldots n]$),

- Consider a new DFA so that s_i is the only accepting state – in other words, set all other states to be non-accepting.
- Perform the state elimination process to get regular expression R_i.

Once we're done, our desired regular expression is the union of $R_1 \ldots R_n$.

State Elimination Exercise 2

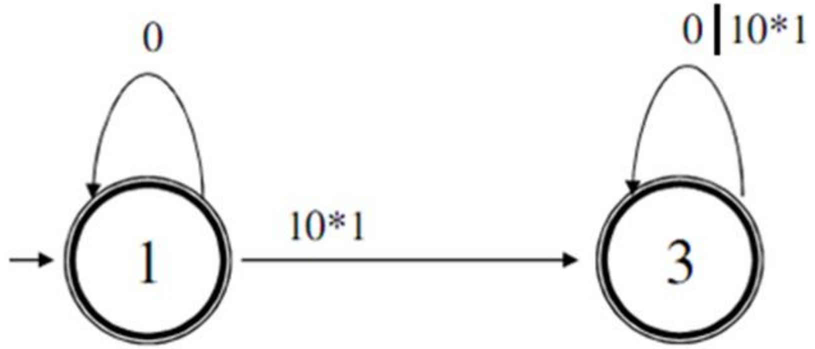
Here's an automata that recognizes any string with an even number of 1's.



State Elimination Exercise 2

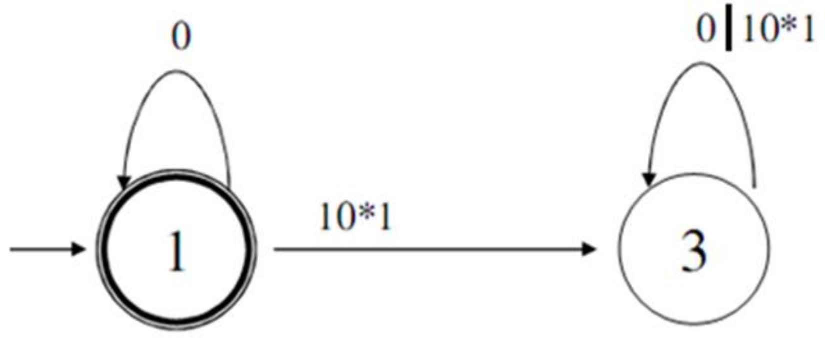
First eliminate State 2. That's the easy part.

But now we have two accepting states to deal with.



State Elimination Exercise 2

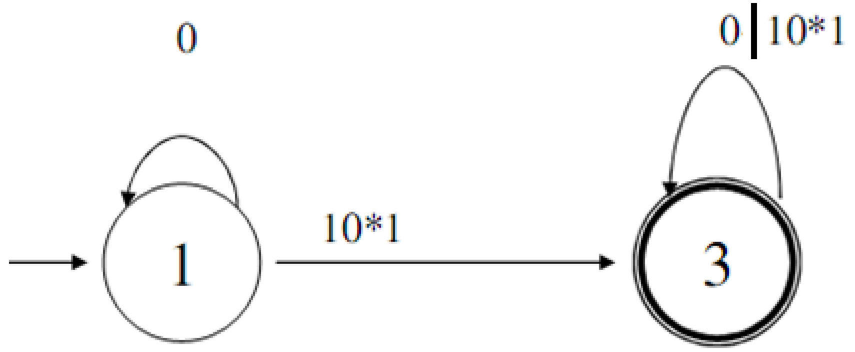
We'll look at the start state first – and it turns out to be easy. This is just 0* - if we ever get a 1, we enter a state we can't ever accept from.



State Elimination Exercise 2

And now the other one – which is less trivial, but is nonetheless already in our preferred final form:

0*10*1(0|10*1)*



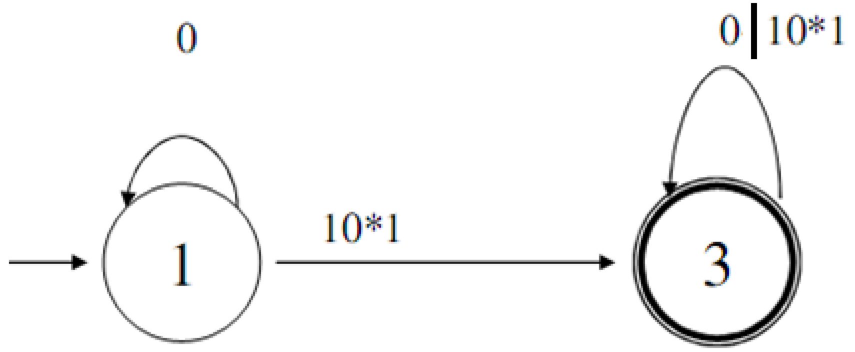
State Elimination Exercise 2

And now the other one – which is less trivial, but is nonetheless already in our preferred final form:

0*10*1(0|10*1)*

We combine the two expressions:

0*|0*10*1(0|10*1)*



Regular Expressions to Automata

…are actually much harder, and by far easiest done by first converting to a *nondeterministic finite automata*.

And that's another story for another course.

Questions?