Theory of Automata and Languages

Finite Automata and Regular Languages - 2

Fall 2024

Sharif University of Technology

Mehran Moeini Jam

Overview

- Non-determinism
- Non-deterministic Finite Automata
- DFA and NFA Equivalence
- Regular Expressions
- Kleene's Theorem
- Proof of Kleene's Theorem – Part 1
- Proof of Kleene's Theorem – Part 2

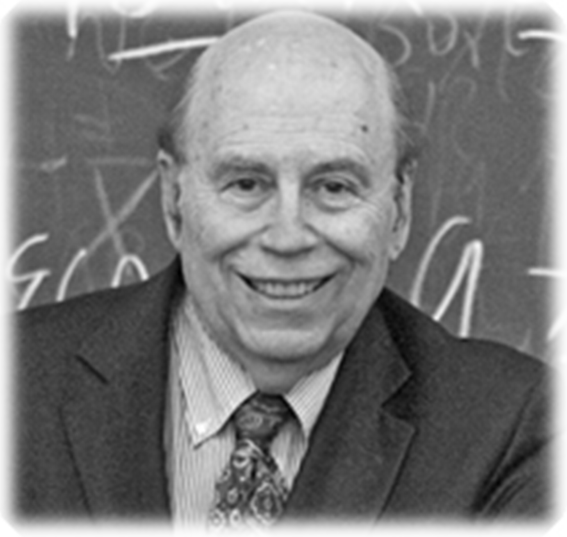
Non-determinism

- The concept of nondeterminism plays a central role in the theory of languages and the theory of computation.

- We first examine this concept in the simpler context of finite automata, and later we shall meet automata whose deterministic and nondeterministic versions are known not to be equivalent, and others for which equivalence is a deep and important open question.

Non-determinism: The Main Contributors

Dana S. Scott



Michael O. Rabin

Non-deterministic Finite Automaton (NFA)

A non-deterministic finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where

- Q is a finite set called states,
- Σ is a finite set called the alphabet,
- $\delta : Q \times \Sigma \longrightarrow 2^Q$ is the transition function,
- $q_0 \in Q$ is the start state, and
- $F \subseteq Q$ is the set of accept (final) states.

String and Language Recognition

- Let M = $(Q, \Sigma, \delta, q_0, F)$ be a non-deterministic finite automata

  and $\omega = a_1 a_2 \cdots a_n$ be a string defined over Σ.

- We say M accepts ω iff a sequence of states $r_0, r_1, \ldots, r_n$ in Q exists such that:

  1. $r_0 = q_0$
  2. $\delta(r_i, a_{i+1}) = r_{i+1}$ for $i = 0, \ldots, n-1$,
  3. $r_n \in F$

String and Language Recognition

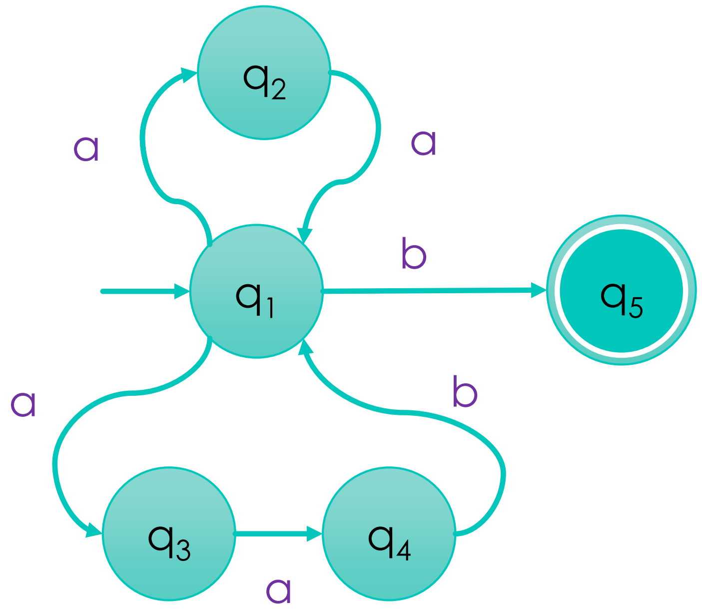
- With the new definition, the automaton might halt on some paths because the current state may not have an explicit transition regarding the last read character. It may also have multiple transition for a single alphabet character.

- So we should no longer think of the computation as a straightforward sequence of steps that each consists a single change of state. Instead, we can think of it as describing a number of different sequences of steps that might be followed.

String and Language Recognition

- We can visualize these sequences by drawing a computation tree.

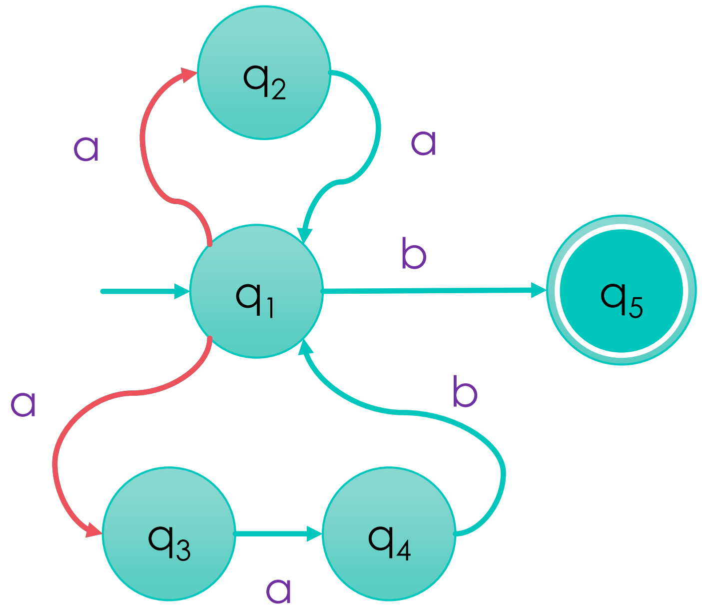
- Consider the following NFA:



String and Language Recognition

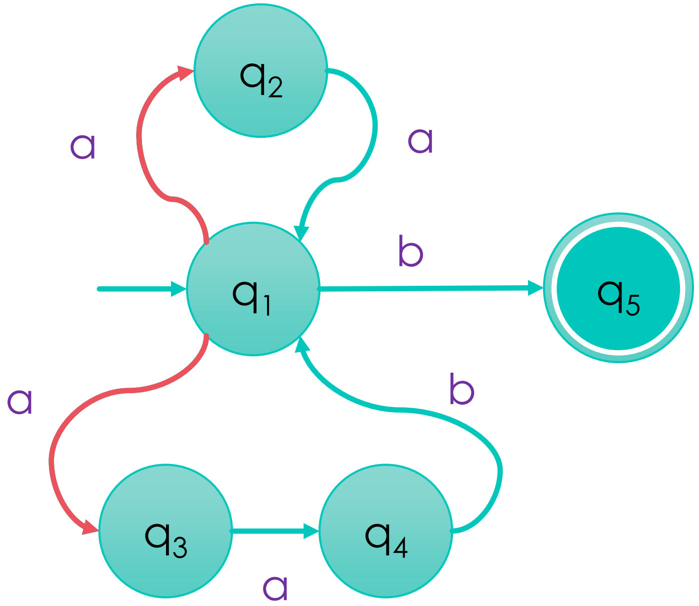
- We can visualize these sequences by drawing a computation tree.

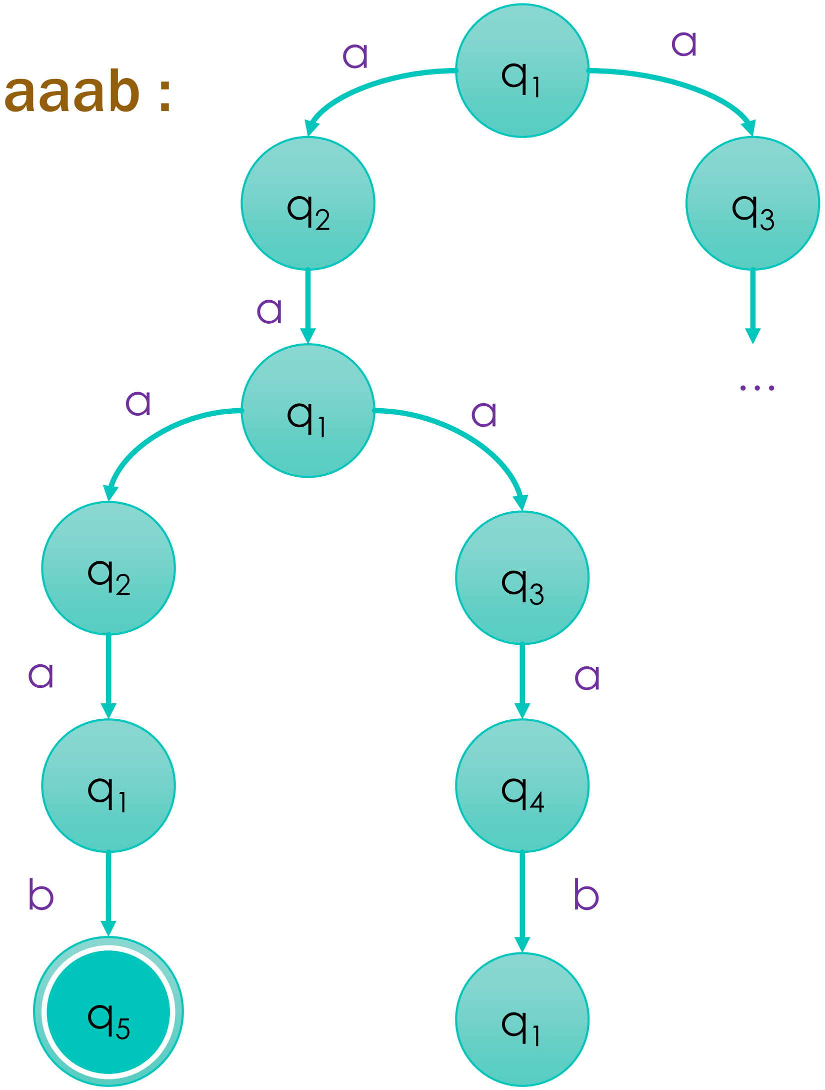
- Consider the following NFA:



String and Language Recognition

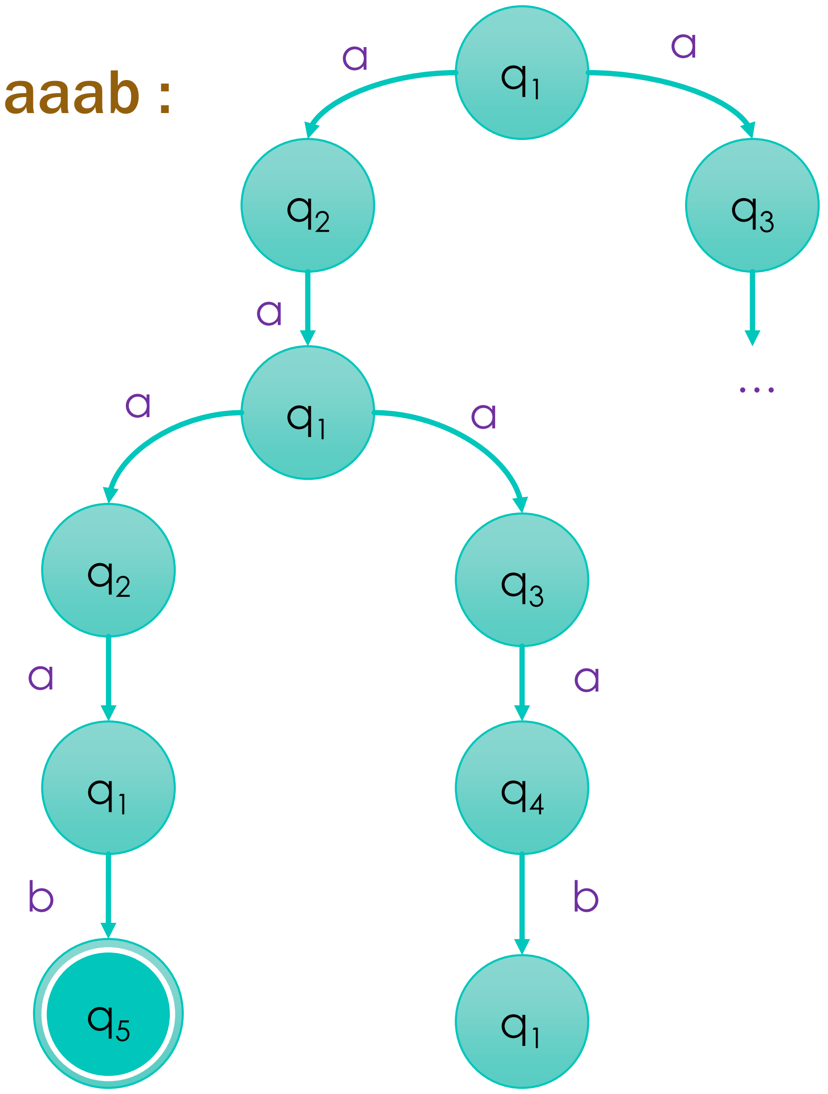
- The corresponding computation tree for the string aaaab :





String and Language Recognition

- The corresponding computation tree for the string aaaab :



- Each level of the tree corresponds to the input (the prefix of the entire input string) read so far, and the states appearing on this level are those in which the device could be, depending on the choices it has made so far.

String and Language Recognition

- Nondeterminism may be viewed as a kind of parallel computation wherein multiple independent “processes” or “threads” can be running concurrently.

- In this alternative thinking, the NFA splits into several children each time it faces multiple choices, and each child proceeds separately.

- If at least one of these children accepts, then the entire computation accepts.

DFA and NFA Equivalence

- For every language $L \subseteq \Sigma^*$ accepted by an NFA = $(Q, \Sigma, \delta, q_0, F)$,

  there exists a DFA = $(Q', \Sigma, \delta', q_0', F')$ that also accepts L.

- So the class of all languages accepted by NFAs is indeed the familiar class of

  regular languages, hence <u>NFAs are no more powerful than DFAs</u>.

DFA and NFA Equivalence

- For every language $L \subseteq \Sigma^*$ accepted by an NFA = $(Q, \Sigma, \delta, q_0, F)$,

  there exists a DFA = $(Q', \Sigma, \delta', q_0', F')$ that also accepts L.

- $Q' = 2^Q$ $\quad q_0' = \{q_0\}$ $\quad F' = \{ R \in Q' \mid R \text{ contains an accept state of NFA}\}$

DFA and NFA Equivalence

- For every language $L \subseteq \Sigma^*$ accepted by an NFA = $(Q, \Sigma, \delta, q_0, F)$,

  there exists a DFA = $(Q', \Sigma, \delta', q_0', F')$ that also accepts L.

- $Q' = 2^Q$ $\qquad q_0' = \{q_0\}$ $\qquad F' = \{ R \in Q' \mid \exists\, r \in R \,.\, r \in F \}$

DFA and NFA Equivalence

- For every language $L \subseteq \Sigma^*$ accepted by an NFA = $(Q, \Sigma, \delta, q_0, F)$,

  there exists a DFA = $(Q', \Sigma, \delta', q_0', F')$ that also accepts L.

- $Q' = 2^Q$ $\quad$ $q_0' = \{q_0\}$ $\quad$ $F' = \{ R \in Q' \mid \exists\, r \in R .\, r \in F \}$

- For $R \in Q'$ and $a \in \Sigma$, let $\delta'(R, a) = \{q \in Q \mid q \in \delta(r, a) \text{ for some } r \in R\}$

  $(R \subseteq Q)$

DFA and NFA Equivalence

- For every language $L \subseteq \Sigma^*$ accepted by an NFA = $(Q, \Sigma, \delta, q_0, F)$,

  there exists a DFA = $(Q', \Sigma, \delta', q_0', F')$ that also accepts L.

- $Q' = 2^Q$ $\quad q_0' = \{q_0\}$ $\quad F' = \{ R \in Q' \mid \exists\, r \in R \,.\, r \in F \}$

- For $R \in Q'$ and $a \in \Sigma$, let $\delta'(R, a) = \bigcup_{r \in R} \delta(r, a)$

NFA with ε-transitions (silent transitions)

A non-deterministic finite automaton is a 5-tuple $(Q, \Sigma_\varepsilon, \delta, q_0, F)$ where

- Q is a finite set called states,
- Σ_ε is a finite set called the alphabet, containing the empty string as well
- $\delta : Q \times \Sigma_\varepsilon \longrightarrow 2^Q$ is the transition function,
- $q_0 \in Q$ is the start state, and
- $F \subseteq Q$ is the set of accept (final) states.

DFA and ε-NFA Equivalence

- For every language $L \subseteq \Sigma^*$ accepted by an ε-NFA = $(Q, \Sigma_\varepsilon, \delta, q_0, F)$,

  there exists a DFA = $(Q', \Sigma, \delta', q_0', F')$ that also accepts L.

- For $R \in Q'$

  Let $E(R) = \{ q \in Q \mid q$ can be reached from R by traveling along 0 or more ε arrows$\}$

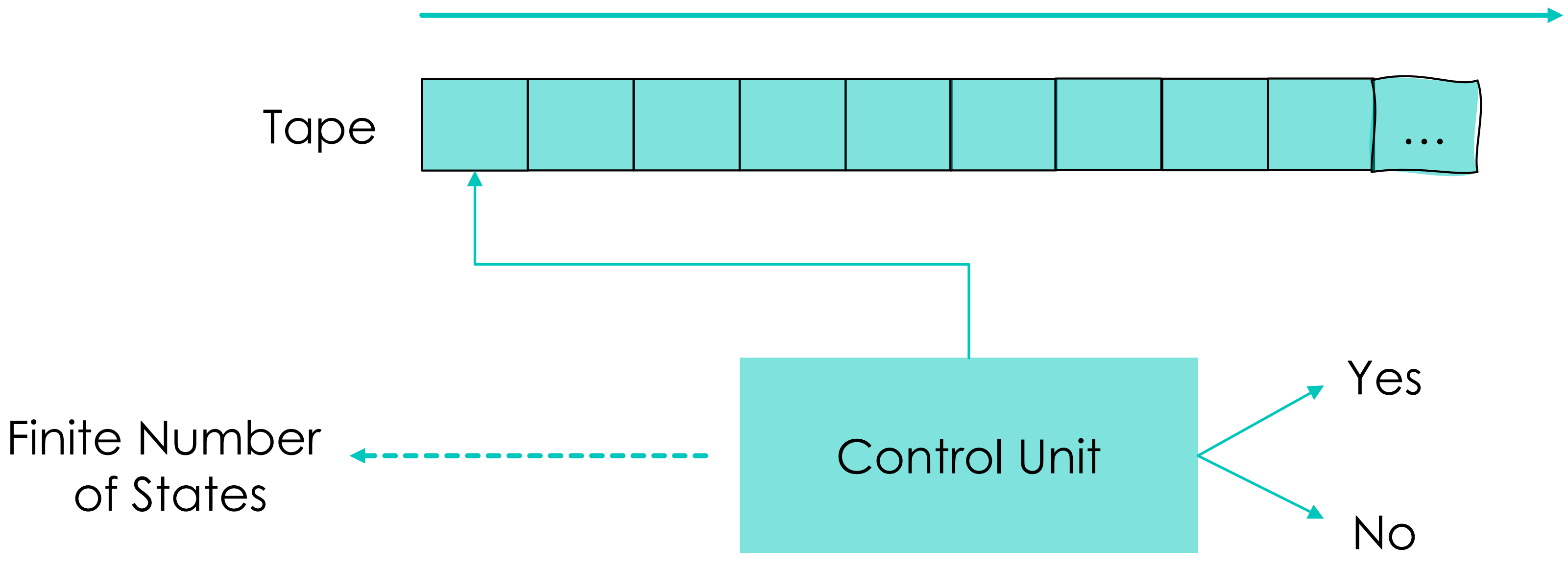
  Then

  $\delta'(R, a) = \{q \in Q \mid q \in E(\delta(r, a))$ for some $r \in R\}$

Notation

Henceforward, each time we mention NFA, we mean ε-NFA and we consider them equal

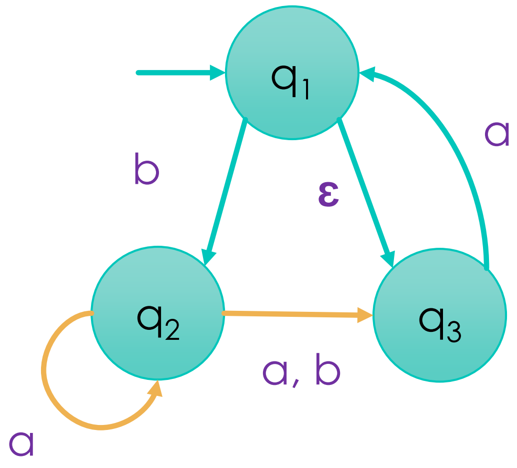
Finite[-State] Automata (as an acceptor)



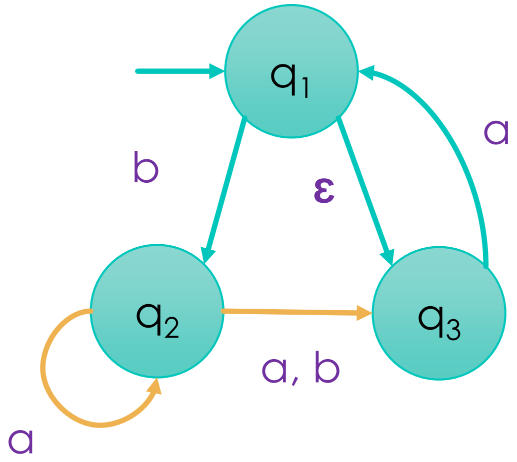
Head of the tape doesn't move on ε-transitions

but a change of state happens in the control unit.

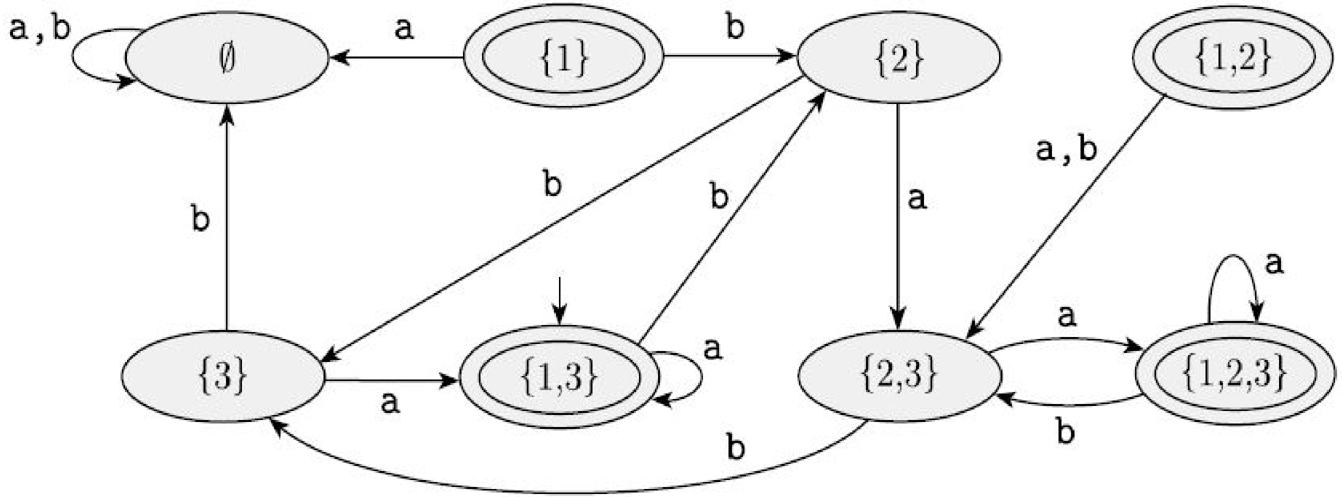
Example



Example



- $Q = \{q_1, q_2, q_3\}$
- $Q' = \{ \emptyset, \{q_1\}, \{q_2\}, \{q_3\}, \{q_1, q_2\}, \{q_1, q_3\}, \{q_2, q_3\}, \{q_1, q_2, q_3\} \}$



Operations on Regular Languages : Reversal

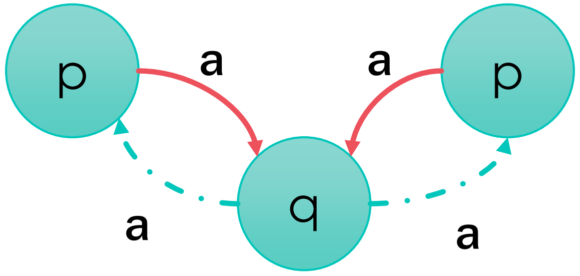
- Suppose $M = (Q, \Sigma, \delta, q_0, F)$ is a deterministic finite automata and $a \in \Sigma$.

- Let M’ be the non-deterministic finite automata $(Q', \Sigma, \delta', q_0', F')$ where

$Q' = Q$ $q_0' = F$ $F' = q_0$

$\delta'(q, a) = \{ p \in Q \mid \delta(p, a) = q \}$

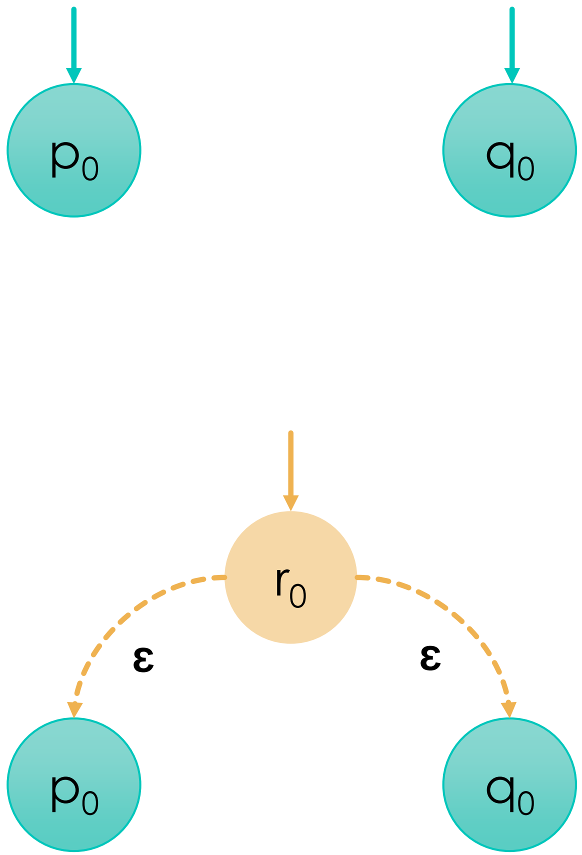
Then M recognizes $L(M)^R$



NFA with More than One Start State

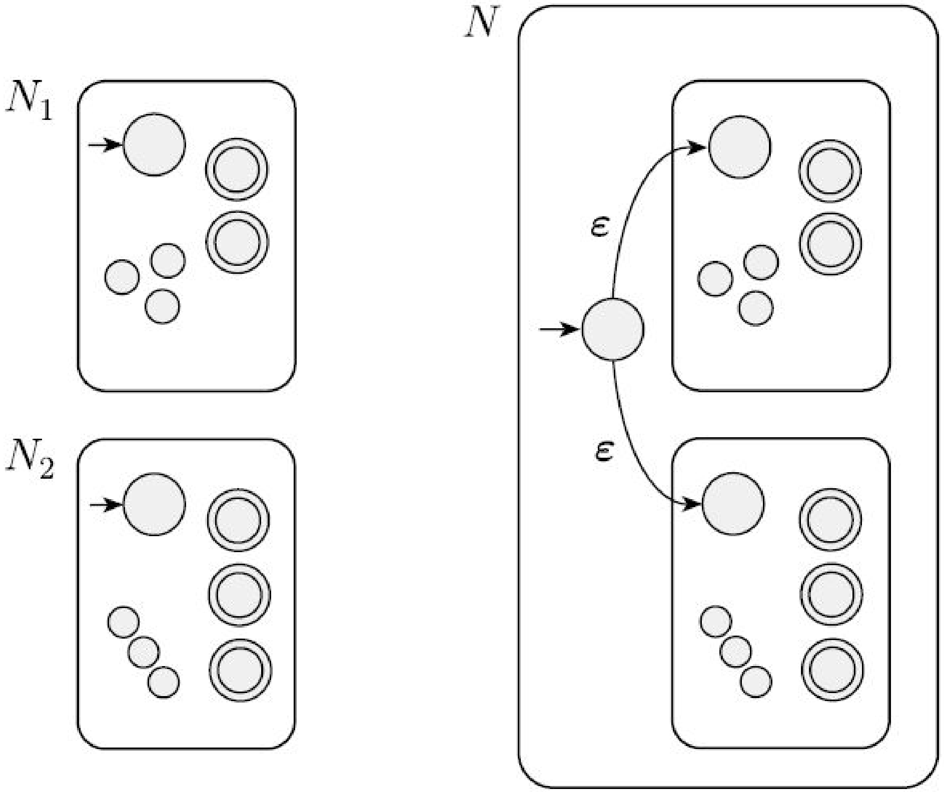
NFA with More than One Start State

- Every NFA with more than one start (initial) state can be transformed into an NFA with only one start state.

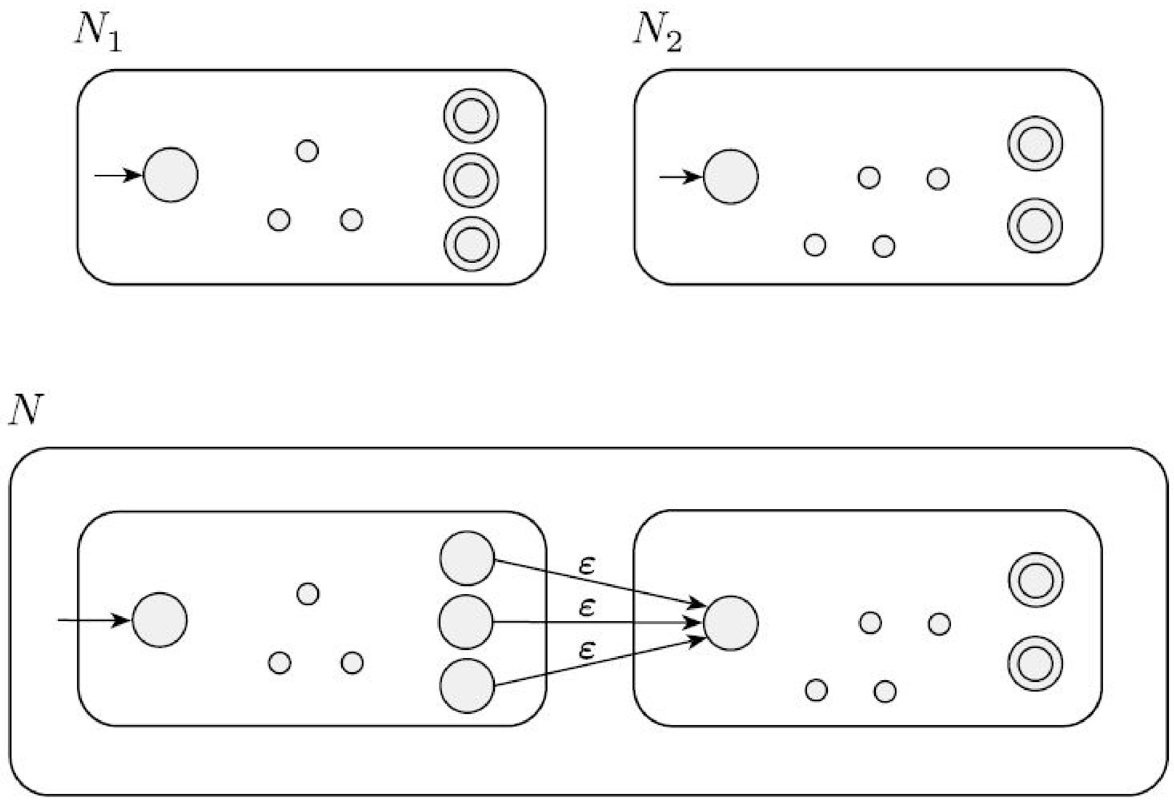


- What about DFA?
- What about Final States?

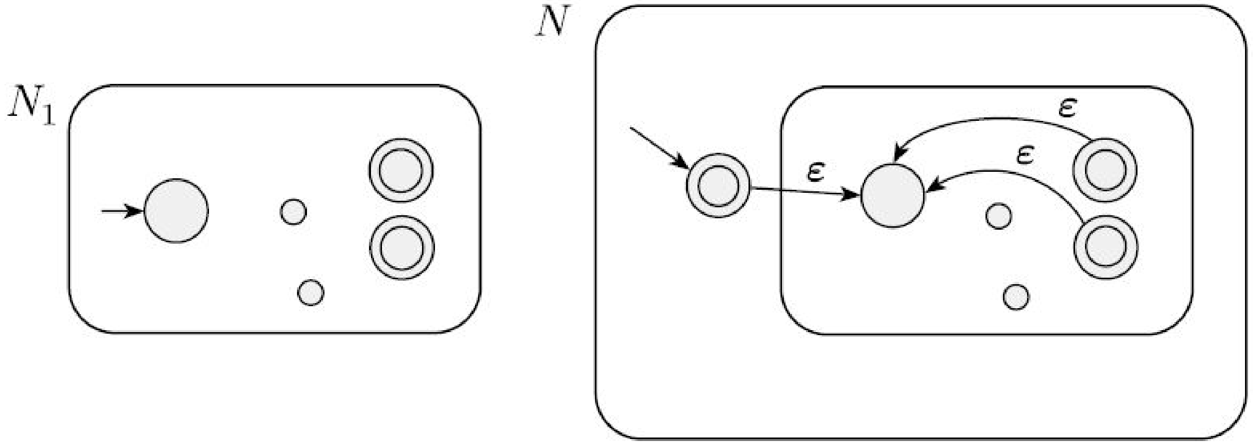
Operations on Regular Languages : Union



Operations on Regular Languages : Concatenation



Operations on Regular Languages : Kleene Star



Regular Expressions (Regex)

- In arithmetic, we can use the operations + and × to build up expressions such as

$$(5 + 3) \times 4$$

- Similarly, we can use the regular operations to build up expressions describing languages, which are called regular expressions, such as

$$(0 \cup 1)0^*$$

Regular Expressions (Regex)

- Consider R as the set of all regular expressions on an alphabet Σ, and $a \in \Sigma$ as a member of that alphabet, ε as the empty string and $\emptyset$ as the empty set:

- $a \in R$ $\varepsilon \in R$ $\emptyset \in R$
- $(r_1 \cup r_2) \in R$ where $r_1, r_2 \in R$
- $(r_1 . r_2) \in R$ where $r_1, r_2 \in R$
- $(r_1{}^*) \in R$ where $r_1 \in R$

Regular Expressions (Regex)

- A regular expression for a language is a slightly more user-friendly formula than using the regular operators on set elements.

- The only real difference is that in a regular expression, curly braces { } are omitted and parentheses ( ) are used whenever necessary.

- The precedence order is: star, then concatenation, then union.

Completeness

- The combination of star, concatenation, and union is complete to express any regular language.
- Try to add other operations, such as intersection, or complement, or even replace some operators with others.
- Does the expressive power increase or decrease by these modifications?

Identities

- $r_1 \cup r_2 = r_2 \cup r_1$
- $(r_1 \cup r_2) \cup r_3 = r_1 \cup (r_2 \cup r_3)$
- $(r_1 r_2) r_3 = r_1 (r_2 r_3)$
- $\emptyset \cup r = r \cup \emptyset = r$
- $\varepsilon r = r \varepsilon = r$
- $\emptyset r = r \emptyset = \emptyset$
- $r_1 (r_2 \cup r_3) = r_1 r_2 \cup r_1 r_3$

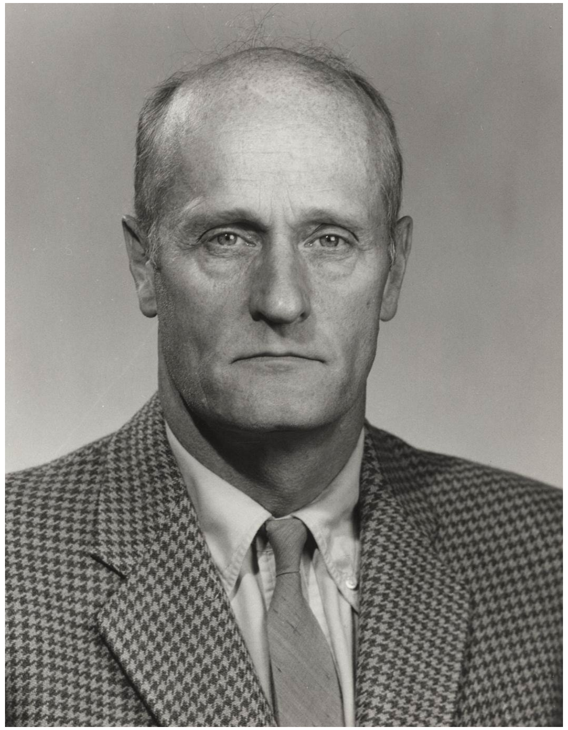
Identities

- $(r_2 \cup r_3)r_1 = r_2r_1 \cup r_3r_1$
- $r_1 \cup r_1 = r_1$
- $(r_1{}^*)^* = r_1{}^*$
- $\emptyset^* = \varepsilon$
- $\varepsilon^* = \varepsilon$
- $(r_1r_2)^*r_1 = r_1(r_2r_1)^*$

Kleene's Theorem, Equivalence with NFA

- For every regular expression, there exists a finite automaton that recognizes the same language that the regular expression describes.

- For every finite automaton, there exists a regular expression that describes the same language that the automaton recognize.



Kleene’s Theorem – Part 1

- For every regular expression, there exists a finite automaton that accepts the same language that the regular expression describes.

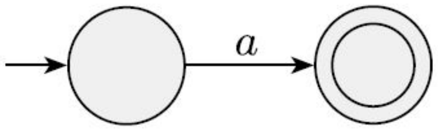
- We show that how a regular expression can be transformed to a finite automaton, by considering the six cases in the previous recursive formal definition.

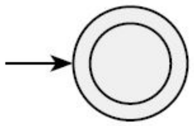
Kleene's Theorem – Part 1

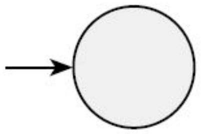
- Consider R as the set of all regular expressions on an alphabet Σ, and $a \in \Sigma$ as a member of that alphabet, ε as the empty string and ∅ as the empty set:

- $a \in R$ $\varepsilon \in R$ $\emptyset \in R$

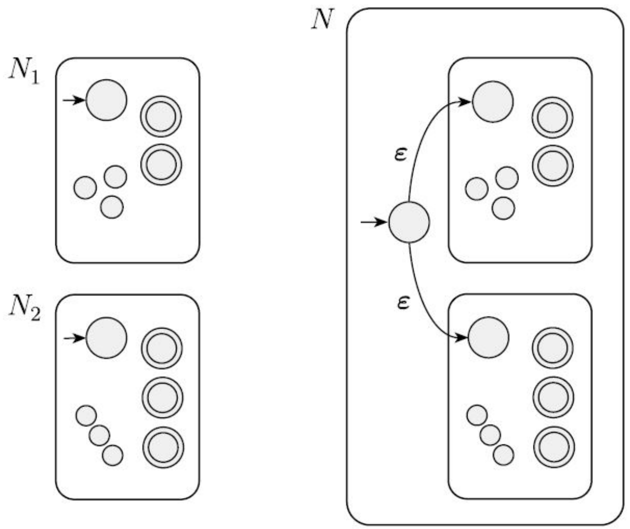
$L(a) = \{a\}$ $L(\varepsilon) = \{\varepsilon\}$ $L(\emptyset) = \emptyset$



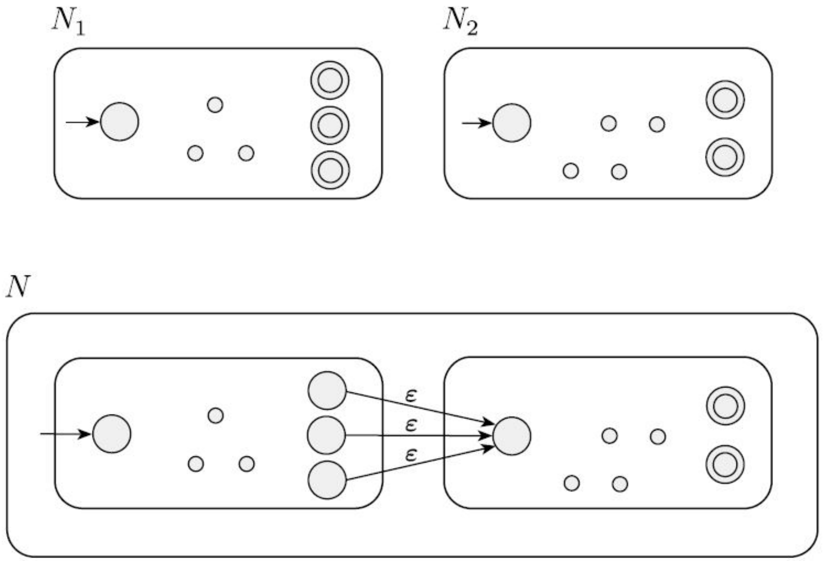




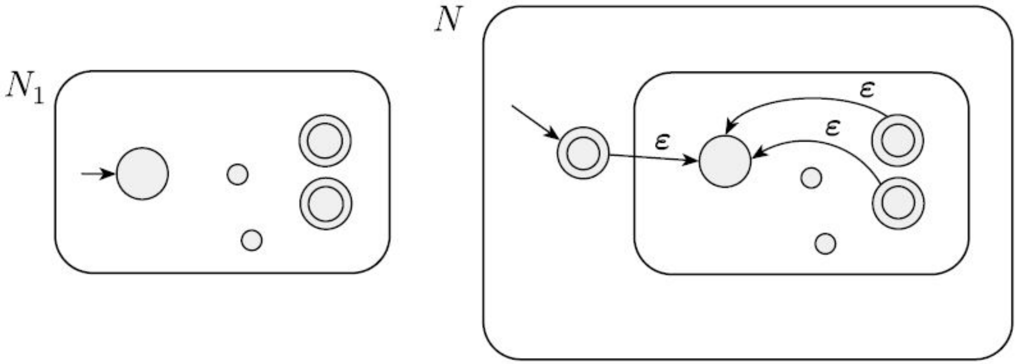
- $(R_1 \cup R_2) \in R$ where $R_1, R_2 \in R$



- $(R_1 . R_2) \in R$ where $R_1, R_2 \in R$



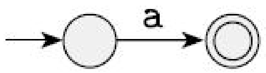
- $(R_1{}^*) \in R$ where $R_1 \in R$



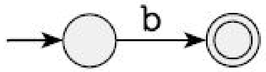
Example

- Consider the language described by (ab ∪ a)*

a



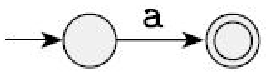
b



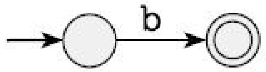
Example

- Consider the language described by (ab ∪ a)*

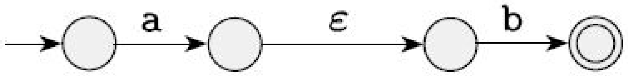
a



b

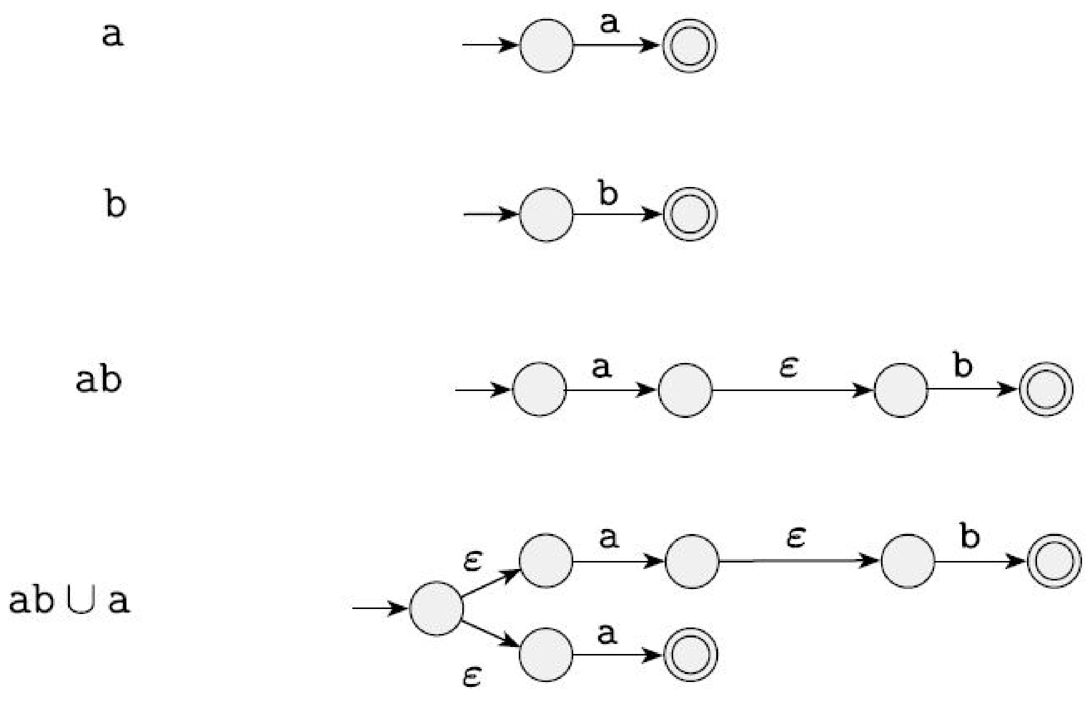


ab



Example

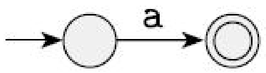
- Consider the language described by (ab ∪ a)*



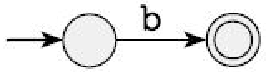
Example

- Consider the language described by (ab ∪ a)*

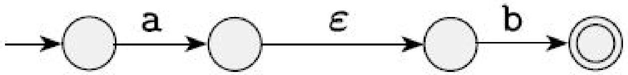
a



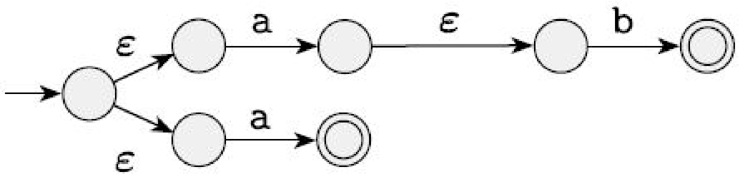
b



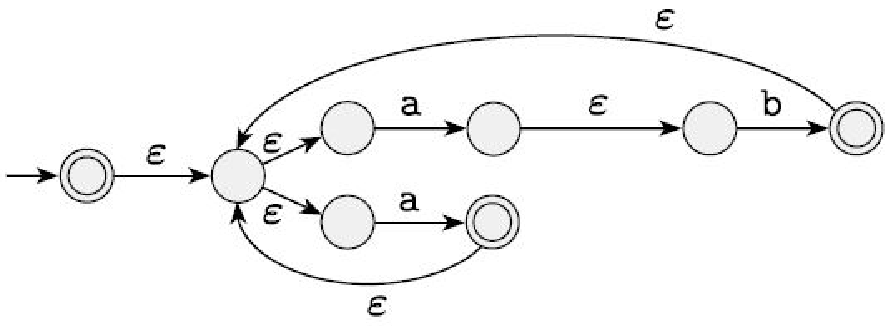
ab



ab ∪ a

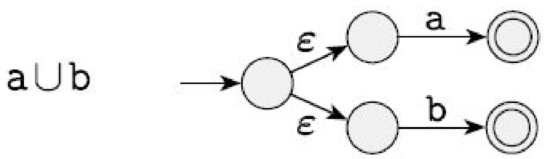


(ab ∪ a)*



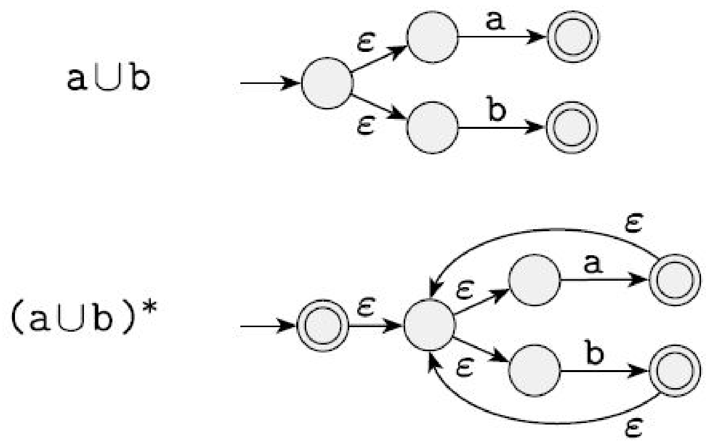
Example

- Consider the language described by (a ∪ b)*aba



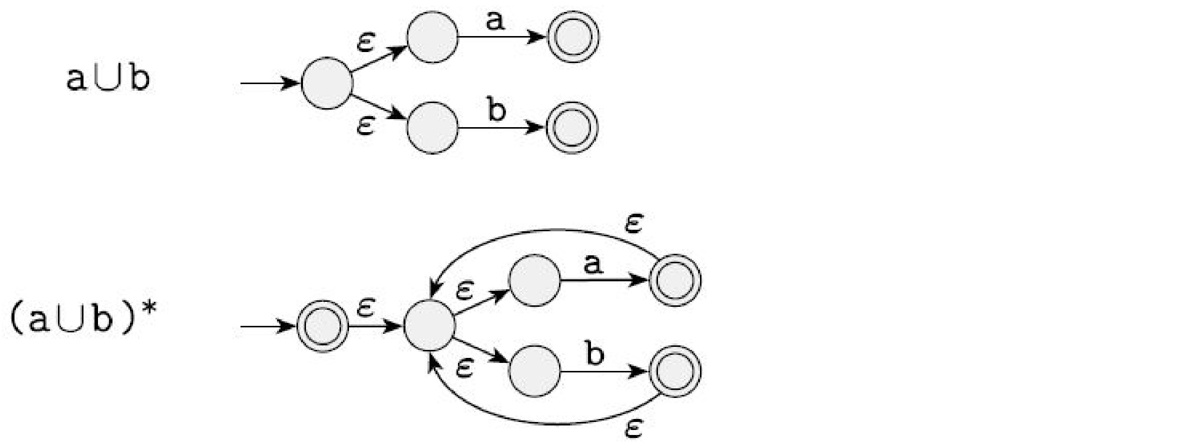
Example

- Consider the language described by (a ∪ b)*aba



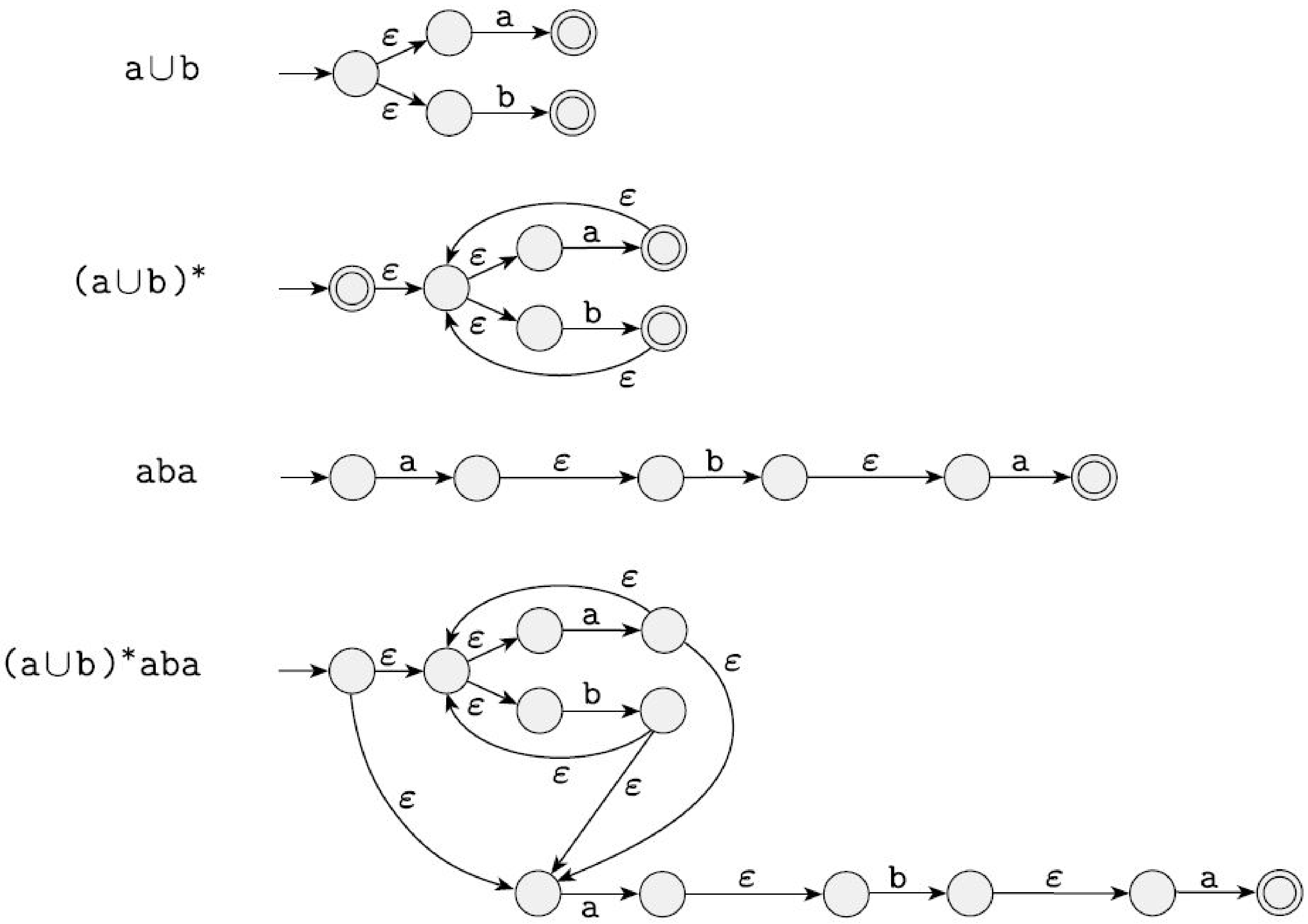
Example

- Consider the language described by (a ∪ b)*aba



Example

- Consider the language described by (a ∪ b)*aba



Kleene's Theorem – Part 2

- For every finite automaton, there exists a regular expression that describes the same language that the automaton accepts.

- We need some additional definitions first.

R_{ijk} – Informal Definition

- Consider a finite automaton with n states numbered from 1 to n.

  Let R(i,j,k) be the set of all strings ω over the same alphabet which starting from state i, the automaton goes through a set of states where each state's number is k or lower and eventually reaches state j.

- In other words, for every string in this set, the biggest state number we see while going from i to j is k.

Some More Conditions

- We don't consider i and j themselves as the middle states.

- k can also be zero, that is sole transitions for single character strings with no other states in between (state 0 is in between, which is undefined)

R_{ijk} – Formal Definition

- $R(i,j,0) = \{ a \in \Sigma \mid \delta(q_i, a) = q_j \}$ $\quad$ $i \neq j$

  $R(i,j,0) = \{ a \in \Sigma \mid \delta(q_i, a) = q_j \} \cup \{\varepsilon\}$ $\quad$ $i = j$

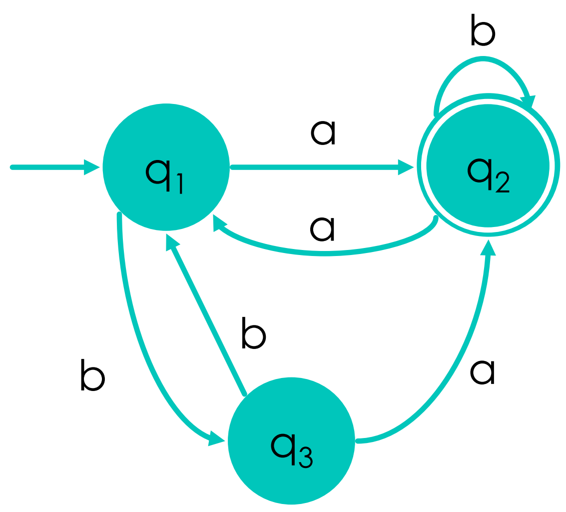
- $R(i,j,k) = R(i,k,k\text{-}1) \left(R(k,k,k\text{-}1) \right)^* R(k,j,k\text{-}1) \cup \mathbf{R(i,j,k\text{-}1)}$ $\quad$ $k > 0$

R_{ijk} – Formal Definition

- $R(i,j,0) = \{ a \in \Sigma \mid \delta(q_i, a) = q_j \}$ $\quad i \neq j$

  $R(i,j,0) = \{ a \in \Sigma \mid \delta(q_i, a) = q_j \} \cup \{\varepsilon\}$ $\quad i = j$

- $R(i,j,k) = R(i,k,k\text{-}1) \left(R(k, k, k\text{-}1) \right)^* R(k, j, k\text{-}1) \;\cup\; R(i,j,k\text{-}1)$ $\quad k > 0$



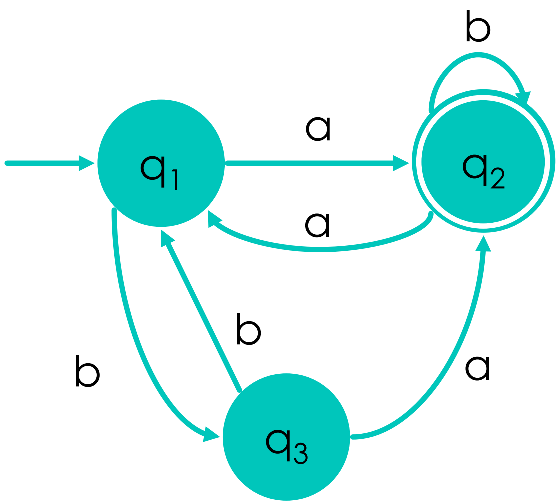
$R(1,2,3) = R(1,3,2)\; R(3,3,2)^*\; R(3,2,2) \;\cup\; R(1,2,2)$

$R(1,3,2) = R(1,2,1)\; R(2,2,1)^*\; R(2,3,1) \;\cup\; R(1,3,1)$

$R(3,3,2) = R(3,3,1)\; R(3,3,1)^*\; R(3,3,1) \;\cup\; R(3,3,1)$

$R(3,2,2) = R(3,2,1)\; R(2,2,1)^*\; R(2,2,1) \;\cup\; R(3,2,1)$

R_{ijk} – Formal Definition



$$R(1,2,3) = R(1,3,2)\ R(3,3,2)^*\ R(3,2,2) \ \cup\ R(1,2,2)$$

...

$R(1,3,0) = \{b\}$ $\Longrightarrow$ b

$R(1,1,0) = \{\varepsilon\}$ $\Longrightarrow$ ε

$R(2,2,0) = \{b\} \cup \{\varepsilon\}$ $\Longrightarrow$ $b \cup \varepsilon$

Generalized Non-deterministic Finite Automata (GNFA)

A Generalized Non-deterministic finite automaton is a 5-tuple $(Q, \Sigma, \delta, q_{start}, q_{acc})$ where

- Q is a finite set called **states**,
- Σ is a finite set called the **alphabet**,
- $\delta : Q - \{q_{acc}\} \times Q - \{q_{start}\} \longrightarrow R$ is the **transition function**,
- $q_{start} \in Q$ is the **start state**, and
- $q_{acc} \in Q$ is the **accept state.**

Generalized Non-deterministic Finite Automata (GNFA)

$$\delta : Q - \{q_{acc}\} \times Q - \{q_{start}\} \longrightarrow R$$

- The start state has a transition to every other state but doesn't have any incoming
- The accept state receives a transition from every other state but doesn't have any outgoing
- Except for start and accept, every other state has a transition to every other state and also to itself

Convert FA to GNFA

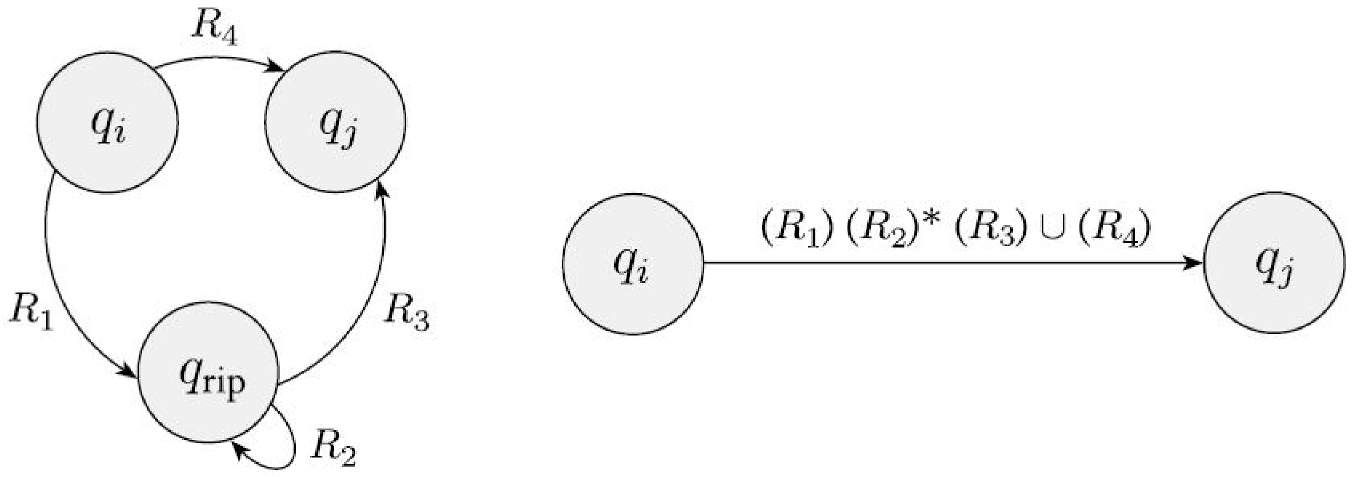
1. Modify every single transition so the labels represent regular expressions
2. Add the start state with an ε-transition to the existing start state
3. Add the accept state which receives an ε-transition from every accept state
4. Add ∅-transitions where no other transition exists

Kleene's Theorem – Part 2

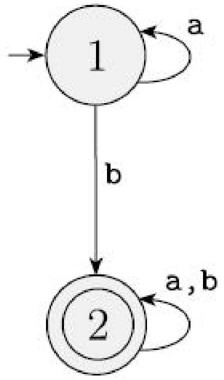
1. Modify every single transition so the labels represent regular expressions
2. Add the start state with an ε-transition to the existing start state
3. Add the accept state which receives an ε-transition from every accept state
4. Add ∅-transitions where no other transition exists
5. Eliminate states one by one, until only q_{start} and q_{acc} remain

Kleene's Theorem – Part 2

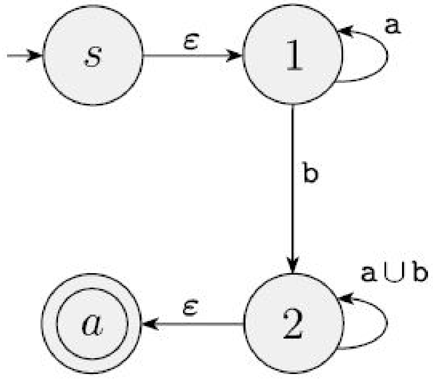
Eliminate states one by one, until only q_{start} and q_{acc} remain



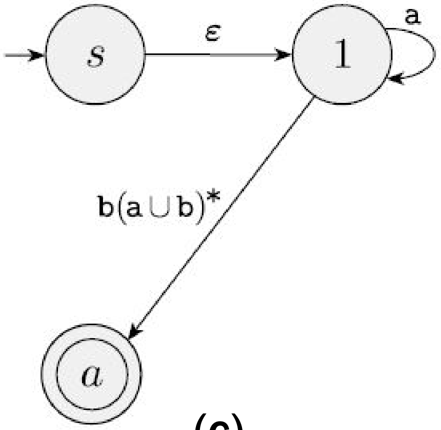
Example



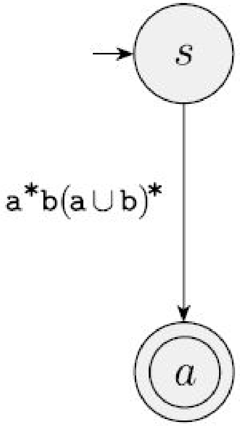
(a)



(b)



(c)



(d)

Next Set of Slides!