Regular Expressions and DFAs

COP 3402

Regular Expression

- Notation to specify a set of strings

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

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

Exercise 1

- Let $\sum$ be a finite set of symbols
- $\sum = \{10, 11\}$, $\sum^* = ?$

Answer

- Answer: $\sum^* = \{\epsilon, 10, 11, 1010, 1011, 1110, 1111, \ldots\}$

Exercises 2

- Let $\sum=\{0,1\}$ be a finite set of symbols and let L1 and L2 be sets of strings from $\sum^*$. L1L2 is the set {xy | x is in L1, and y is in L2}
- L1 = {10, 1}, L2 = {011, 11}, L1L2 = ?

Answer

- L1L2 = {10011, 1011, 111}

Exercises 3

- Write RE for
  1. All strings of 0's and 1's
  2. All strings of 0's and 1's with at least 2 consecutive 0's
  3. All strings of 0's and 1's beginning with 1 and not having two consecutive 0's

Answer

1. (0|1)*

All strings of 0's and 1's

2. (0|1)*00(0|1)*

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

3. (1|10)+

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

More Exercises

- 1) (0|1)*011
- 2) 0*1*2*
- 3) 00*11*22*

More Exercises (Answers)

1) (0|1)*011

Answer: all strings of 0's and 1's ending in 011

2) 0*1*2*

- Answer: any number of 0's followed by any number of 1's followed by any number of 2's
- 3) 00*11*22*

Answer: strings in 0*1*2 with at least one of each symbol

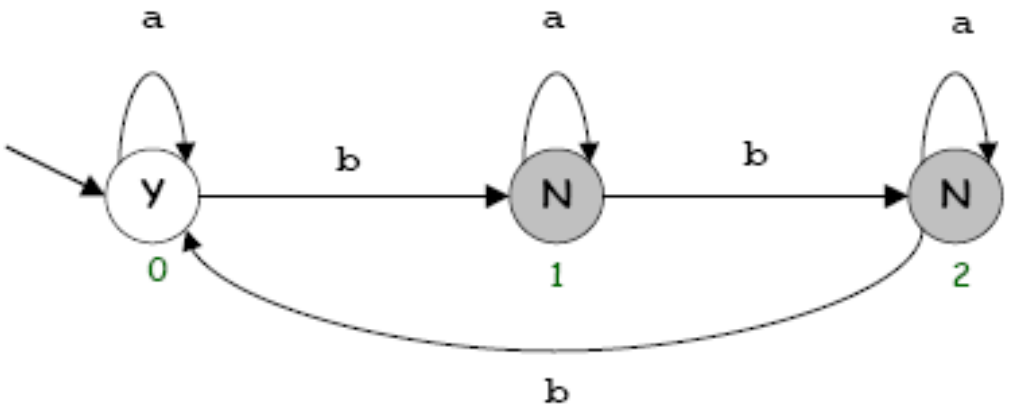
Using Regular Expressions

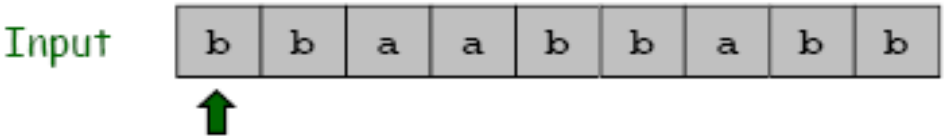
- Regular expressions are a standard programmer's tool.
- Built in to Java, Perl, Unix, Python, ...

Deterministic Finite Automata (DFA)

- Simple machine with N states.
- Begin in start state.
- Read first input symbol.
- Move to new state, depending on current state and input symbol.
- Repeat until last input symbol read.
- Accept or reject string depending on label of last state.

DFA



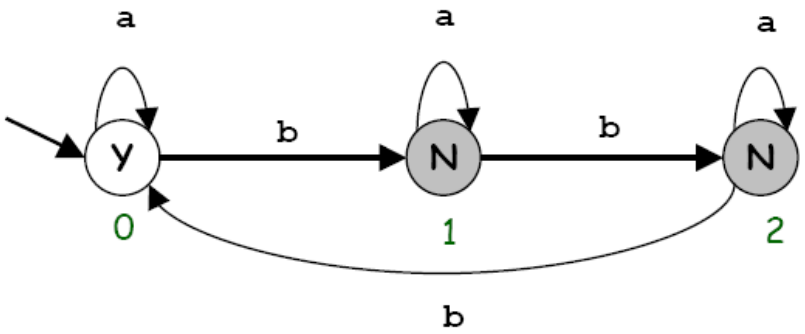


Theory of DFAs and REs

- RE. Concise way to describe a set of strings.
- DFA. Machine to recognize whether a given string is in a given set.
- **Duality**: for any DFA, there exists a regular expression to describe the same set of strings; for any regular expression, there exists a DFA that recognizes the same set.

Duality Example

- DFA for multiple of 3 b's:



- RE for multiple of 3 b's:

$$(a^*ba^*ba^*ba^*)^* \mid a^*$$

Duality…

- Practical consequence of duality proof: to match regular expression patterns, (i) build DFA and (ii) simulate DFA on input string.

Fundamental Questions

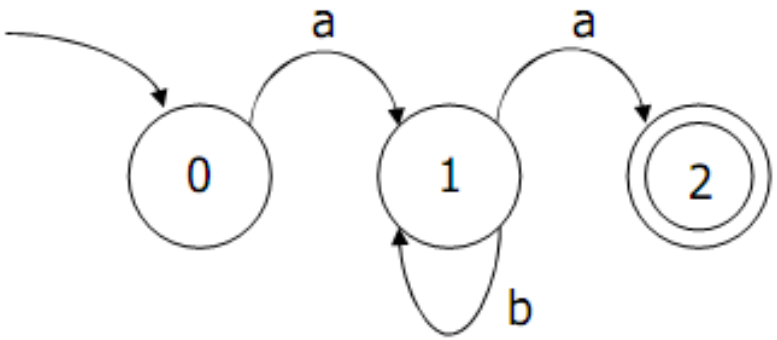
- Which languages CANNOT be described by any RE?
- Set of all bit strings with equal number of 0s and 1s.
- Set of all decimal strings that represent prime numbers.
- Many more. . . .

Problem 1

- Make a DFA that accepts the strings in the language denoted by regular expression ab*a

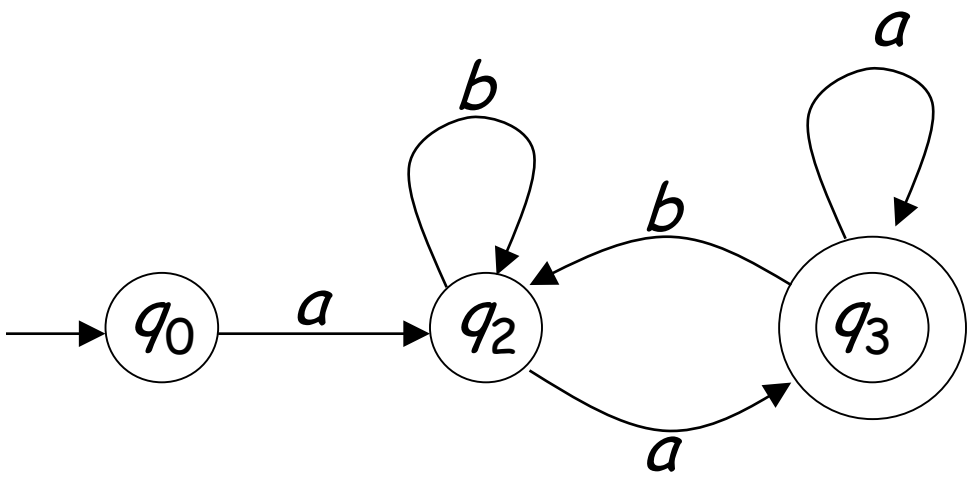
Solution

- ab*a:



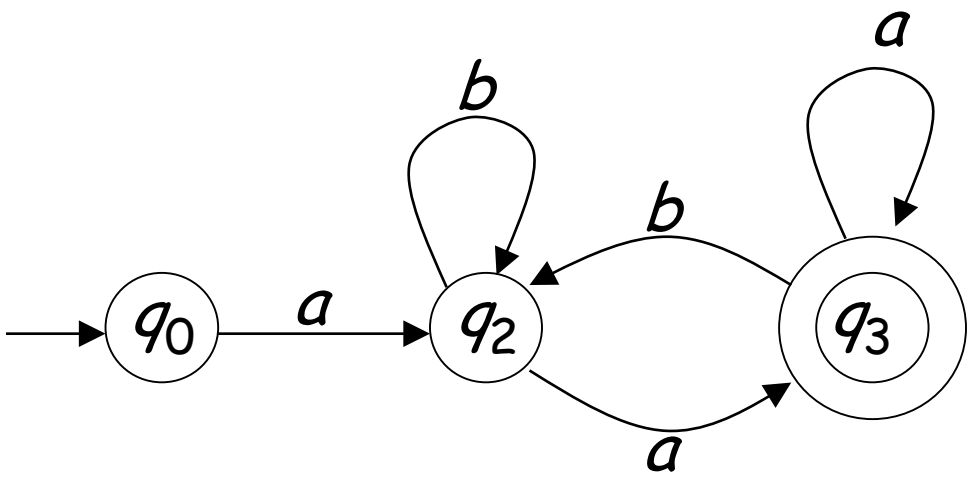
Problem 2

- Write the RE for the following automata:



Solution

- a(a|b)*aa*

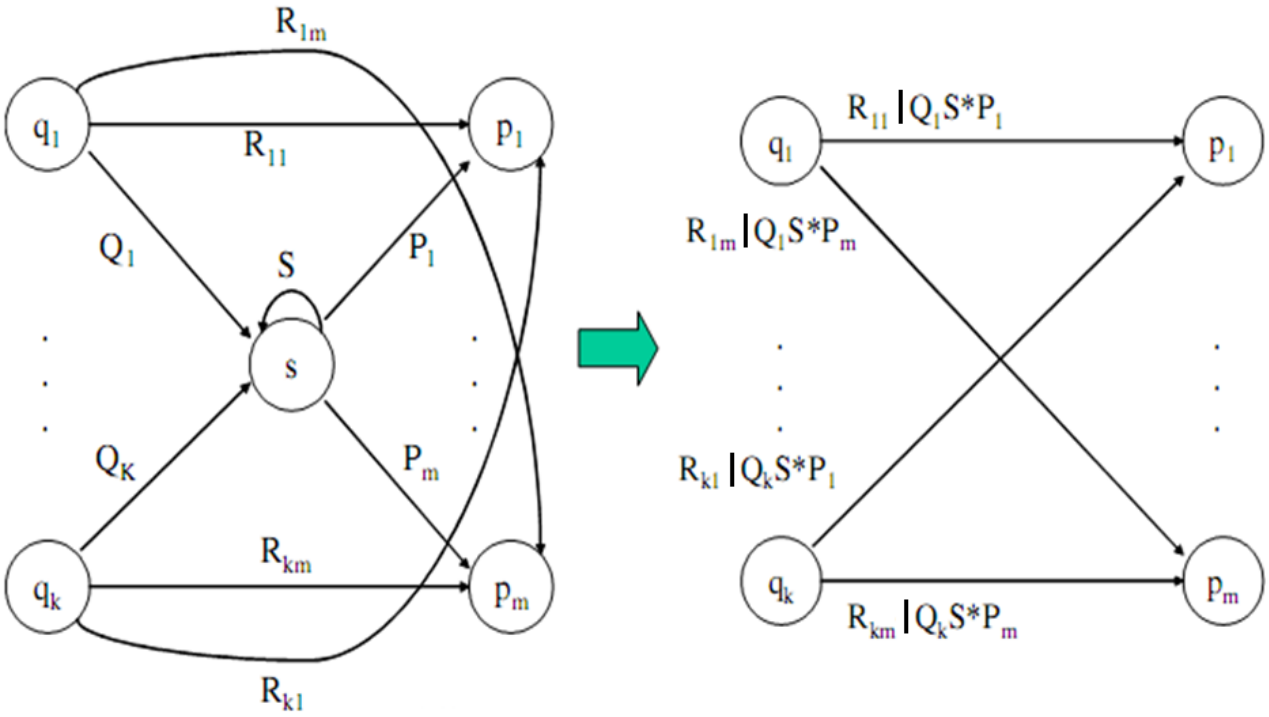


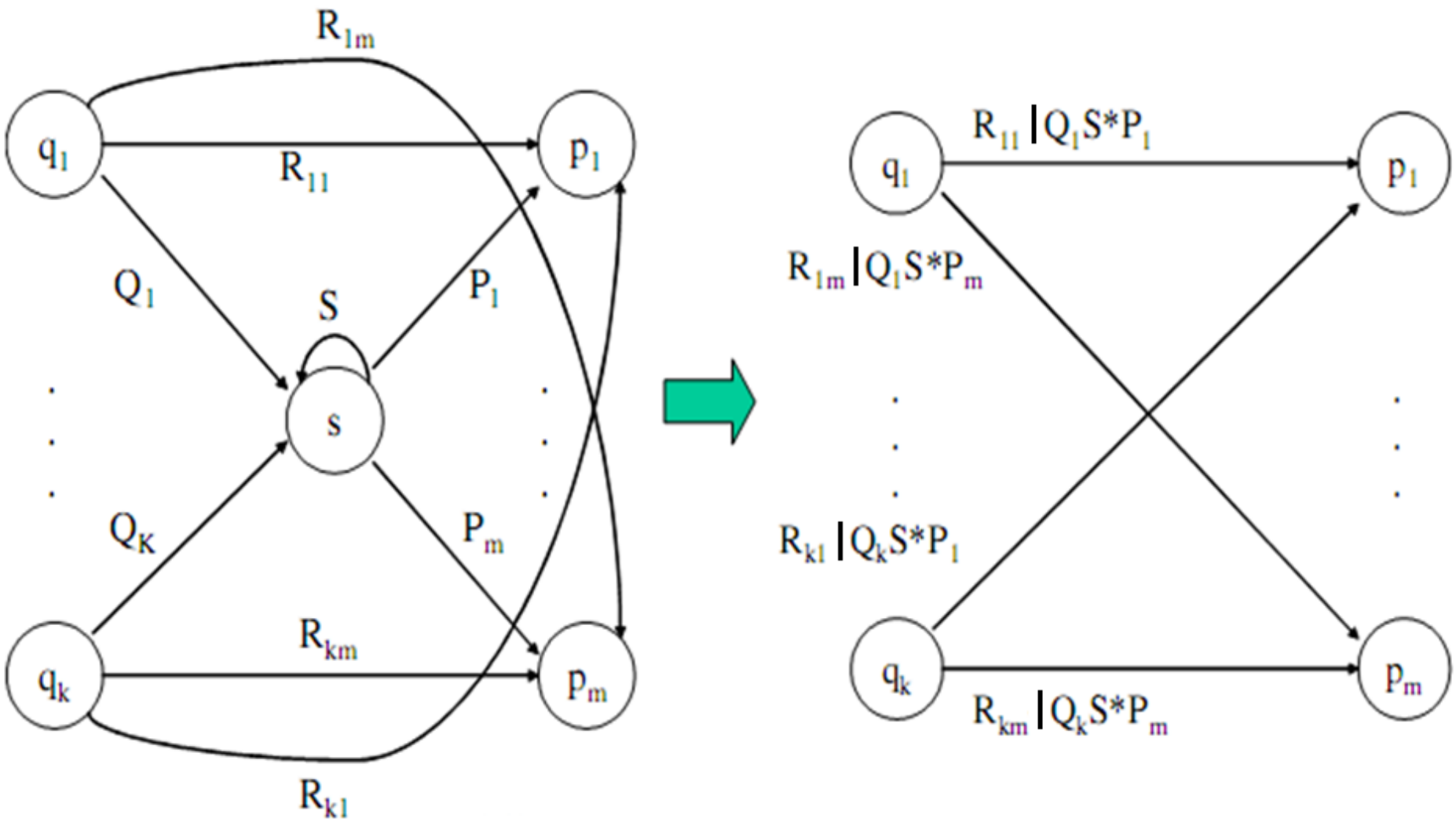
DFA to RE: State Elimination

- Eliminates states of the automaton and replaces the edges with regular expressions that includes the behavior of the eliminated states.
- Eventually we get down to the situation with just a start and final node, and this is easy to express as a RE

State Elimination

- Consider the figure below, which shows a generic state s about to be eliminated.
- The labels on all edges are regular expressions.
- To remove s, we must make labels from each qi to p1 up to pm that include the paths we could have made through s.



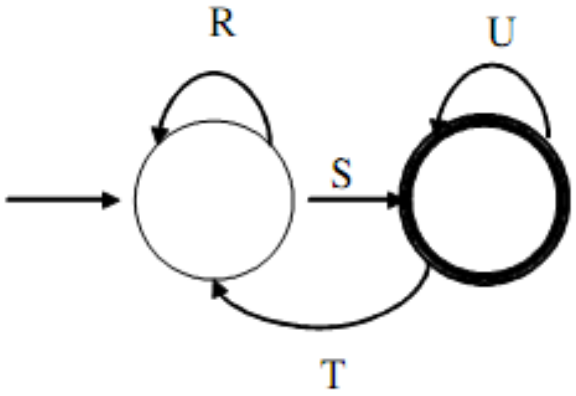

R1m
q1
p1
R11
Q1
S
P1
s
QK
Pm
qk
Rkm
pm
Rk1
R11 | Q1S*P1
R1m | Q1S*Pm
Rk1 | QkS*P1
Rkm | QkS*Pm

DFA to RE via State Elimination (1)

- Starting with intermediate states and then moving to accepting states, apply the state elimination process to produce an equivalent automaton with regular expression labels on the edges.
- The result will be a one or two state automaton with a start state and accepting state.

DFA to RE State Elimination (2)

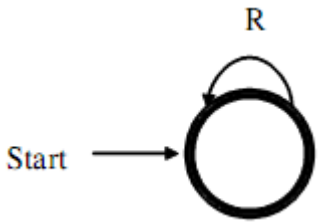
- If the start state is not an accepting state, we will have an automaton that looks like the following:



- We can describe this automaton as: (R | SU*T)*SU*

DFA to RE State Elimination (3)

- If the start state is also an accepting state, then we must also perform a state elimination from the original automaton that gets rid of every state but the start state. This leaves the following:



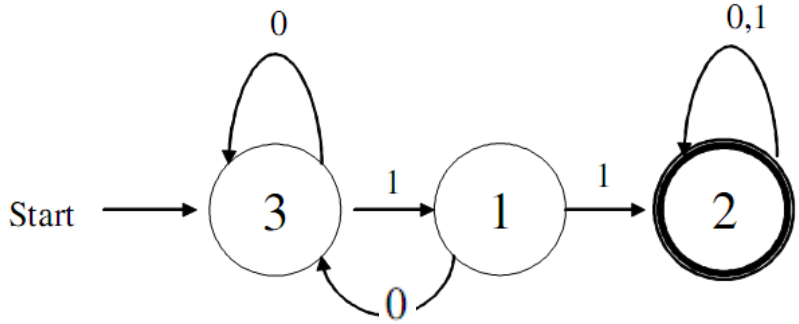
- We can describe this automaton as simply R*

DFA to RE State Elimination (4)

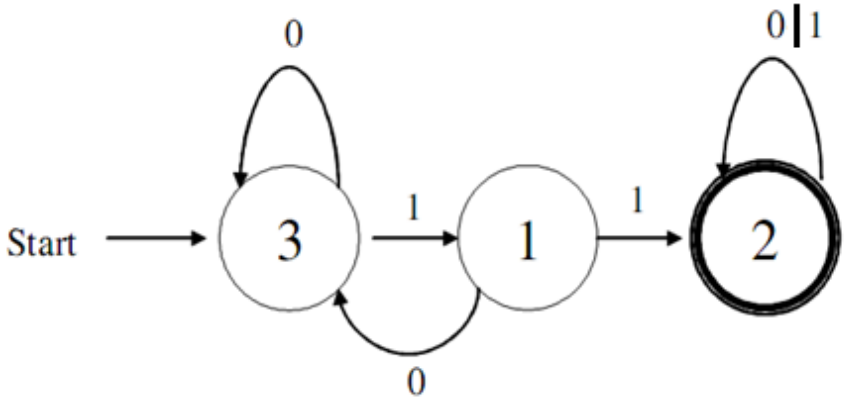
- If there are n accepting states, we must repeat the above steps for each accepting states to get n different regular expressions, R1, R2, … Rn.
- For each repeat we turn any other accepting state to non-accepting.
- The desired regular expression for the automaton is then the union of each of the n regular expressions: R1 U R2… U RN

DFA->RE Example

- Convert the following to a RE:

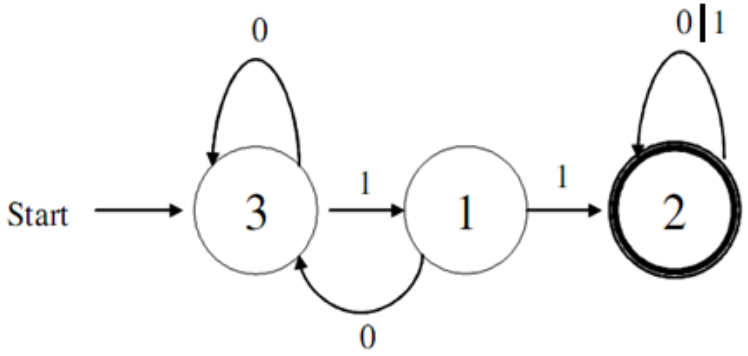


- First convert the edges to RE's:

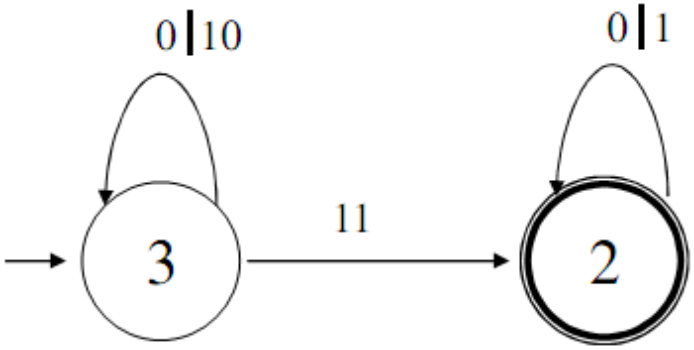


DFA -> RE Example (2)

- Eliminate State 1:



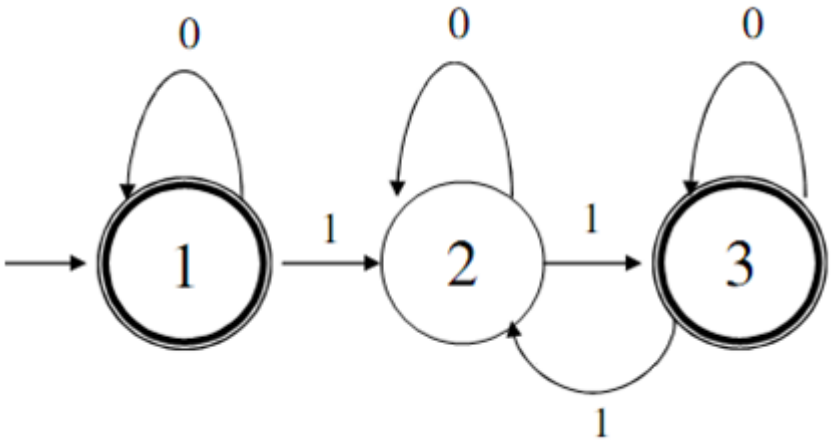
- Note edge from 3->3



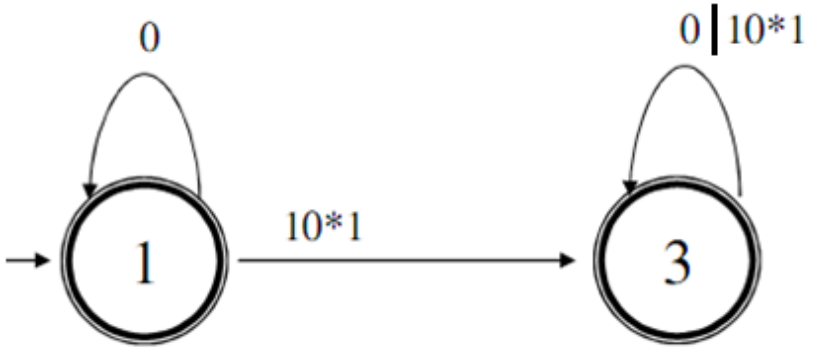
- Answer: (0|10)*11(0|1)*

Second Example

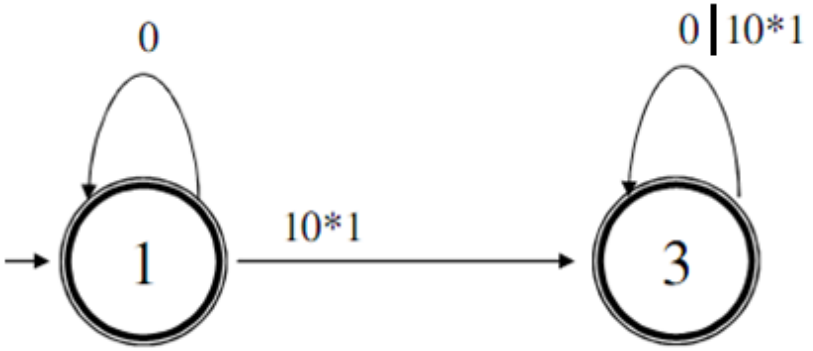
- Automata that accepts even number of 1's



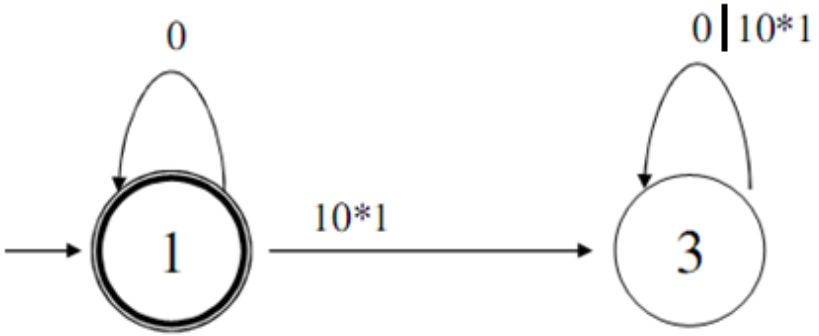
- Eliminate state 2:



Second Example (2)

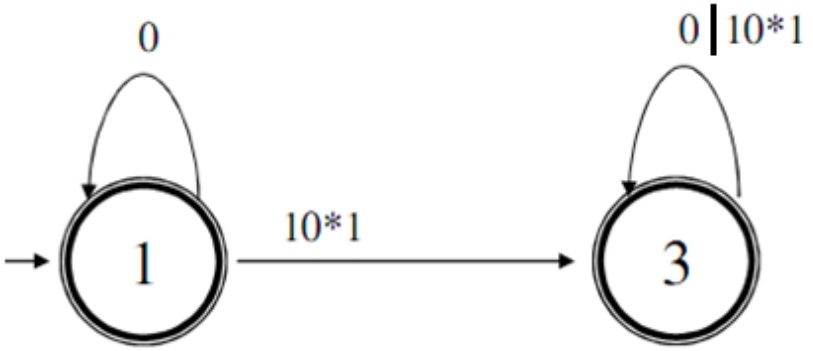


- Two accepting states, turn off state 3 first

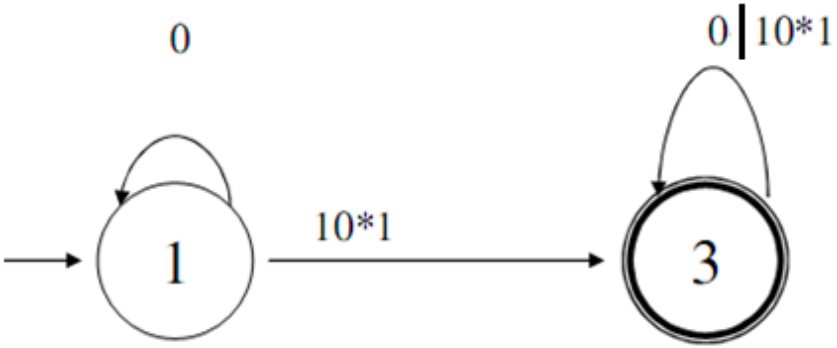


- This is just 0*; we can ignore going to state 3 since we would “die” (can’t come back to an accepting state).

Second Example (3)



- Turn off state 1 second:



- This is just 0*10*1(0|10*1)*
- Combine from previous slide to get 0* | 0*10*1(0|10*1)*

RE -> Automata

- We can do this easiest by converting a RE to an NFA (Non-deterministic Finite Automata).
- Beyond the scope of this course…

Questions