Theory of Automata

Regular Expressions

Dr. Sabina Akhtar

Revision

REGULAR EXPRESSIONS

Introduction

- Similar to programming language
- Important applications
  - Text-search application
  - Compiler components
- User-friendly alternative to NFA

Regular Expressions vs. Finite Automata

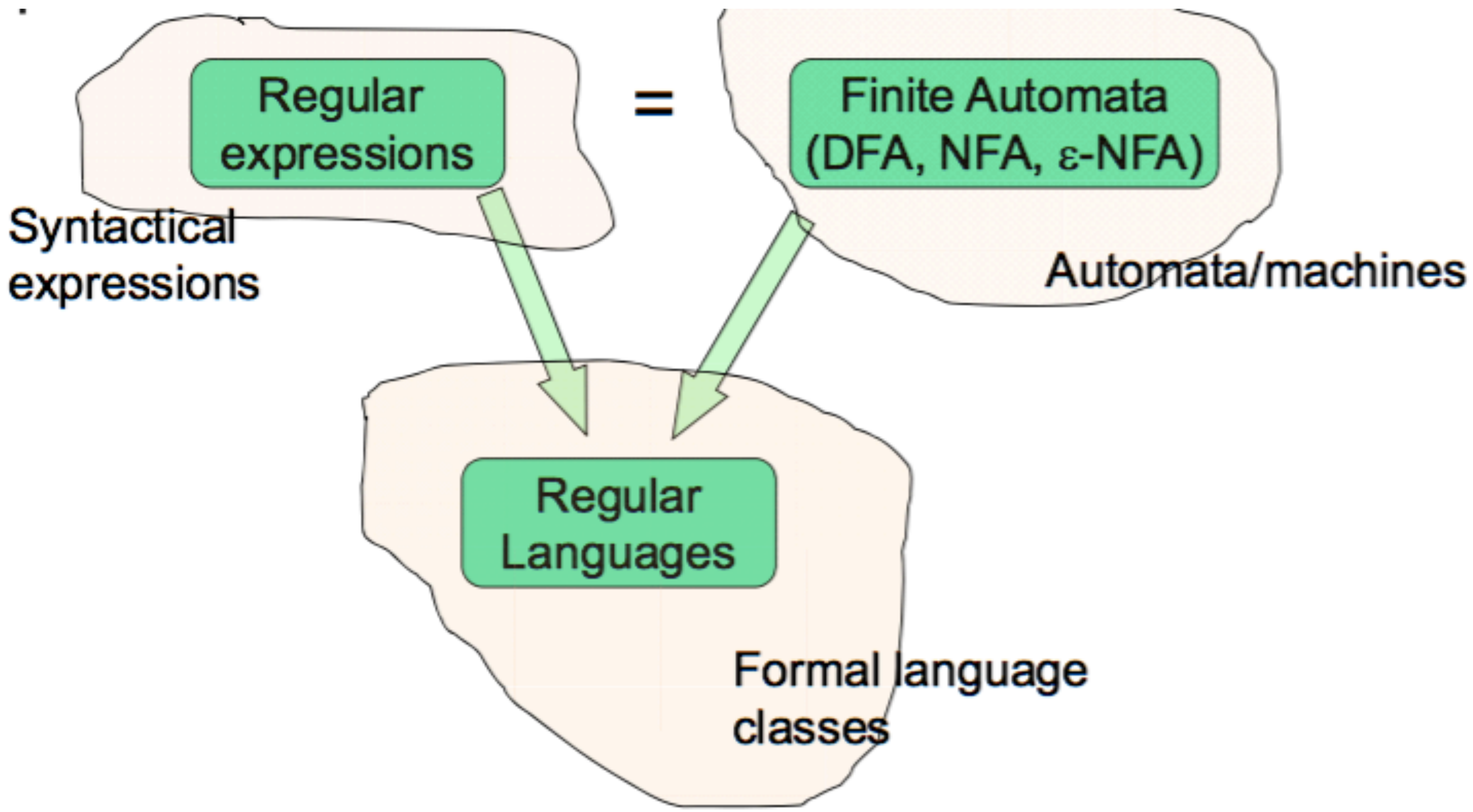
- Offers a declarative way to express the pattern of any string we want to accept
  - E.g., 01*+ 10*

- 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, Javacc

Regular expressions



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))*

Precedence of Operators

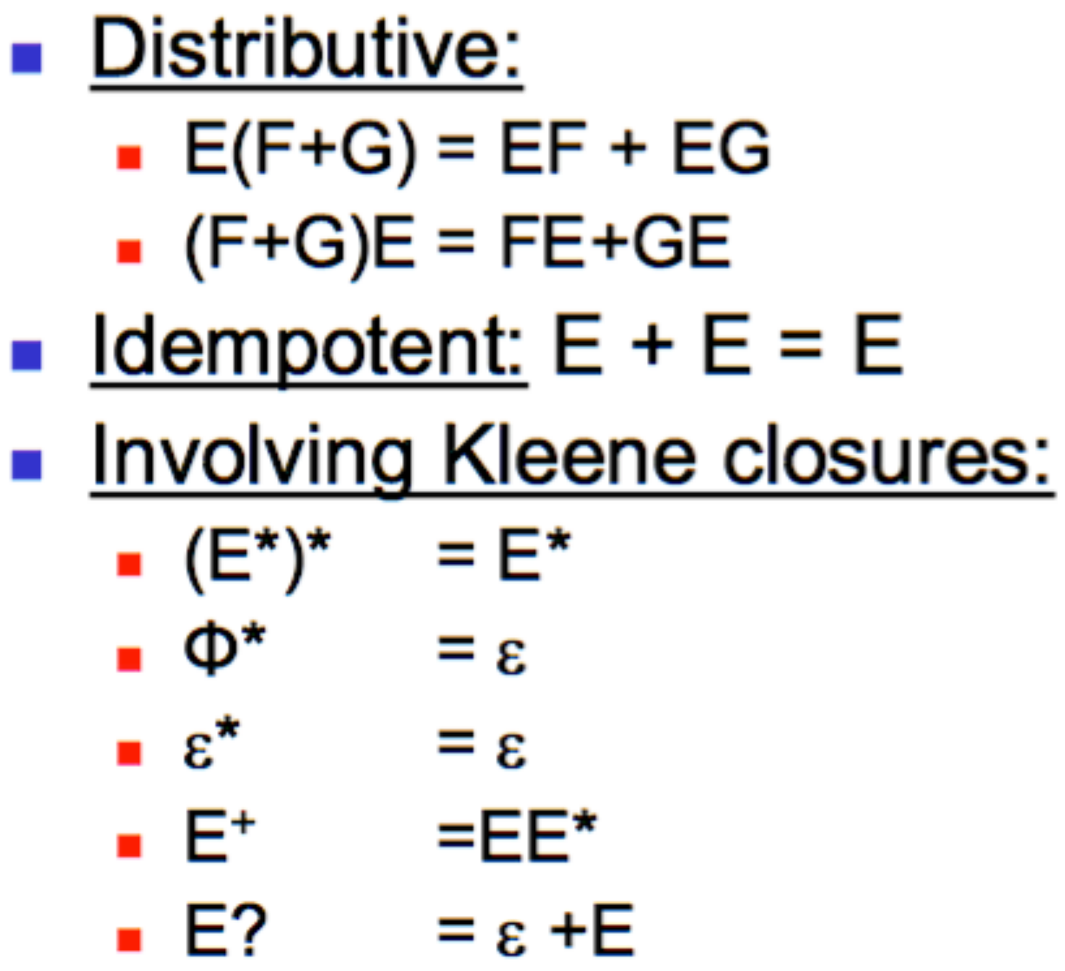
- Highest to lowest
  - * operator (star)
  - . (concatenation)
  - + operator
- Example:

$$01^*+1 = (0.((1)^*))+ 1$$

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:
  - $(E^*)^* \quad = E^*$
  - $\Phi^* \quad = \varepsilon$
  - $\varepsilon^* \quad = \varepsilon$
  - $E^+ \quad =EE^*$
  - $E? \quad = \varepsilon +E$

Example

- Regular expression?
  - Set of all the strings over binary alphabet.
  - L = {01, 11, 00, 10}
  - L = {ε, 0, 00, 000, 0000, …}
  - Set of all the strings containing a single 1

Class Activity

- Example 3.2: Consider the language consisting of strings of a’s and b’s containing aab.
  - Regular expression?

Class Activity

- ***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

- Write the regular expression.

Solution

- 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 = $(\varepsilon + 1)(01)^*(\varepsilon + 0)$

Class Activity

- ***L = { w | w is a binary string that contains 111 as a substring anywhere in the string)***
  - E.g., w = 0101011101 is in L, while w = 10010 is not in L
- ***L_2 = { w | w is a binary string that contains odd number of 1s in the string)***
  - E.g., w = 0100100001 is in L_2, while w = 10010 is not in L_2

- Write the regular expression.

FINITE AUTOMATA AND REGULAR EXPRESSION

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

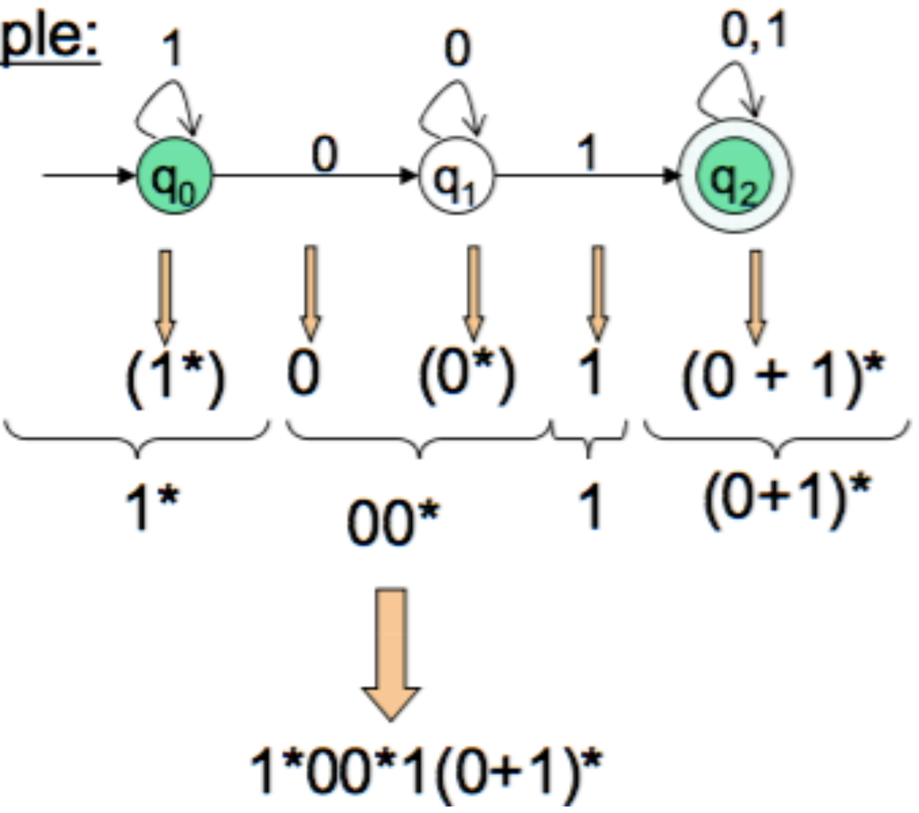
$$\begin{tikzcd}
\varepsilon\text{-NFA} \arrow[dr, shift left] & \text{NFA} \arrow[d, shift right] \\
\text{Reg Ex} \arrow[u, "\text{Theorem 2}"] & \text{DFA} \arrow[ul, shift left] \arrow[u, shift right] \arrow[l, "\text{Theorem 1}"]
\end{tikzcd}$$

Kleene Theorem

DFA to RE construction (using state elimination technique)

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>



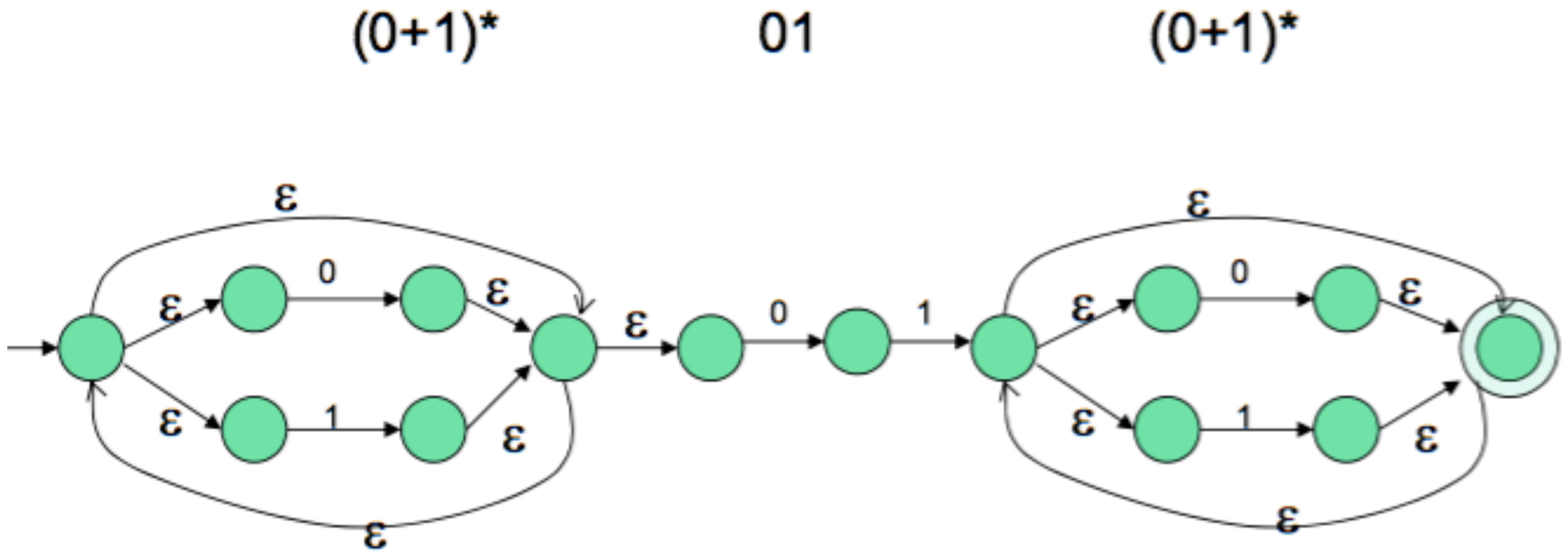
Q) What is the language?

Class Activity

- Example page 94

RE to ?-NFA construction

Example: (0+1)*01(0+1)*



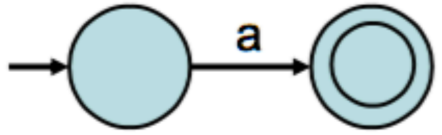
REGULAR EXPRESSIONS DENOTE FA-RECOGNIZABLE LANGUAGES

Languages denoted by regular expressions

- The languages denoted by regular expressions are exactly the regular (FA-recognizable) languages.
- Theorem 1: If R is a regular expression, then L(R) is a regular language (recognized by a FA).
  - Proof: Easy.
- Theorem 2: If L is a regular language, then there is a regular expression R with L = L(R).
  - Proof: Harder, more technical.

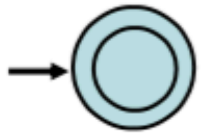
Theorem 1

- Theorem 1: If R is a regular expression, then L(R) is a regular language (recognized by a FA).
- Proof:
  - For each R, define an NFA M with L(M) = L(R).
  - Proceed by induction on the structure of R:
    - Show for the three base cases.
    - Show how to construct NFAs for more complex expressions from NFAs for their subexpressions.
  - Case 1: R = a
    - L(R) = { a }



Accepts only a.

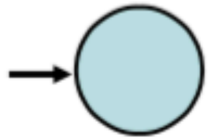
  - Case 2: R = ε
    - L(R) = { ε }



Accepts only ε.

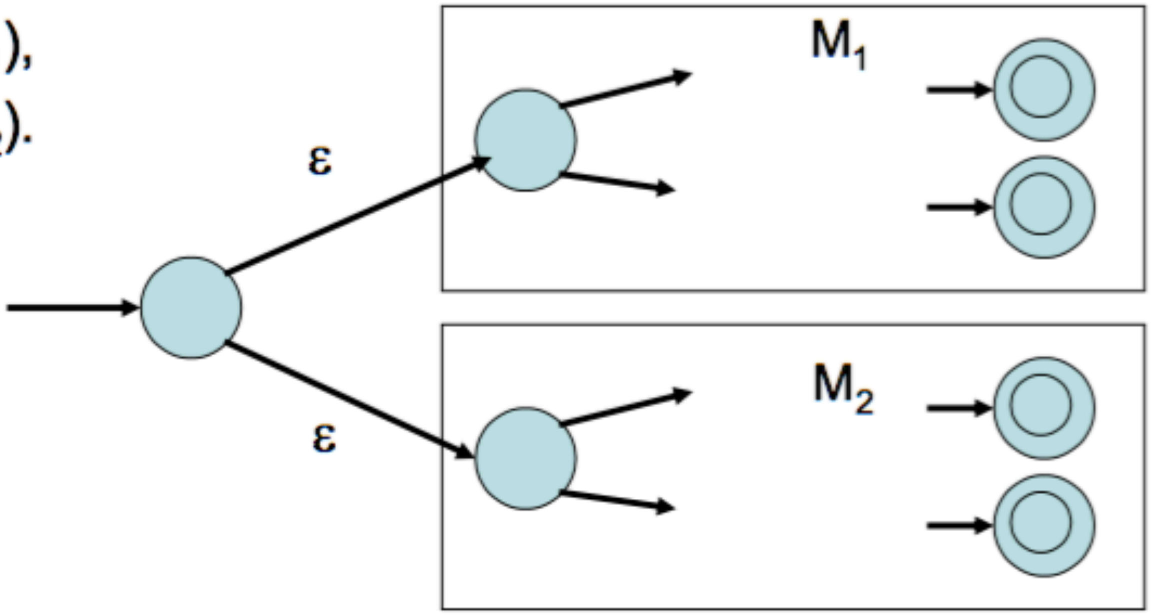
Theorem 1

- Theorem 1: If R is a regular expression, then L(R) is a regular language (recognized by a FA).
- Proof:
  - Case 3: $R = \emptyset$
    - $L(R) = \emptyset$



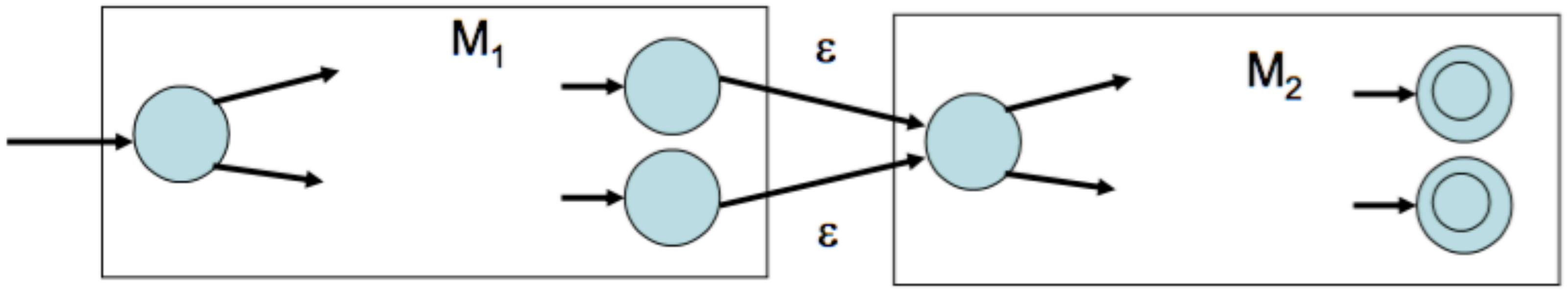
Accepts nothing.

  - Case 4: $R = R_1 \cup R_2$
    - M_1 recognizes $L(R_1)$,
    - M_2 recognizes $L(R_2)$.
    - Same construction we used to show regular languages are closed under union.



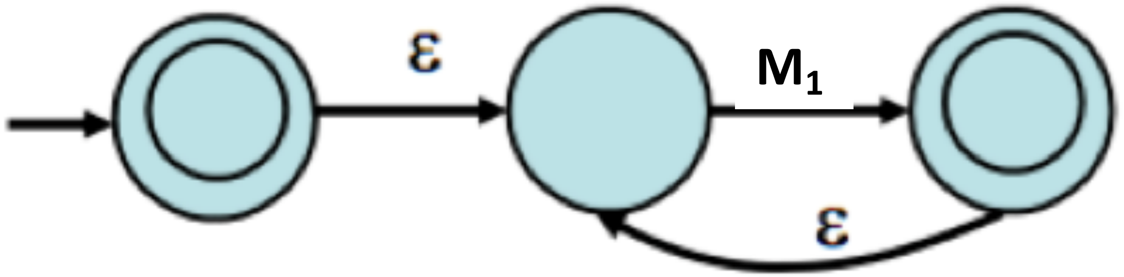
Theorem 1

- Theorem 1: If R is a regular expression, then L(R) is a regular language (recognized by a FA).
- Proof:
  - Case 5: $R = R_1 \circ R_2$
    - M_1 recognizes $L(R_1)$,
    - M_2 recognizes $L(R_2)$.
    - Same construction we used to show regular languages are closed under concatenation.



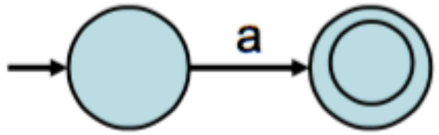
Theorem 1

- Theorem 1: If R is a regular expression, then L(R) is a regular language (recognized by a FA).
- Proof:
  - Case 6: $R = (R_1)^*$
    - M_1 recognizes $L(R_1)$,
    - Same construction we used to show regular languages are closed under star.

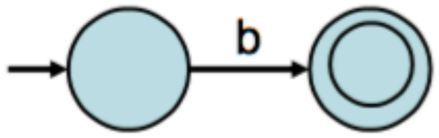


Example for Theorem 1

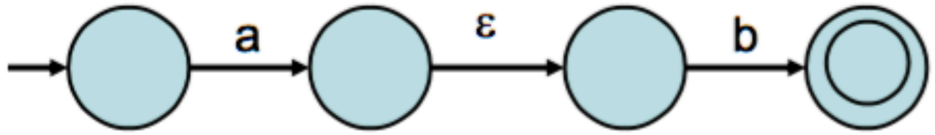
- L = ab ∪ a*
- Construct machines recursively:
- a:



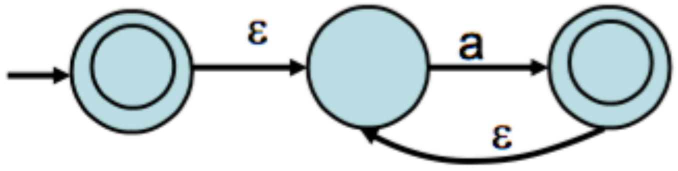
b:



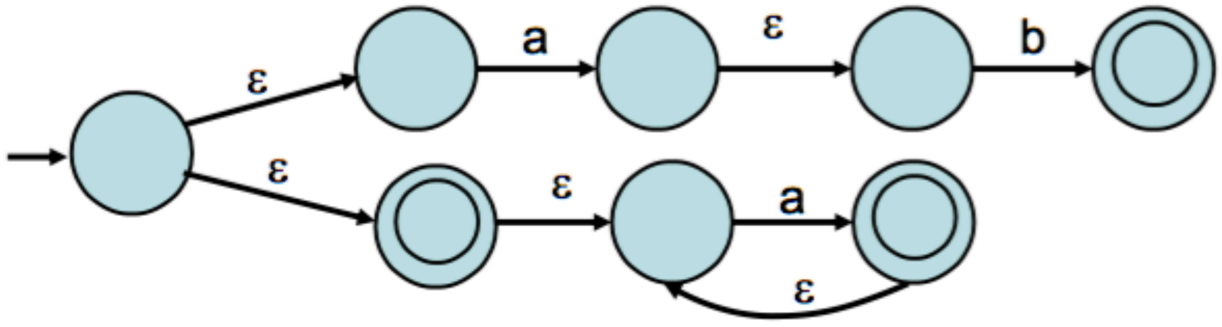
- ab:



- a*:



- ab ∪ a*:



Class Activity

- Convert RE to ε-NFA
  - (0+1)*1(0+1)
  - 01*
  - (0+1)01

References

- Book Chapter 3
- Lectures from Stanford University
  - http://infolab.stanford.edu/~ullman/ialc/spr10/spr10.html#LECTURE%20NOTES
- Lecture by Prof. Nancy Lynch from MIT
- Lectures from Washington State University
  - http://www.eecs.wsu.edu/~ananth/CptS317/Lectures/