Regular Expressions

Reading: Chapter 3

Regular Expressions vs. Finite Automata

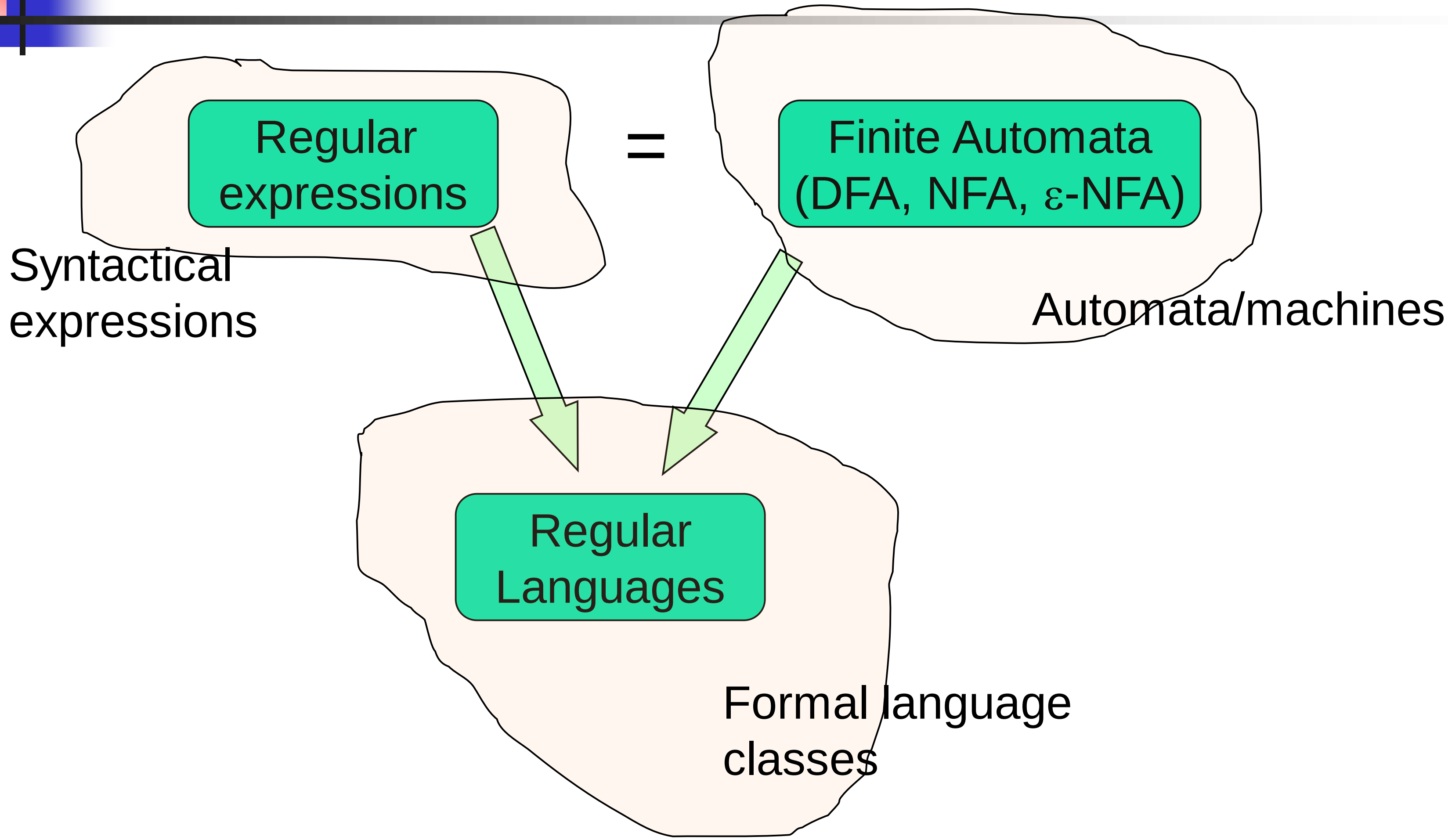
- Offers a declarative way to express the pattern of any string we want to accept
  - E.g., 01*+ 10*

- Automata => more machine-like

  < input: string , output: [accept/reject] >
- Regular expressions => more program syntax-like

- Unix environments heavily use regular expressions
  - E.g., bash shell, grep, vi & other editors, sed
- Perl scripting – good for string processing
- Lexical analyzers such as Lex or Flex

Regular Expressions



Language Operators

- Union of two languages:
  - **L U M** = all strings that are either in L or M
  - Note: A union of two languages produces a third language

- Concatenation of two languages:
  - **L . M** = all strings that are of the form *xy* s.t., $x \in L$ and $y \in M$
  - The *dot* operator is usually omitted
    - i.e., **LM** is same as L.M

“i” here refers to how many strings to concatenate from the parent language L to produce strings in the language L^i

Kleene Closure (the * operator)

- Kleene Closure of a given language L:
  - $L^0 = \{\varepsilon\}$
  - $L^1 = \{w \mid \text{for some } w \in L\}$
  - $L^2 = \{ w_1 w_2 \mid w_1 \in L, w_2 \in L \text{ (duplicates allowed)}\}$
  - $L^i = \{ w_1 w_2 \ldots w_i \mid \text{all w's chosen are } \in L \text{ (duplicates allowed)}\}$
  - (Note: the choice of each w_i is independent)
  - **L^*** $= \bigcup_{i \geq 0} L^i$ (arbitrary number of concatenations)

Example:

- Let L = { 1, 00}
  - $L^0 = \{\varepsilon\}$
  - $L^1 = \{1, 00\}$
  - $L^2 = \{11, 100, 001, 0000\}$
  - $L^3 = \{111, 1100, 1001, 10000, 000000, 00001, 00100, 0011\}$
  - **L^*** $= L^0 \cup L^1 \cup L^2 \cup \ldots$

Kleene Closure (special notes)

- L^* is an infinite set iff $|L| \geq 1$ and $L \neq \{\varepsilon\}$ **Why?**
- If $L=\{\varepsilon\}$, then $L^* = \{\varepsilon\}$ **Why?**
- If $L = \Phi$, then $L^* = \{\varepsilon\}$ **Why?**

Σ^* denotes the set of all words over an alphabet Σ

- Therefore, an abbreviated way of saying there is an arbitrary language L over an alphabet Σ is:
  - $L \subseteq \Sigma^*$

Building Regular Expressions

- Let E be a regular expression and the language represented by E is L(E)
- Then:
  - (E) = E
  - L(E + F) = L(E) U L(F)
  - L(E F) = L(E) L(F)
  - L(E*) = (L(E))*

Example 1: how to use these regular expression properties and language operators?

- ***L = { w | w is a binary string which does not contain two consecutive 0s or two consecutive 1s anywhere)***
  - E.g., w = 01010101 is in L, while w = 10010 is not in L
- Goal: Build a regular expression for L
- Four cases for w:
  - Case A: w starts with 0 and |w| is even
  - Case B: w starts with 1 and |w| is even
  - Case C: w starts with 0 and |w| is odd
  - Case D: w starts with 1 and |w| is odd
- Regular expression for the four cases:
  - Case A: (01)*
  - Case B: (10)*
  - Case C: 0(10)*
  - Case D: 1(01)*
- Since L is the union of all 4 cases:
  - Reg Exp for L = (01)* + (10)* + 0(10)* + 1(01)*
- If we introduce ε then the regular expression can be simplified to:
  - Reg Exp for L = (ε +1)(01)*(ε +0)

Some more examples:

- ***L2 = { w | w is a binary string which ends with 1 does not contain 00}.***
  - E.g., w = 01010101 is in L, while w = 10010 is not in L
- <u>Goal: </u>Build a regular expression for L2
  - Reg Exp for L = (1+01)*(1+01)

- ***L3 = { w | w is a binary string which ends with 01}.***
- <u>Goal: </u>Build a regular expression for L3
  - Reg Exp for L = (1+0)*01

Precedence of Operators

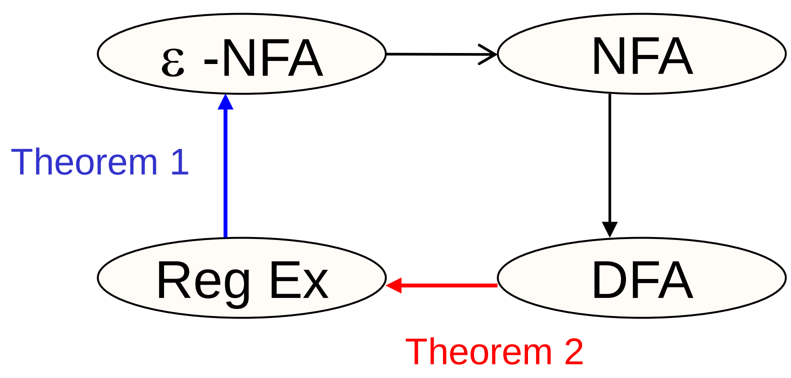
- Highest to lowest
  - * operator (star)
  - .   (concatenation)
  - + operator

- Example:
  - $01^* + 1 \quad = \quad (\,0\,.\,((1)^*)\,) + 1$

Finite Automata (FA) & Regular Expressions (Reg Ex)

- To show that they are interchangeable, consider the following theorems:
  - *Theorem 1: For every DFA A there exists a regular expression R such that L(R)=L(A)*
  - *Theorem 2: For every regular expression R there exists an ε -NFA E such that L(E)=L(R)*

Proofs in the book

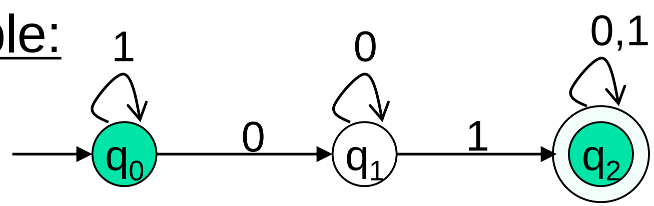


Kleene Theorem

DFA to RE construction

Informally, trace all distinct paths (traversing cycles only once) from the start state to *each of the* final states and enumerate all the expressions along the way

<u>Example:</u>



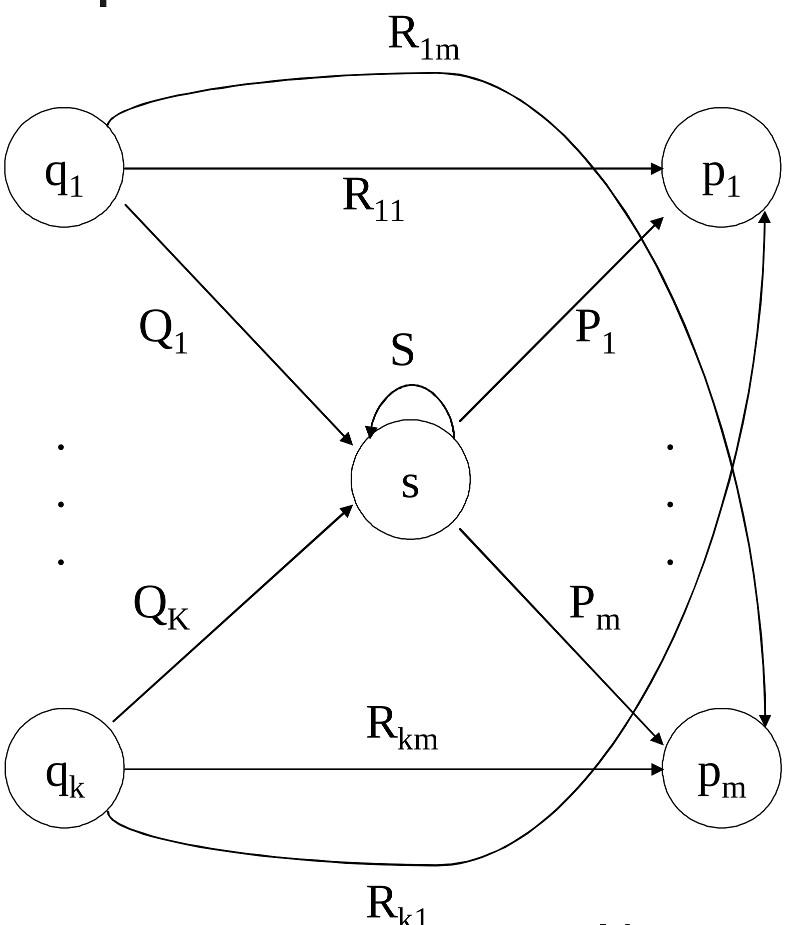
$$\underbrace{(1^*)}_{1^*}\ \underbrace{0\ \ (0^*)}_{00^*}\ \underbrace{1}_{1}\ \underbrace{(0+1)^*}_{(0+1)^*}$$

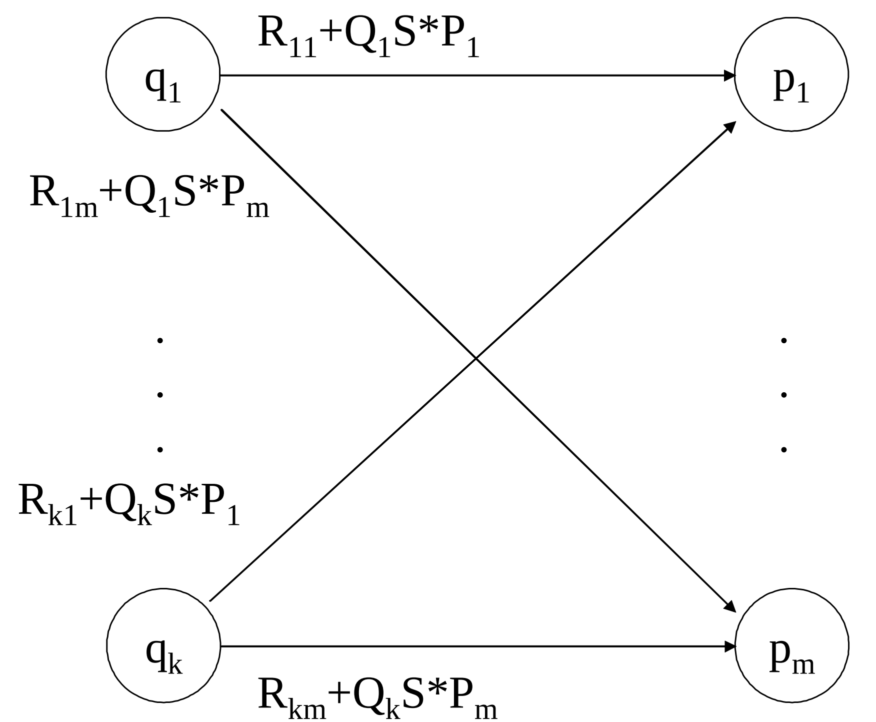
1*00*1(0+1)*

Q) What is the language?

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 q_i to p_1 up to p_m that include the paths we could have made through s.





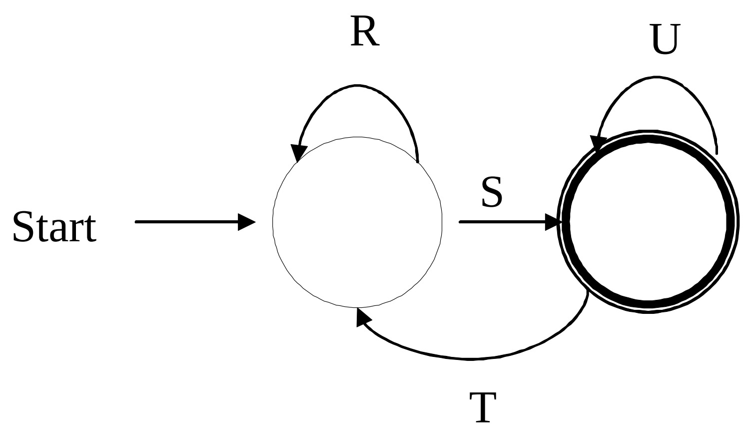
Note: q and p may be the same state!

DFA to RE via State Elimination (1)

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)

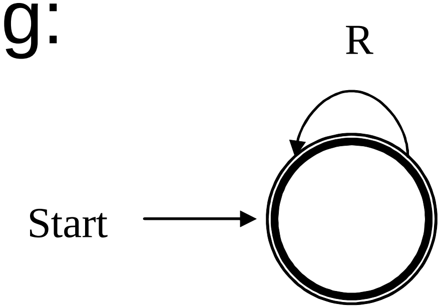
2. If the two states are different, we will have an automaton that looks like the following:



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

DFA to RE State Elimination (3)

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:



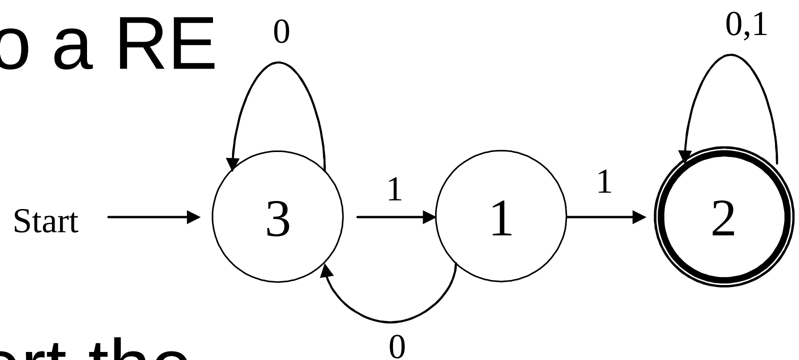
We can describe this automaton as simply R*.

DFA to RE State Elimination (4)

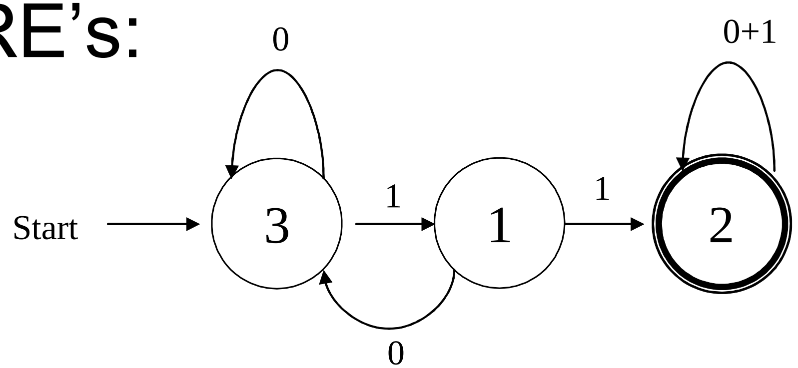
4. If there are n accepting states, we must repeat the above steps for each accepting states to get n different regular expressions, R_1, R_2, … R_n. For each repeat we turn any other accepting state to non-accepting. The desired regular expression for the automaton is then the union of each of the n regular expressions: $R_1 \cup R_2 \ldots \cup R_N$

DFA→RE Example

- Convert the following to a RE

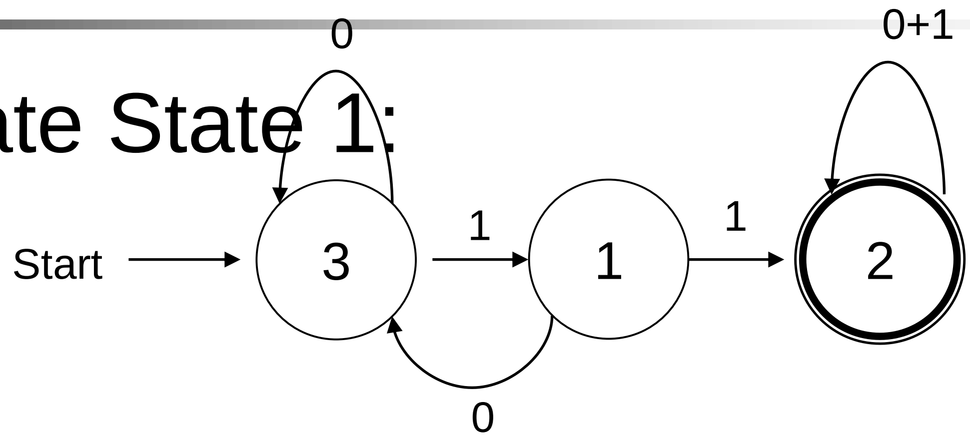


- First convert the edges to RE's:



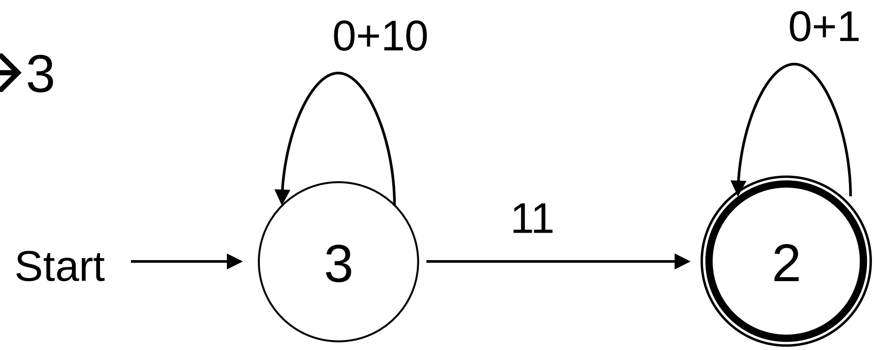
DFA → RE Example (2)

- Eliminate State 1:



- To:

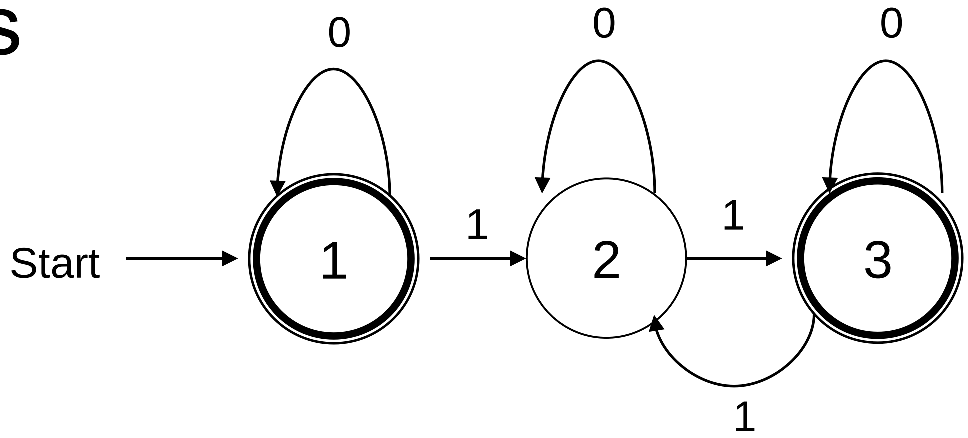
Note edge from 3→3



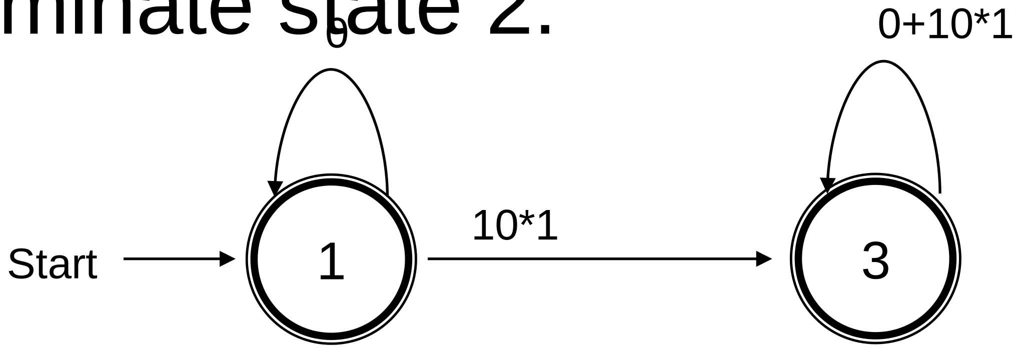
Answer: (0+10)*11(0+1)*

Second Example

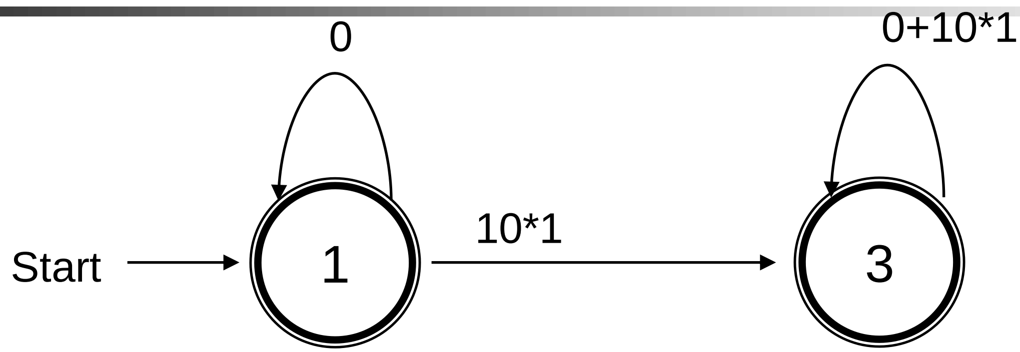
- Automata that accepts even number of 1's



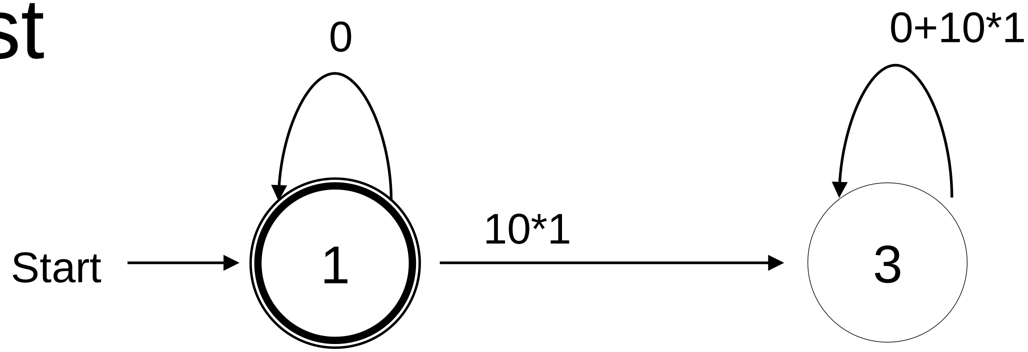
- Eliminate state 2:



Second Example (2)

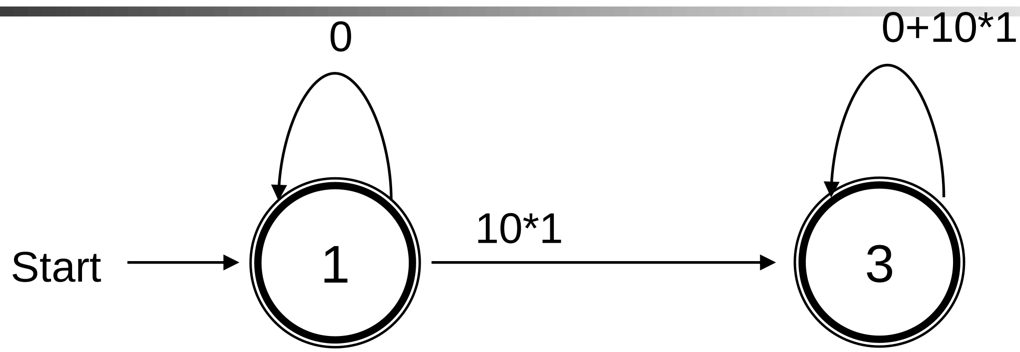


- Two accepting states, turn off state 3 first

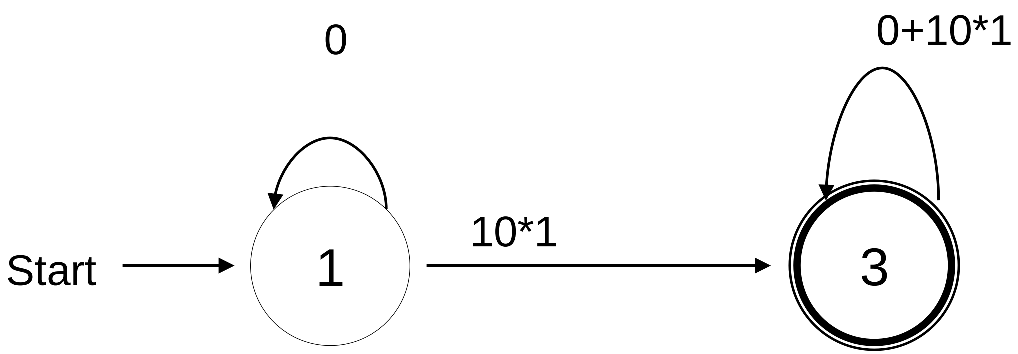


This is just 0*; can ignore going to state 3 since we would “die”

Second Example (3)



- Turn off state 1 second:



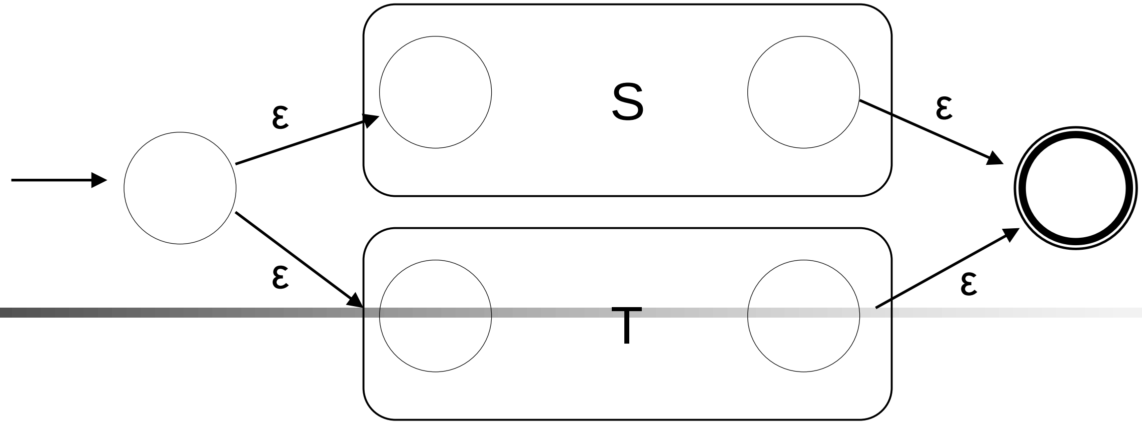
This is just 0*10*1(0+10*1)*

Combine from previous slide to get 0* + 0*10*1(0+10*1)*

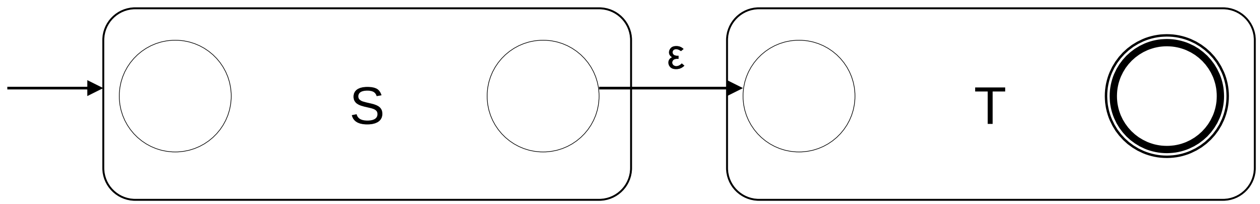
RE to ε-NFA construction

- Suppose ε-NFA$_1$ and ε-NFA$_2$ are the automata for R_1 and R_2
- Three operations to worry about: union $R_1 + R_2$, concatenation R_1R2), closure R_1^*
- With ε-transitions, construction is straightforward
  - Union: create a new start state, with ε-transitions into the start states of ε-NFA$_1$ and ε-NFA$_2$; create a new final state, with ε-transitions from the two final states of ε-NFA$_1$ and ε-NFA$_2$
  - Concatenation: ε-transition from final state of ε-NFA$_1$ to the start state of ε-NFA$_2$
  - Closure: closure can be supported by an ε-transition from final to start state; need a few more ε-transitions (why?)

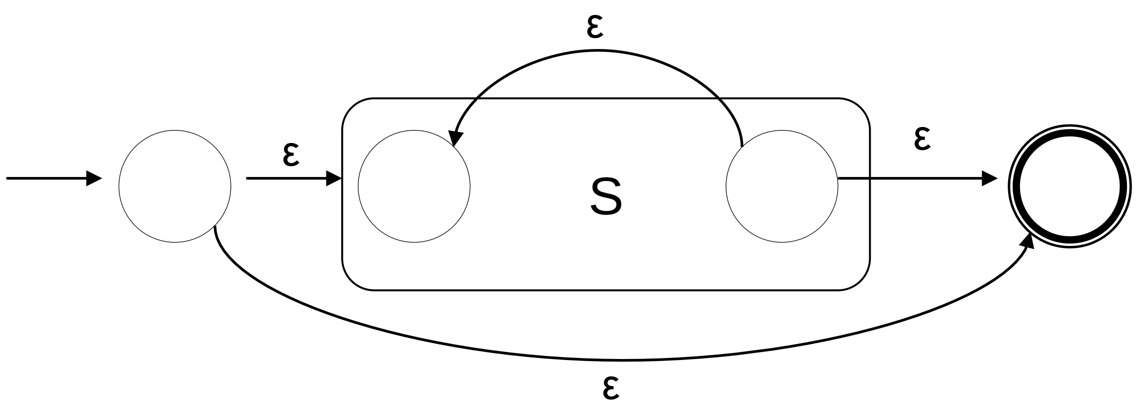
R=S+T



R=ST



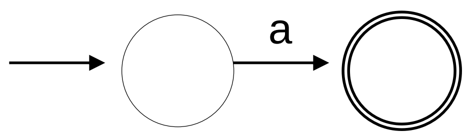
$R=S^*$



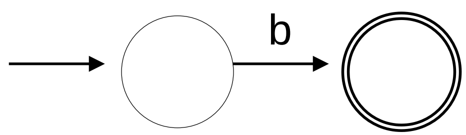
RE to ε-NFA Example

- Convert R= (ab+a)* to an NFA
  - We proceed in stages, starting from simple elements and working our way up

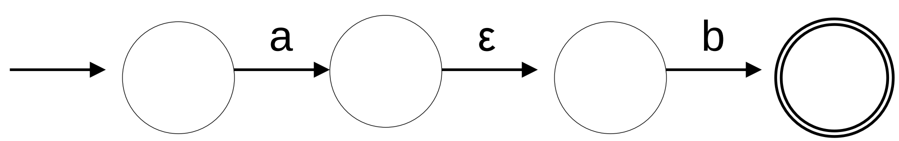
a



b

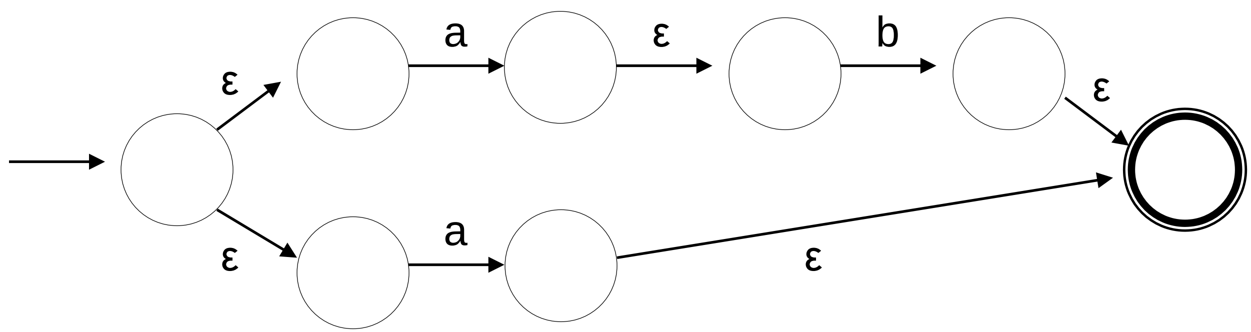


ab

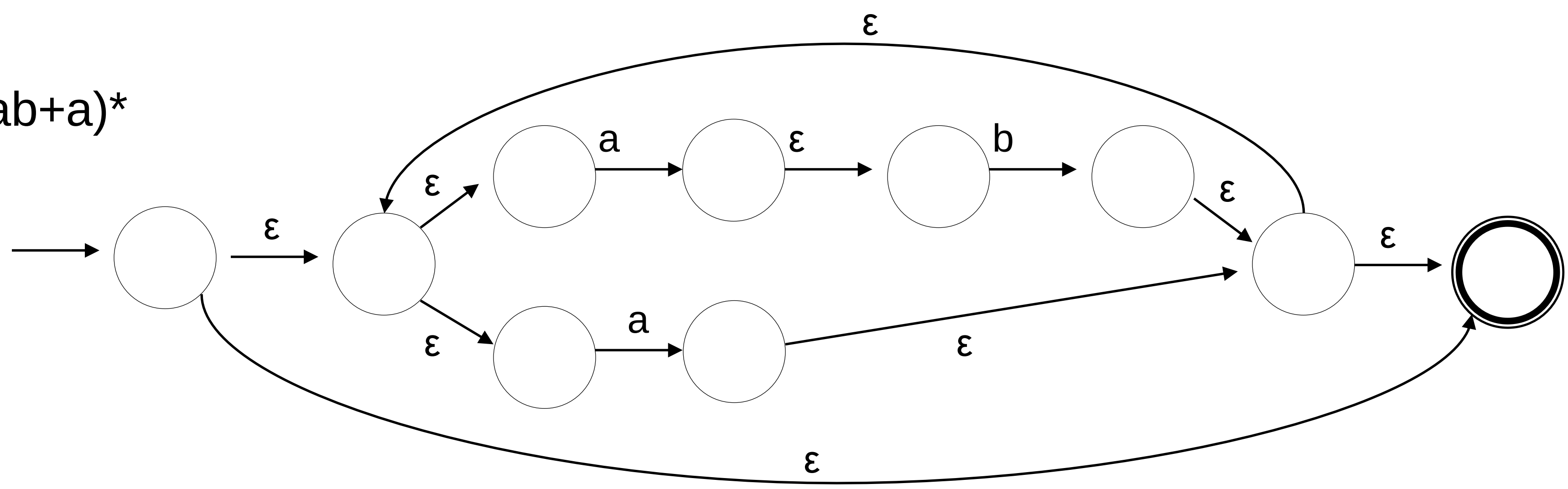


RE to ε-NFA Example (2)

ab+a



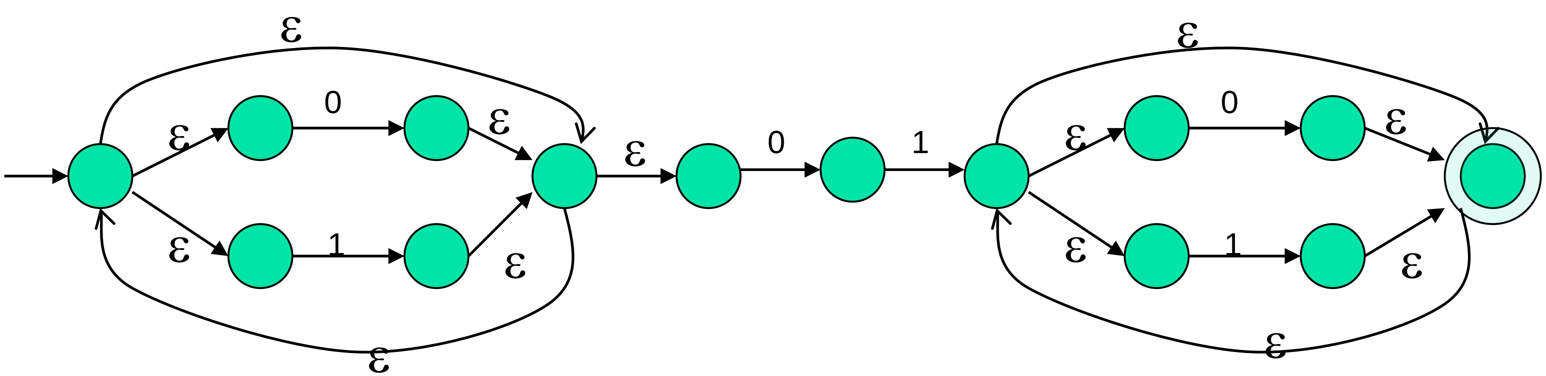
(ab+a)*



RE to ε-NFA construction

<u>Example:</u> (0+1)*01(0+1)*

(0+1)* 01 (0+1)*



Algebraic Laws of Regular Expressions

- Commutative:
  - $E+F = F+E$
- Associative:
  - $(E+F)+G = E+(F+G)$
  - $(EF)G = E(FG)$
- Identity:
  - $E+\Phi = E$
  - $\varepsilon E = E \varepsilon = E$
- Annihilator:
  - $\Phi E = E\Phi = \Phi$

Algebraic Laws…

- Distributive:
  - $E(F+G) = EF + EG$
  - $(F+G)E = FE+GE$
- Idempotent: $E + E = E$
- Involving Kleene closures:
  - $(E^*)^* = E^*$
  - $\Phi^* = \varepsilon$
  - $\varepsilon^* = \varepsilon$
  - $E^+ = EE^*$
  - $E? = \varepsilon + E$

True or False?

Let R and S be two regular expressions. Then:

1. $((R^*)^*)^* = R^*$?
2. $(R+S)^* = R^* + S^*$?
3. $(RS + R)^* RS = (RR^*S)^*$?

Summary

- Regular expressions
- Equivalence to finite automata
- DFA to regular expression conversion
- Regular expression to ε-NFA conversion
- Algebraic laws of regular expressions
- Unix regular expressions and Lexical Analyzer