Theory of Computation
Chapter 1

Finite Automata

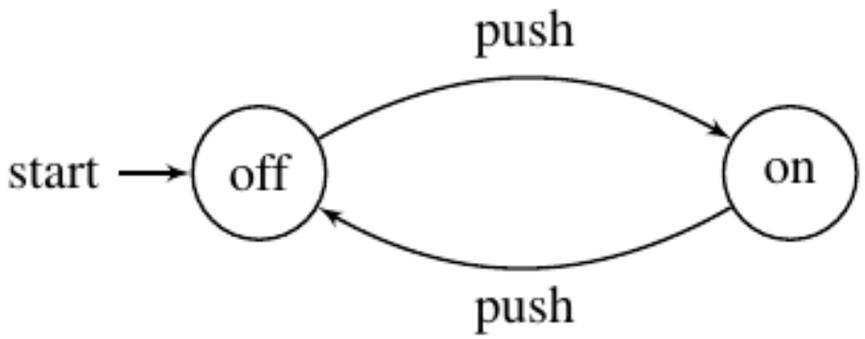
School of Engineering | Computer Science

Grace Murray Hopper
1906-1992

- PhD in mathematics, Yale, 1934
- Volunteered for Navy during WWII, became a programmer on Harvard Mark I
- Helped develop UNIVAC I
- Developed first compiler
- Led development of COBOL programming language
- Popularized the term “debugging”
- First female Admiral in US Navy
- USS Hopper, launched in 1996, named for her

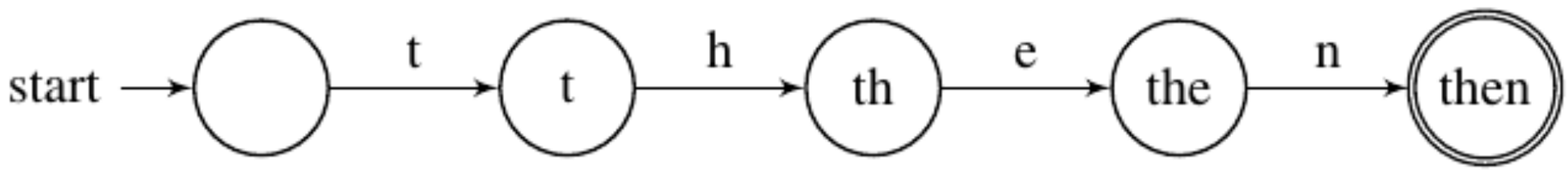
Finite Automata

- Good models for computers with extremely limited amount of memory
- Ex: An on/off switch



Finite Automata

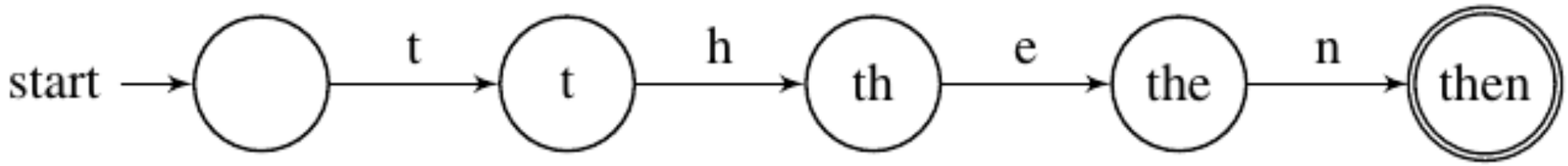
- Ex: Part of a lexical analyzer for recognizing the word then



- Move from state to state on input given, starting with start state
- States with double circles are accept states. All others are reject states.
- Final state is where end up after processing the given input.

Finite Automata

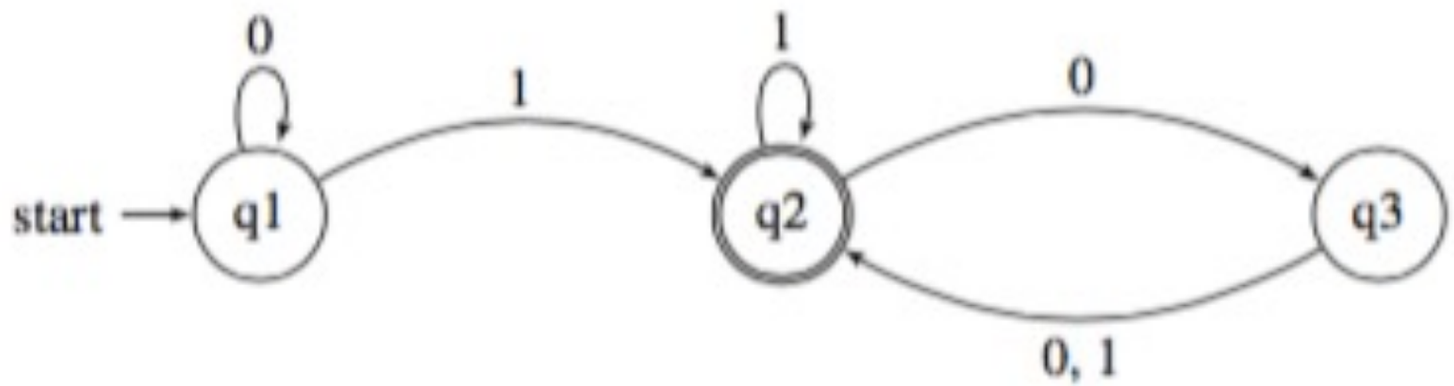
- Ex: Part of a lexical analyzer for recognizing the word then



- Input 1:
  - String “the”
  - States: t, th, the – end on a non-accepting state
- Input 2:
  - String “then”
  - States: t, th, the, then – end on an accepting state

Finite Automata

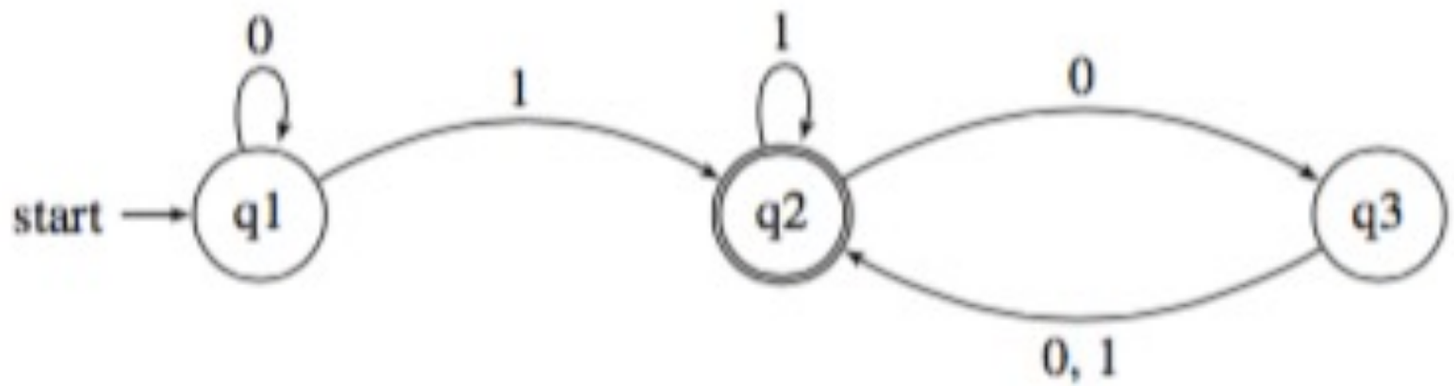
- Formal Example
  - Alphabet: $\Sigma = \{0, 1\}$
  - State Diagram:



- Start State is q1, Accept state is q2
- Arrow are the transitions between states

Finite Automata

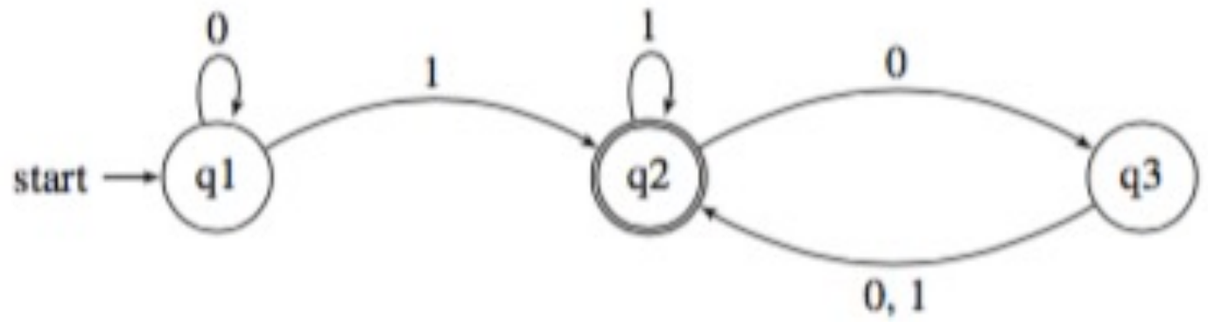
- Formal Example
  - Alphabet: $\Sigma = \{0, 1\}$
  - State Diagram:



- Will this automaton accept the string 1101?
  - q1 → q2 → q2 → q3 → q2 (accept state – Yes!)
    1 1 0 1
  - Move from state to state on string characters. When characters end, determine if in an accept state or not

Finite Automata

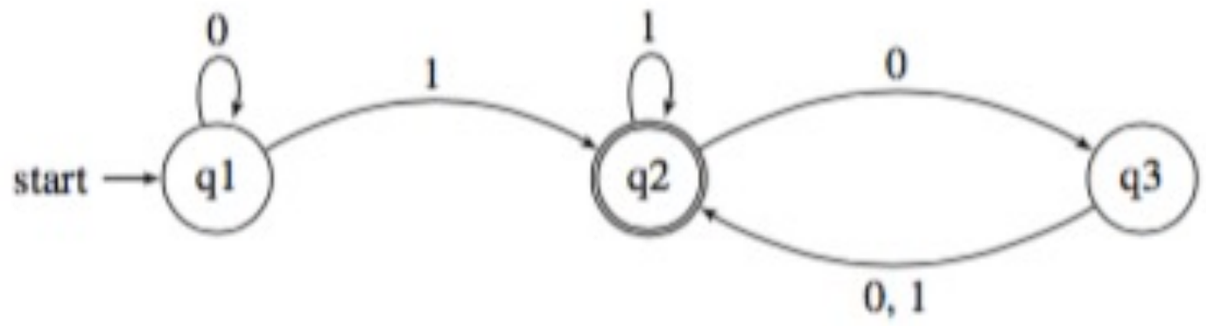
- Formal Example
  - Alphabet: $\Sigma = \{0, 1\}$
  - State Diagram:



- Will this automaton accept the string 0001?
  - q1 → q1 → q1 → q1 → q2 (accept state – Yes!)
    0 0 0 1
- Will this automaton accept the empty string ε?
  - q1 (not an accept state – No)

Finite Automata

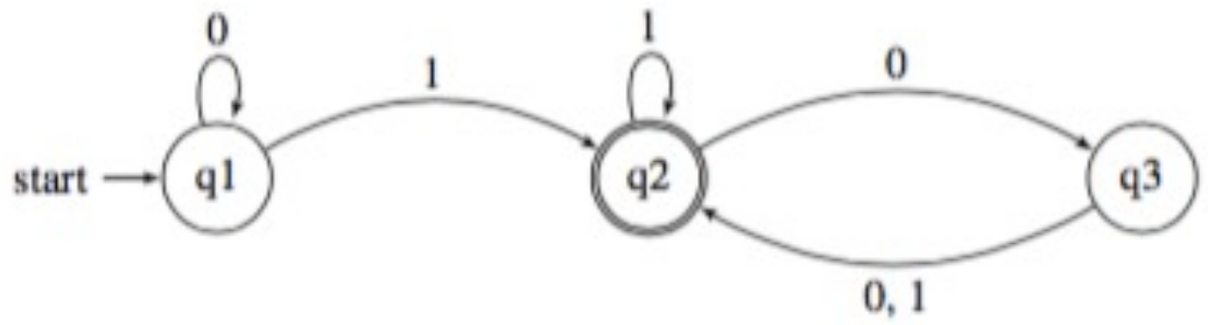
- Formal Example
  - Alphabet: $\Sigma = \{0, 1\}$
  - State Diagram:



  - Will this automaton accept:
    - 0?
    - 0110101?
    - 011000?

Finite Automata

- Formal Example
  - Alphabet: $\Sigma = \{0, 1\}$
  - State Diagram:



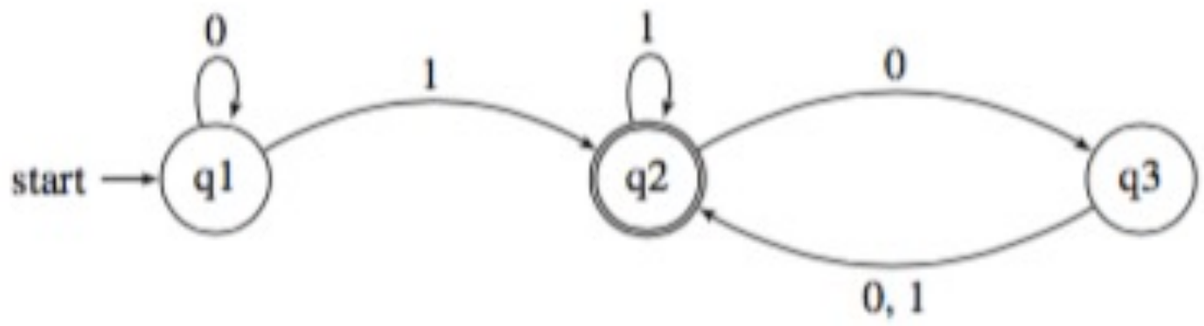
  - Will this automaton accept:
    - 0? Ans: $q1 \xrightarrow{1} q1$ (no)
    - 0110101?
      - Ans: $q1 \xrightarrow{0} q1 \xrightarrow{1} q2 \xrightarrow{1} q2 \xrightarrow{0} q3 \xrightarrow{1} q2 \xrightarrow{0} q3 \xrightarrow{1} q2$ (yes)
    - 011000?
      - Ans: $q1 \xrightarrow{0} q1 \xrightarrow{1} q2 \xrightarrow{1} q2 \xrightarrow{0} q3 \xrightarrow{0} q2 \xrightarrow{0} q3$ (no)

Deterministic Finite Automata

- Formal Definition (Deterministic Finite Automata (DFA))
  - A finite automata is a 5-tuple (Q, Σ, δ,q_0, F) where:
    1. Q is a finite set of states
    2. Σ is a finite set called the alphabet
    3. δ: Q $\times$ Σ $\rightarrow$ Q is the transition function
    4. $q_0 \in Q$ is the start state
    5. $F \subseteq Q$ is a set of accept states

Deterministic Finite Automata

- Ex: M_1



1. Q =
2. Σ =
3. q_0 =
4. F =

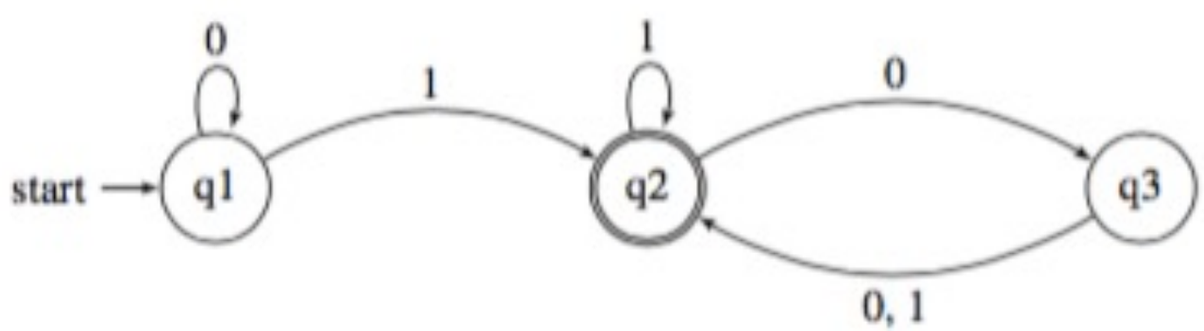
δ	**0**	**1**
q_1		
q_2		
q_3		

When at state ? With an input of ? Move to state ?

- Deterministic means that each state has exactly one transition for each symbol in the alphabet

Deterministic Finite Automata

- Ex: M_1



1. $Q = \{q_1, q_2, q_3\}$
2. $\Sigma = \{0, 1\}$
3. $q_0 = q_1$
4. $F = \{q_2\}$

δ	**0**	**1**
$\mathbf{q_1}$	q_1	q_2
$\mathbf{q_2}$	q_3	q_2
$\mathbf{q_3}$	q_2	q_2

When at state ? With an input of ? Move to state ?

Ex: State q_1 (row), input of 0 (column), move to q_1 (output in grid)

- Deterministic means that each state has <u>exactly one</u> transition <u>for each</u> symbol in the alphabet

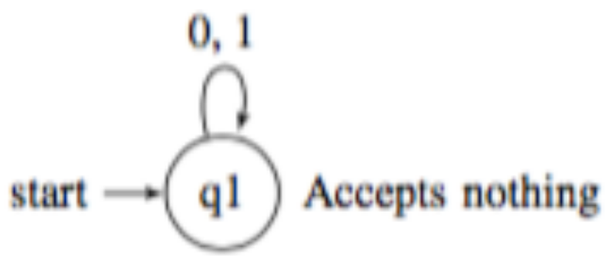
Regular Languages

- Languages are just a set of strings
- Language of Machine M:
  - If A is a set of all strings that machine M accepts, say A is the language of M (L(M) = A)
    - M recognizes A
      - Ex: A = {w | w contains at least one 1 and an even number of 0's following the last 1}
      - L(M) = A
    - Generic: L(M) = {x | x is accepted by M}
      - Build a machine M to accept L(M) or A

Regular Languages, cont.

- Language of Machine M:
  - If A is a set of all strings that machine M accepts, say A is the language of M (L(M) = A)
  - $A \subseteq \Sigma^*$ (Σ^* means all possible strings formed from symbols in the alphabet Σ including the empty string)
- Say $\Sigma = \{0, 1\}$
  - Can build a machine to accept all Σ^*

- F = ∅, then the language of the machine L(M) = ∅
  - No accept state



Design an Automata

- Ex: Design M where the language is:
  - L(M) = {x | x ∈ {0, 1, 2}* such that the sum of digits of x is divisible by 3}
    - How do we build this?

Design an Automata

- Ex: Design M where the language is:
  - L(M) = {x | x ∈ {0, 1, 2}* such that the sum of digits of x is divisible by 3}
    - How do we build this?
      - Σ = {0, 1, 2}
      - First x can be the empty set, ε.
        - The start state should be an accept state since can add the empty set to itself and it should be included.
      - Try numbers: ε + 0 = 0 (divisible by 3, so loop back to start state on 0)

Design an Automata, cont.

- Ex: Design M where the language is:
  - L(M) = {x | x ∈ {0, 1, 2}* such that the sum of digits of x is divisible by 3}
    - How do we build this?
      - Σ = {0, 1, 2}
      - Try numbers: ε + 1 = 1 (not divisible by 3, so move to a new state: q_1)
      - Try numbers: ε + 2 = 2 (not divisible by 3, so move to a new state: q_2)
      - Try numbers: 1 + 1 = 2 (not divisible by 3, but same as q_2, so move to state: q_2)
      - Try numbers: 2 + 1 = 3 (divisible by 3, so move back to start state: q_0)

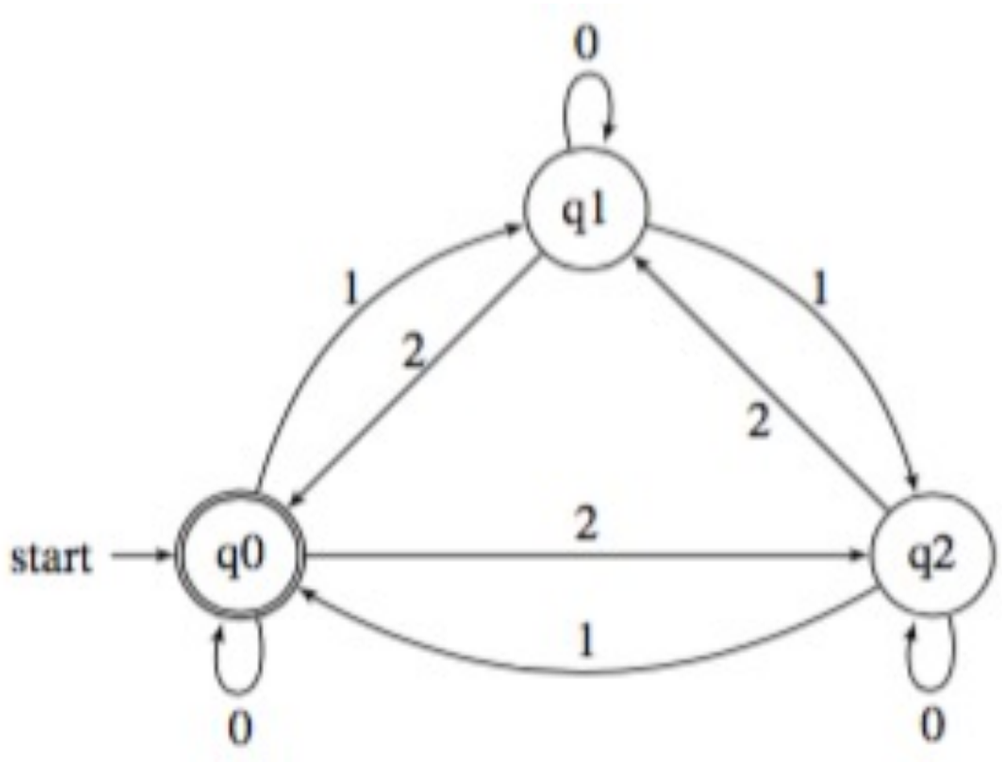
Design an Automata, cont.

- Ex: Design M where the language is:
  - L(M) = {x | x ∈ {0, 1, 2}* such that the sum of digits of x is divisible by 3}
    - How do we build this?
      - Σ = {0, 1, 2}
      - Try numbers: 1 + 1 + 1 = 3 (divisible by 3, so move back to start state: q_0)
      - Try numbers: 2 + 2 = 4 (not divisible by 3, 4 % 3 = 1, so move to state: q_1)
      - Try numbers: 1 + 2 = 3 (divisible by 3, so move to start state: q_0)
      - Adding 0 to any number does not change the value so loop back to state

Design an Automata, cont.

- Ex: Design M where the language is:
  - L(M) = {x | x ∈ {0, 1, 2}* such that the sum of digits of x is divisible by 3}
  - Q = $\{q_0, q_1, q_2\}$
  - Σ = {0, 1, 2}
  - $q_0 = q_0$
  - F = $\{q_0\}$

δ	**0**	**1**	**2**
q_0	q_0	q_1	q_2
q_1	q_1	q_2	q_0
q_2	q_2	q_0	q_1

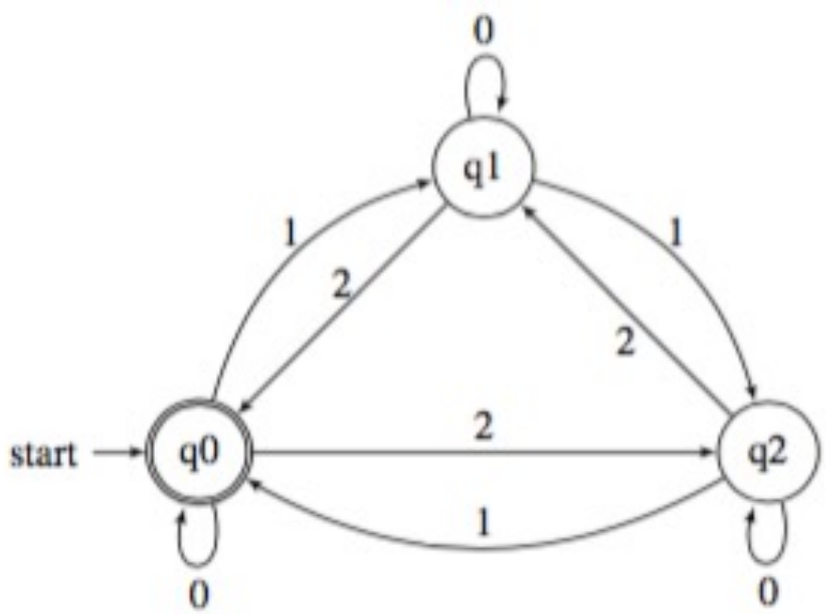


Formal Definition of Computation

- Let M = (Q, Σ, δ, q_0, F) be a finite automata (DFA)
  - Let w = $w_1 w_2 \ldots w_n \in \Sigma^*$ be an input string
  - Then M accepts w if a sequence of states $r_0, r_1, \ldots, r_n$ in Q exists with 3 conditions:
    1. r_0 is the start state q_0 ($r_0 = q_0$)
    2. For $i = 0, \ldots, n-1$, $\delta(r_i, w_{i+1}) = r_{i+1}$
    3. $r_n \in F$ (r_n is an accepting state)
  - Say M recognizes language A if A = {w | M accepts w}
- A language is called a regular language if some finite automaton recognizes it.

Formal Definition of Computation, cont.

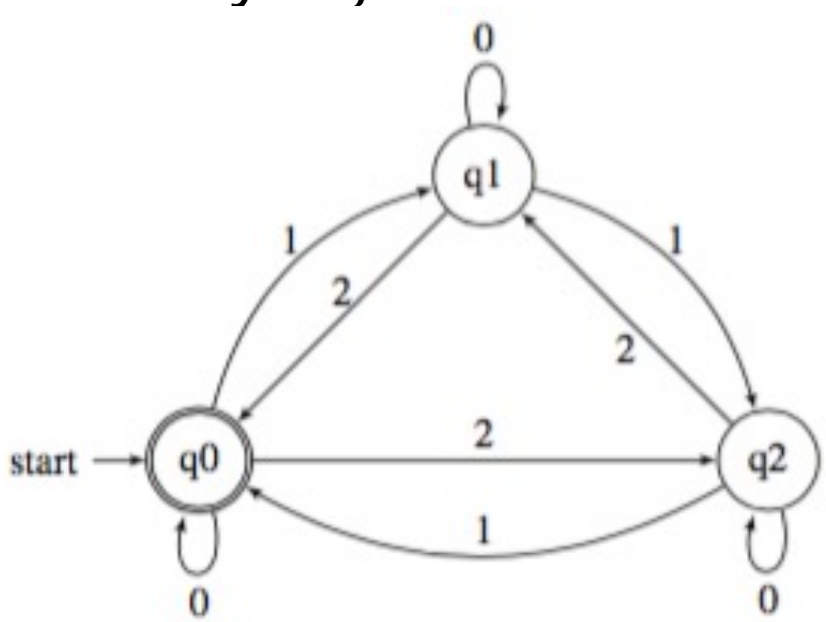
- Ex: Formally show M where language L(M) = {x | x ∈ {0, 1, 2}* where sum is divisible by 3}



- If w = 1022010 will the machine accept it?
  - States r =

Formal Definition of Computation, cont.

- Ex: Formally show M where language L(M) = {x | x ∈ {0, 1, 2}* where sum is divisible by 3}



- If w = 1022010 will the machine accept it?
  - States r = $q_0 \xrightarrow{1} q_1 \xrightarrow{0} q_1 \xrightarrow{2} q_0 \xrightarrow{2} q_2 \xrightarrow{0} q_2 \xrightarrow{1} q_0 \xrightarrow{0} q_0$ (ends at an accept state, so Yes!)

Designing Finite Automata

- Ex: L(M) = {x ∈ {0, 1}* | x has odd number of 1's}
  - How do we design this?

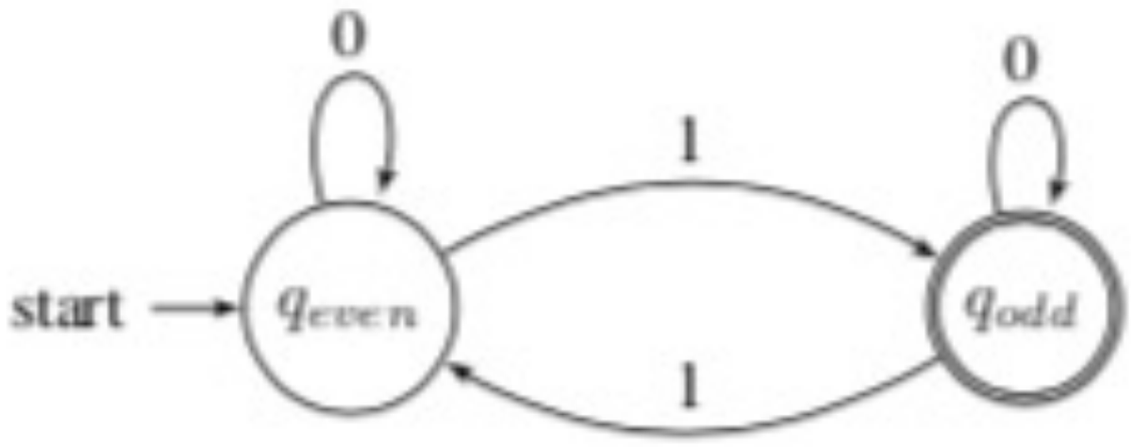
Designing Finite Automata

- Ex: L(M) = {x ∈ {0, 1}* | x has odd number of 1's}
  - How do we design this?
    - Have to have at least one 1, so start state cannot be accept state, but can make the second state the accept state.
    - Can go back to start state if second 1 is given
    - If another 1 is given can go to the second state, and can continue to go back and forth
    - What about 0's? You should stay at the same state when given a 0 since it does not change the number of 1's.

Designing Finite Automata, cont.

- Ex: L(M) = {x ∈ {0, 1}* | x has odd number of 1's}
  - How do we design this?
    - Two states: Q = {q_{even}, q_{odd}}
    - Alphabet: Σ = {0, 1}
    - Start state: q_0 = q_{even}
    - Final states: F = {q_{odd}}
    - Transition function:

δ	**0**	**1**
q_{even}	q_{even}	q_{odd}
q_{odd}	q_{odd}	q_{even}



Designing Finite Automata, cont.

- Ex: L(M) = $\{x \in \{0, 1\}^* \mid x \text{ contains the substring } 001\}$
  - How do we design this?

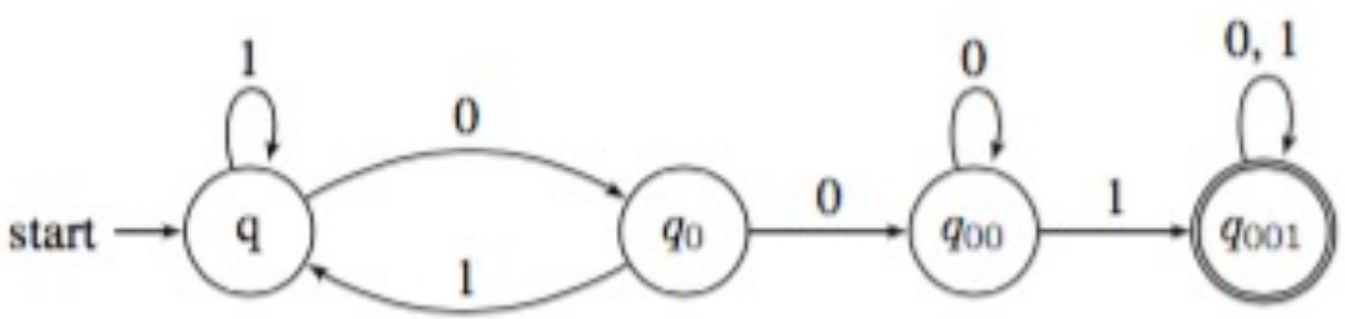
Designing Finite Automata, cont.

- Ex: L(M) = {x ∈ {0, 1}* | x contains the substring 001}
  - How do we design this?
    - Have to have 001 in a sequence.
    - Can start on any input, but only a 0 should move us towards the accept state
    - If enter a 1 after one 0, need to move back to the start, since that is not part of the sequence
    - A second 0 moves us to another state.
    - At this point a 1 takes us to the accept state, another 0 should keep us where we are since it does not change the sequence
    - Any more 0's or 1's do not matter and should keep us where we are

Designing Finite Automata, cont.

- Ex: L(M) = {x ∈ {0, 1}* | x contains the substring 001}
  - How do we design this?
    - Two states: Q = {q, q_0, q_{00}, q_{001}}
    - Alphabet: Σ = {0, 1}
    - Start state: q_0 = q
    - Final states: F = {q_{001}}
    - Transition function:

δ	**0**	**1**
q	q_0	q
q_0	q_{00}	q
q_{00}	q_{00}	q_{001}
q_{001}	q_{001}	q_{001}



DFA Review

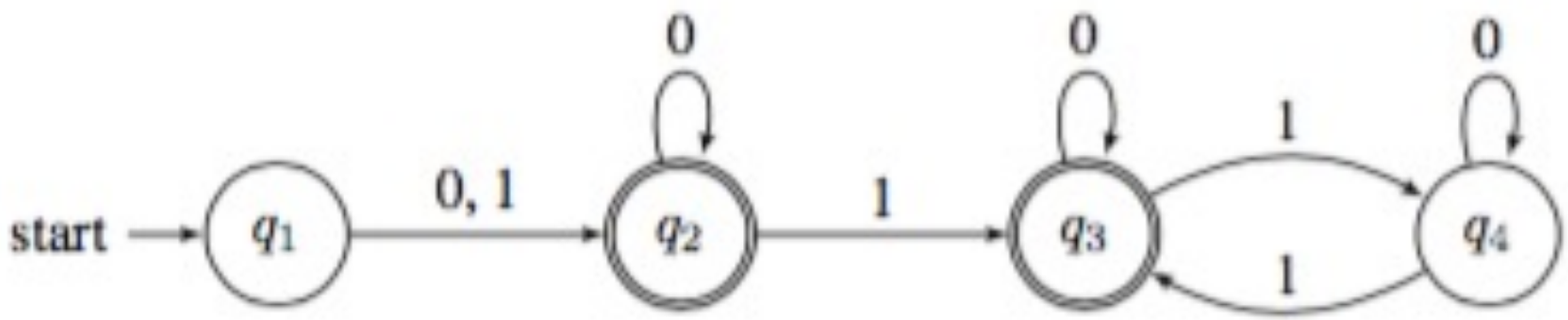
- A finite automata is a 5-tuple (Q, Σ, δ,q_0, F) where:
  1. Q is a finite set of states
  2. Σ is a finite set called the alphabet
  3. δ: Q × Σ → Q is the transition function
  4. $q_0 \in Q$ is the start state
  5. $F \subseteq Q$ is a set of accept states
- Deterministic means that <u>each</u> state has <u>exactly one</u> transition <u>for each</u> symbol in the alphabet

Review, cont.

- Language of Machine M (the DFA):
  - If A is a set of all strings that machine M accepts, say A is the language of M (L(M) = A)
    - M recognizes A
    - L(M) = {x | x is accepted by M}
      - Build a machine M to accept L(M) or A
    - $A \subseteq \Sigma^*$ (Σ^* means all possible strings formed from symbols in the alphabet Σ)

Review, cont.

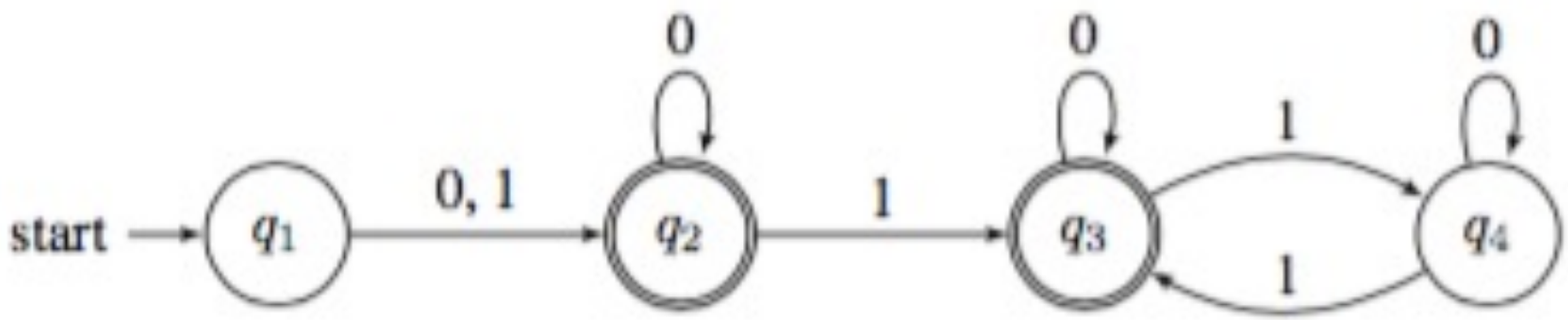
- Formally describe the language of this DFA:



- Will this DFA accept string 00111?
- Give the formal description of this machine.

Review, cont.

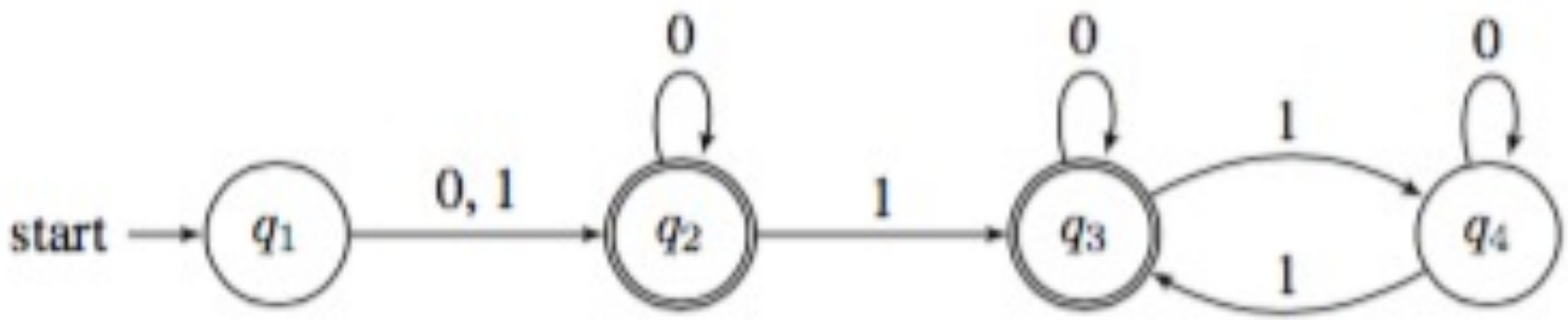
- Formally describe the language of this DFA:



- Will this DFA accept string 00111?
  - Ans: $q_1 \xrightarrow{0} q_2 \xrightarrow{0} q_2 \xrightarrow{1} q_3 \xrightarrow{1} q_4 \xrightarrow{1} q_3$ (accept state, so Yes!

Review, cont.

- Formally describe the language of this DFA:



- Give the formal description of this machine.
  - $Q = \{q_1, q_2, q_3, q_4\}$, $\Sigma = \{0, 1\}$, $q_0 = q_1$, $F = \{q_2, q_3\}$

δ	**0**	**1**
q_1	q_2	q_2
q_2	q_2	q_3
q_3	q_3	q_4
q_4	q_4	q_3

Review, cont.

- Given a DFA M = ($\{q_1, q_2, q_3, q_4, q_5\}$, $\{0, 1\}$, δ, q_1, $\{q_3, q_4, q_5\}$) where δ is

δ	**0**	**1**
$\mathbf{q_1}$	q_2	q_2
$\mathbf{q_2}$	q_3	q_1
$\mathbf{q_3}$	q_2	q_4
$\mathbf{q_4}$	q_5	q_4
$\mathbf{q_5}$	q_3	q_5

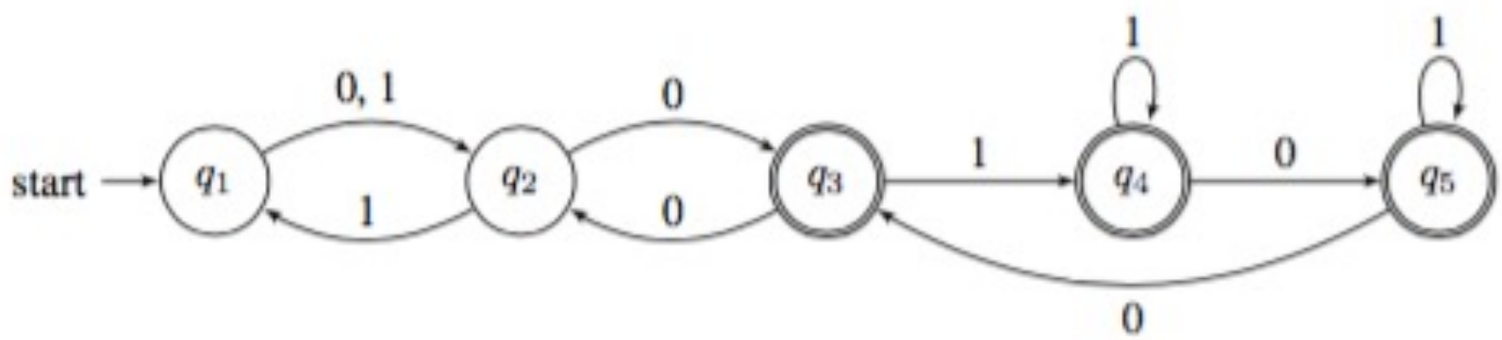
- Draw the state diagram

Review, cont.

- Given a DFA M = ({q_1, q_2, q_3, q_4, q_5}, {0, 1}, δ, q_1, {q_3, q_4, q_5}) where δ is

δ	**0**	**1**
q_1	q_2	q_2
q_2	q_3	q_1
q_3	q_2	q_4
q_4	q_5	q_4
q_5	q_3	q_5

- State diagram:

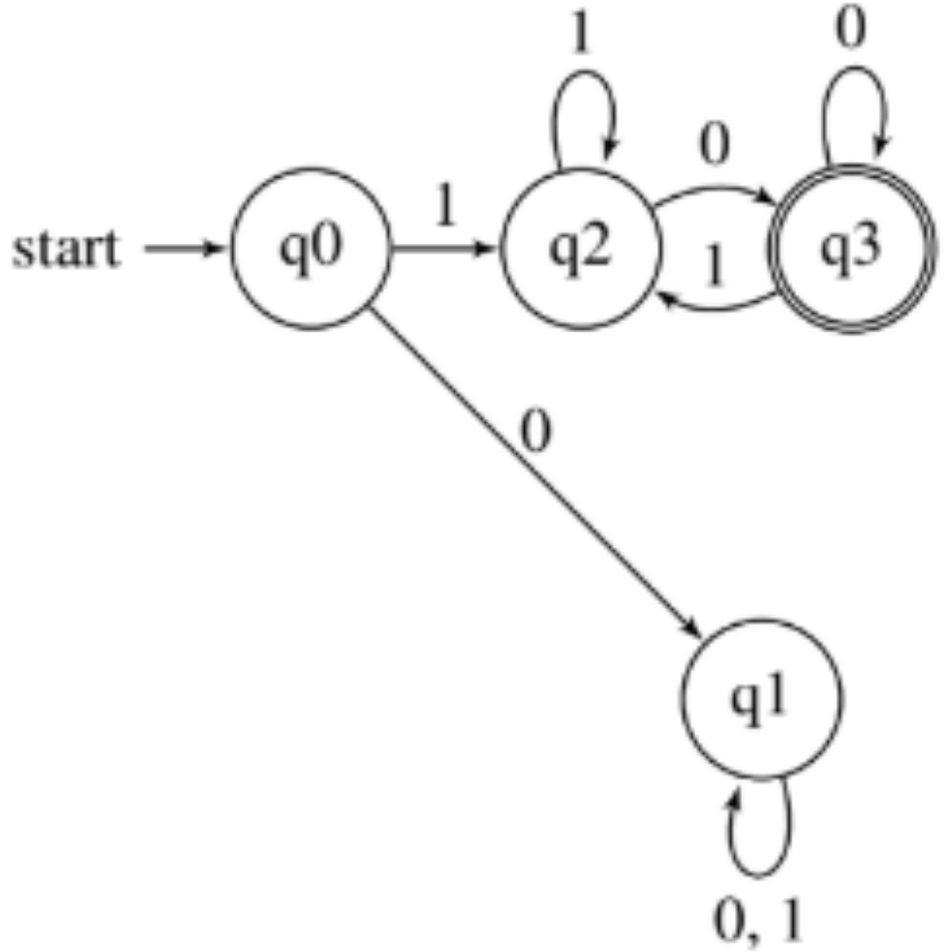


Try It

- Design a finite automata where L(M) = {x ∈ {0, 1}* | x begins with a 1 and ends with a 0}

Try It

- Design a finite automata where L(M) = {x ∈ {0, 1}* | x begins with a 1 and ends with a 0}



Theory of Computation
Chapter 1

Regular Languages Part 1

School of Engineering | Computer Science

Claude Shannon
1916-2001

- Wrote “the most important Master’s thesis of all time”, proposing that a machine based on Boolean logic could solve problems.
- Later, founder of “information theory” – theory of how to compress, encode and store information.
- Named the “bit”

Regular Languages

- A is a regular language if there is a finite automaton, DFA, M, that recognizes it L(M) = A
- Regular languages are the set of languages that are recognizable by deterministic finite automata

Closure of Regular Languages

- Given two regular languages A and B, which operations applied to A and B produce a new regular language C?
- There are three regular operations:
  - Union: $A \cup B = \{ x \mid x \in A \text{ or } x \in B\}$
  - Concatenation: $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$
  - Star: $A^* = \{x_1x_2\ldots x_k \mid k \geq 0 \text{ and } x_i \in A \text{ for all } 1 \leq i \leq k\}$
    ($k = 0$ is allowed → empty string, ε, which is always in A^*)
- Regular operations are helpful and allow us to break up complex languages into smaller, simpler pieces and to combine them using regular operations

Closure of Regular Languages, cont.

- There are three regular operations:
  - Union: $A \cup B = \{ x \mid x \in A \text{ or } x \in B\}$
  - Concatenation: $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$
  - Star: $A^* = \{x_1x_2\ldots x_k \mid k \geq 0 \text{ and } x_i \in A \text{ for all } 1 \leq i \leq k\}$ ($k = 0$ is allowed → empty string, ε, which is always in A^*)
- Ex: $S = \{a, b\}$, $T = \{0, 1\}$, $\Sigma = \{0, 1, a, b\}$
  - $S \cup T =$
  - $S \circ T =$
  - $S^* =$

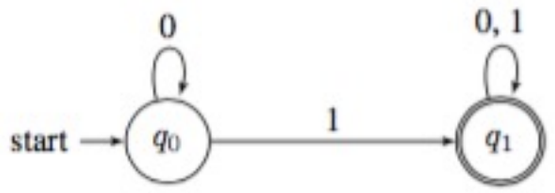
Closure of Regular Languages, cont.

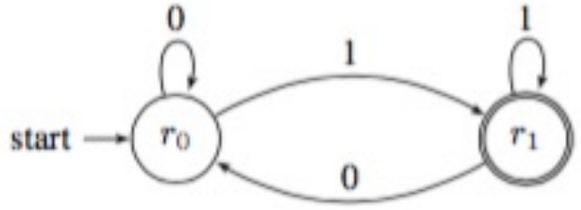
- There are three regular operations:
  - Union: $A \cup B = \{ x \mid x \in A \text{ or } x \in B\}$
  - Concatenation: $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$
  - Star: $A^* = \{x_1x_2\ldots x_k \mid k \geq 0 \text{ and } x_i \in A \text{ for all } 1 \leq i \leq k\}$ ($k = 0$ is allowed → empty string, ε which is always in A^*)
- Ex: $S = \{a, b\}$, $T = \{0, 1\}$, $\Sigma = \{0, 1, a, b\}$
  - $S \cup T = \{a, b, 0, 1\}$
  - $S \circ T = \{a0, a1, b0, b1\}$
  - $S^* = \{a, b\}^* = \{\varepsilon, a, b, aa, bb, ba, bb, aaa, \ldots\}$

Closure Definition

- Closure:
  - Are regular languages closed under their operations?
  - What is closure?
    - If a class of languages is closed under an operation, then, when that operation is performed on languages in that class, the result is always a language within that class of languages.
      - Ex: Adding two integer numbers, the result is an integer value → integers are closed under addition
      - Ex: Dividing two integer numbers, the result is not always an integer → integers are not closed under division

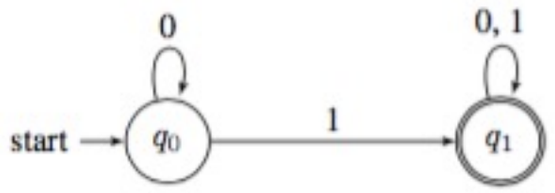
Closure of Regular Languages Definition

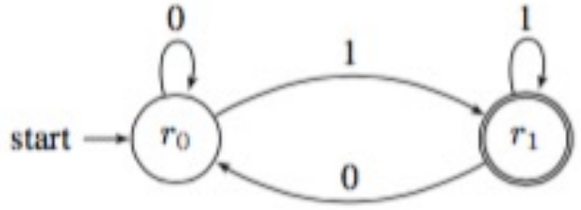
- Are regular languages closed under the union operation?
  - Proof Ex: M_1: 

M_2: 

  - M_3 recognizes $L(M_1) \cup L(M_2)$:
    - Combine the states in every combination
    - Show the transitions simultaneously

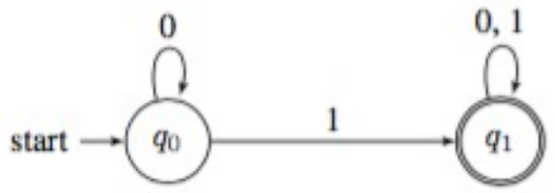
Closure of Regular Languages Definition

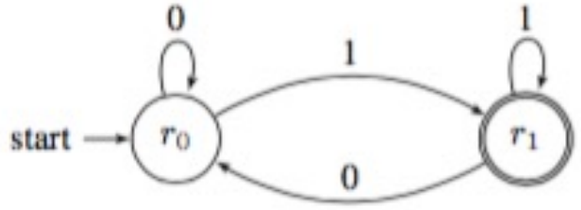
- Are regular languages closed under the union operation?
  - Proof Ex: M_1: 

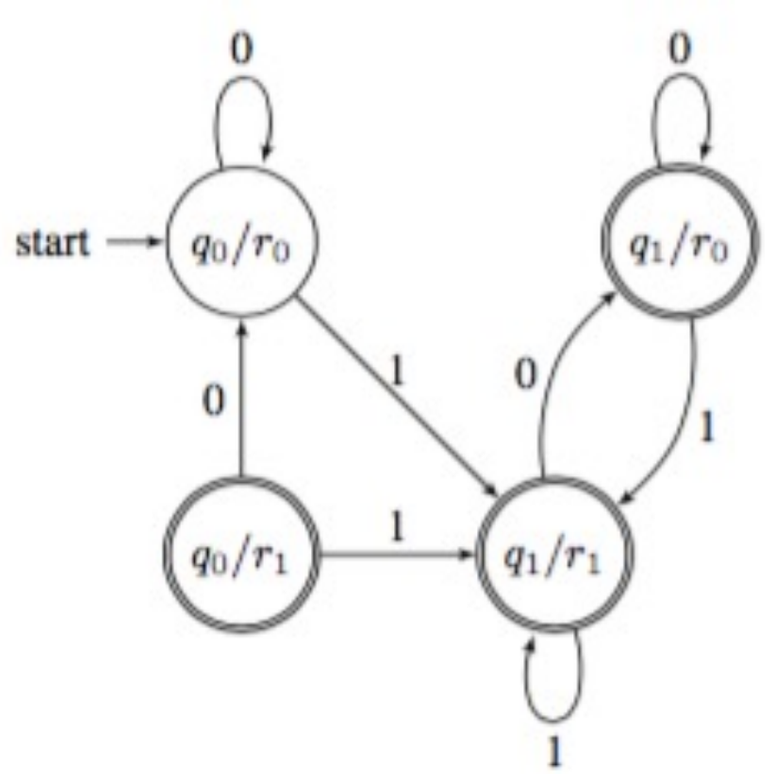
M_2: 

  - M_3 recognizes $L(M_1) \cup L(M_2)$:

Closure of Regular Languages Definition

- Are regular languages closed under the union operation?
  - Proof Ex: M_1: 

M_2: 

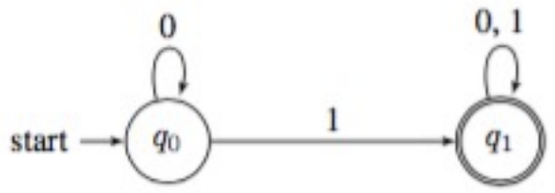
  - M_3 recognizes $L(M_1) \cup L(M_2)$: 

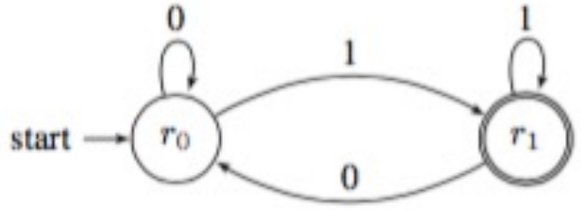
Closure of Regular Languages Definition, cont.

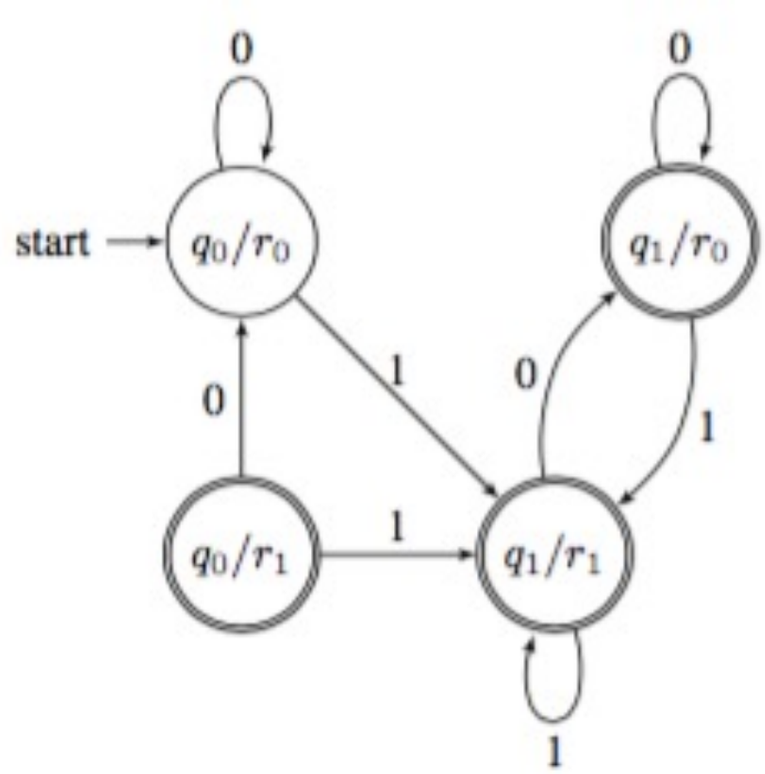
- Are regular languages closed under the union operation?
  - Put the idea into words:
    - Let Q_1 and Q_2 be sets of states for the finite automata M_1 and M_2 recognizing the languages A and B, respectively.

1. States for $M_1 \cup M_2$ are $Q_1 \times Q_2$
2. Simultaneously track the transitions for both M_1 and M_2
3. Accept only if string ends in state (q_i, r_j) such that either q_i or r_j is an accepting state in M_1 or M_2 respectively

Closure of Regular Languages Definition

- Are regular languages closed under the union operation?
  - Proof Ex: M_1: 

M_2: 

  - M_3 recognizes $L(M_1) \cup L(M_2)$: 

Closure of Regular Languages Definition, cont.

- Formal Proof that regular languages are closed under the union operation
    - Let M_1 recognize A_1 where $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and M_2 recognize A_2 where $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$, construct $M_3 = (Q_3, \Sigma, \delta_3, q_3, F_3)$ such that $L(M_3) = A_1 \cup A_2$ (M_3 recognizes $A_1 \cup A_2$)

1. $Q_3 = Q_1 \times Q_2 = \{(r_1, r_2) \mid r_1 \in Q_1, r_2 \in Q_2\}$
2. Σ is the same
3. For all $(r_1, r_2) \in Q_3$, and all $a \in \Sigma$, let $\delta_3((r_1, r_2), a) = (\delta_1(r_1, a), \delta_2(r_2, a))$
4. $q_3 = (q_1, q_2)$
5. $F_3 = \{(r_1, r_2) \mid r_1 \in F_1 \text{ or } r_2 \in F_2\}$ (same as $F_3 = (F_1 \times Q_2) \cup (Q_1 \times F_2)$)

Summary Closure of Regular Languages

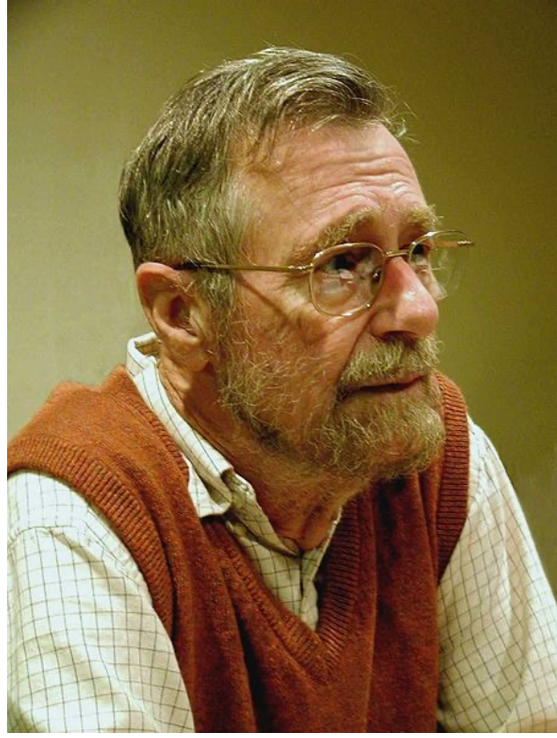
- We just showed that we can formally prove that regular languages are closed under the union operation.
- The same is true for applying the concatenation and star operations, but the proofs will be easier once we study non-deterministic finite automata (NFAs).
- We will learn about NFAs soon.

Theory of Computation
Chapter 1

Non-Deterministic Finite Automata

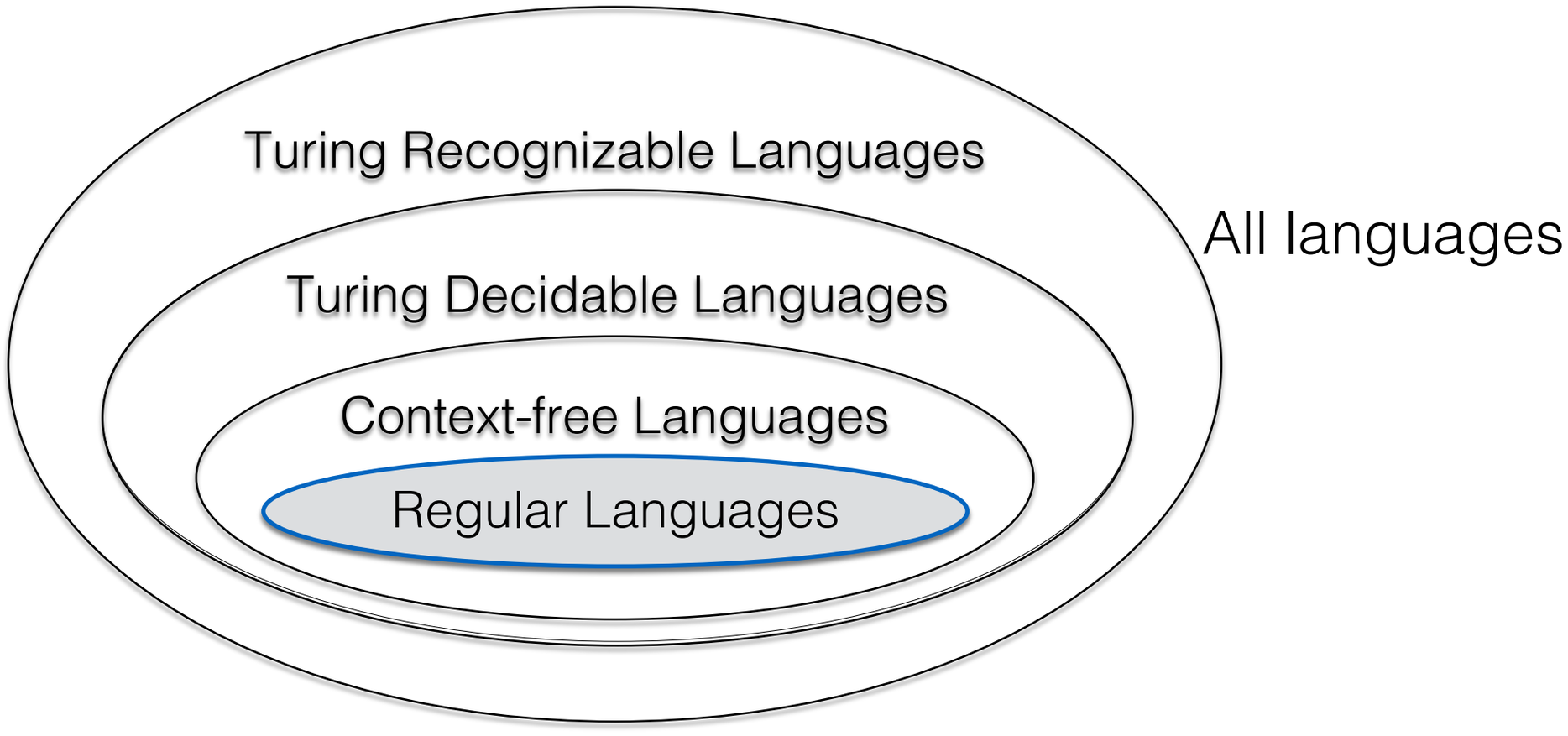
School of Engineering | Computer Science

Edsger Dijkstra 1930-2002



- Developed:
  - Shortest path algorithm (Dijkstra's algorithm)
  - Concept of a semaphore, Reverse Polish notation, the THE multiprogramming system, Banker's algorithm, CS sub-field of self stabilization….
  - Wrote scathing commentary on (once) popular GOTO programming statement
  - 1972 Turing award for programming language contributions
  - "two or or more, use a for"
  - "Computer science is no more about computers than astronomy is about telescopes."

Regular Languages



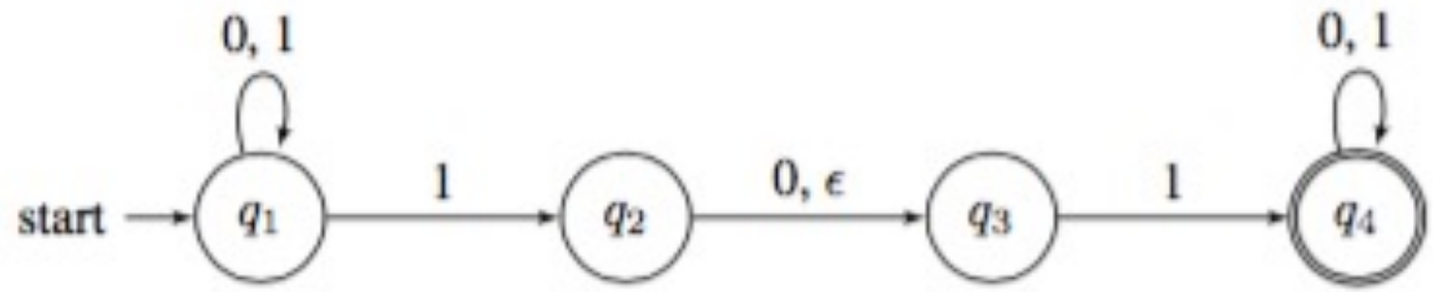
Regular Languages
DFA = NFA = RE
Closed under union, $\cup$, concatenation, $\circ$, and star, $*$.

Non-Deterministic Finite Automata (NFA)

- NFA = DFA where at each step you can follow multiple choices simultaneously
- Remember that a DFA has exactly one transition for each symbol at each state
  - This does not apply to the NFA
- The differences between an NFA and a DFA
  1. NFA's can label transitions with ε,the empty string
  2. Each symbol $a \in \Sigma$ can appear in 0, 1, or many transitions from a given state
     - What happens if the symbol a appears 0 times at a state?
       - Ans: There is no transition for a and the string is not accepted

Non-Deterministic Finite Automata, cont.

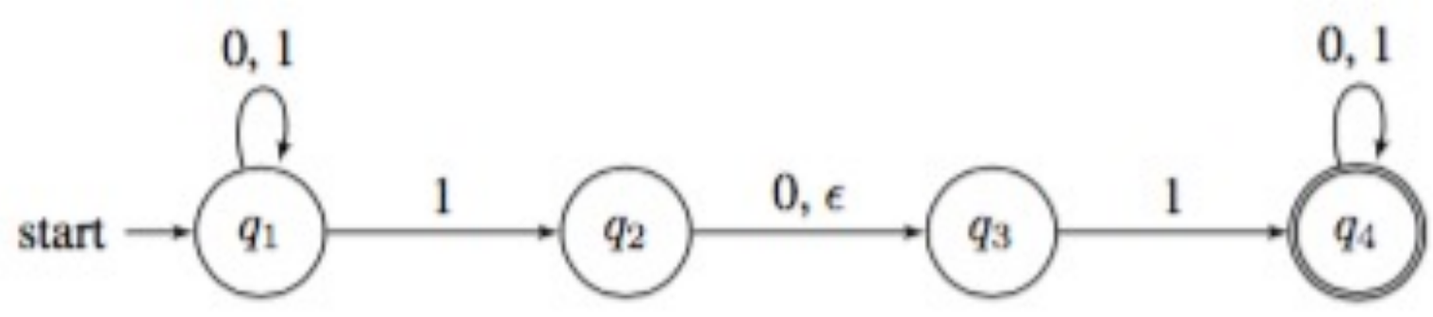
- Key Idea: An NFA will accept a string if there is any computational branch that leads to an accept state
- Ex: $L(N_1) = \{x \mid x \text{ contains 11 or 101 as a substring}\}$
- NFA N_1:



- What do you notice at q_1?
- NFA accepts if any branch on the computational tree for an input string ends in an accept state.

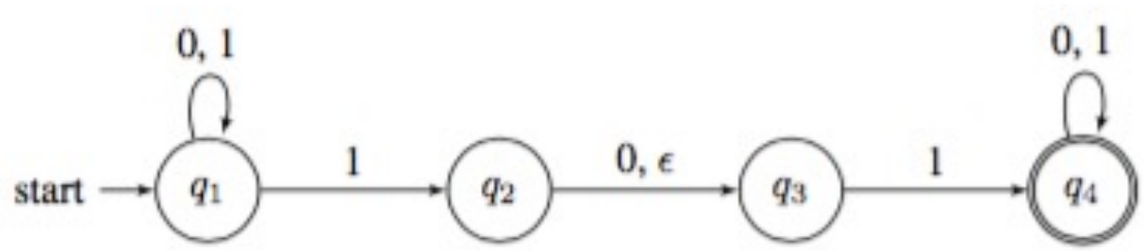
Non-Deterministic Finite Automata, cont.

- Ex: $L(N_1)$ = {x | x contains 11 or 101 as a substring}
  - NFA N_1:



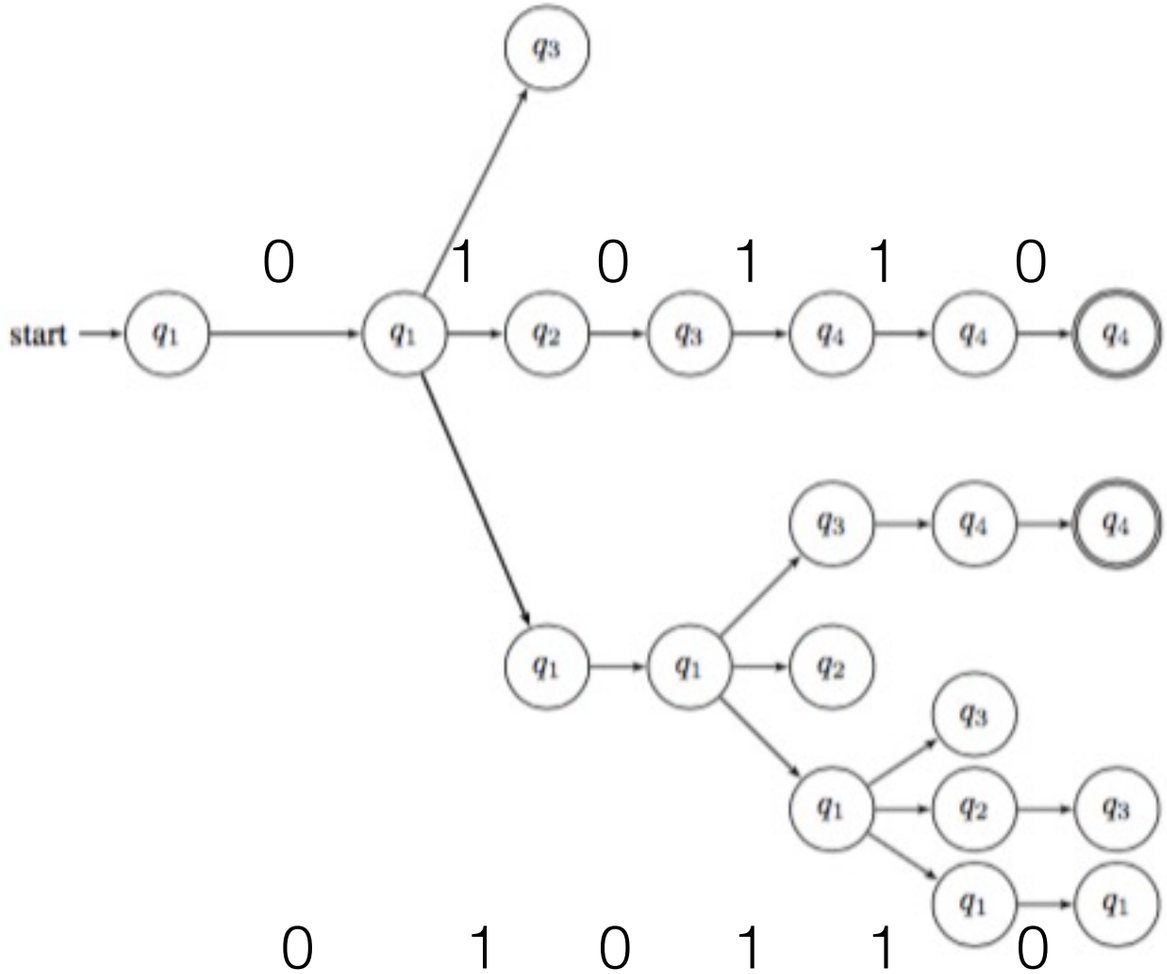
- Creating the Computational Tree for string x = 010110 to determine if the string is accepted:
  - Follow each of the possible paths that the NFA can take for each symbol from the string x. Because it is an NFA, there is more than one possible path.
  - Like a maze where you hope that one path will lead to an accept state

Non-Deterministic Finite Automata, cont.



These are all the possible ways 010110 can be derived.

- Computational Tree for string x = 010110:

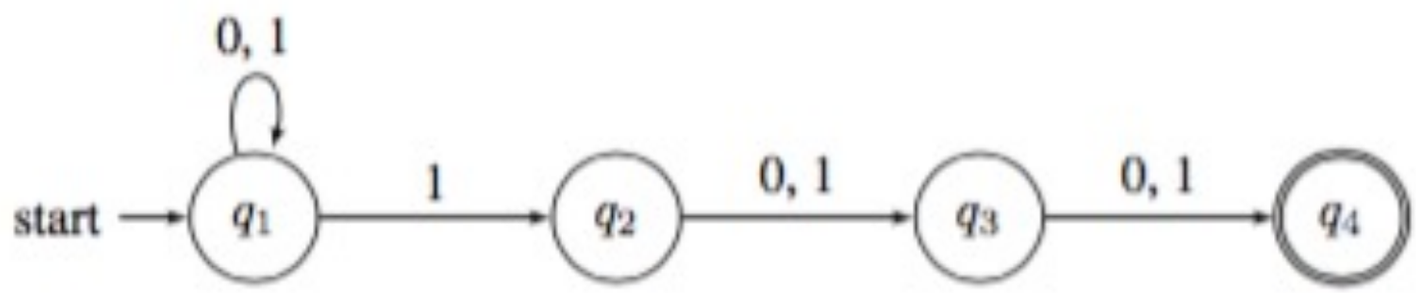


Note that the ε transition can move to the next state without reading a symbol from the string as in q_2 to q_3, which makes it look like q_1 goes directly to q_3.

The string is accepted since at least one branch ends at an accept state.

Designing NFA's

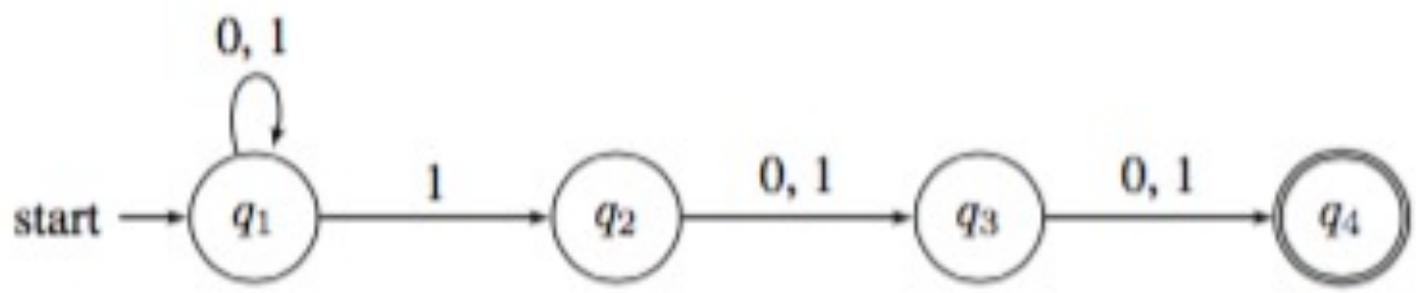
- Ex: NFA N_2:



- What is the language that this NFA recognizes for strings with $\Sigma = \{0,1\}$?

Designing NFA's

- Ex: NFA N_2:



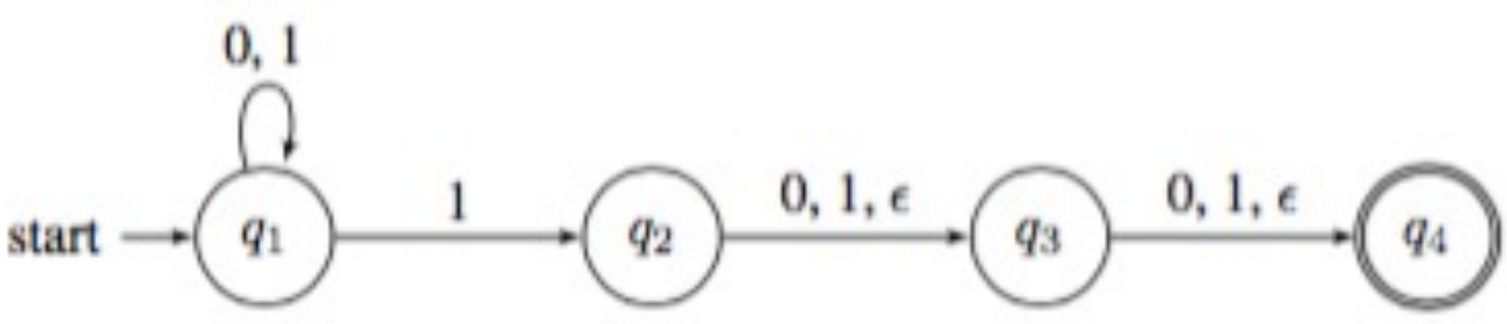
  - What is the language that this NFA recognizes for strings with $\Sigma = \{0,1\}$?
    - $L(N_2) = \{x \mid x \in \Sigma^* 1 \Sigma^2\}$ (all strings where there is a 1 in the 3rd to last position from the end)

Designing NFA's

- Ex: $L(N_3)$ = {x | x has a 1 in at least one of the last three positions}
  - What is the NFA?

Designing NFA's

- Ex: $L(N_3) = \{x \mid x$ has a 1 in at least one of the last three positions$\}$
  - What is the NFA?
  - NFA N_3:



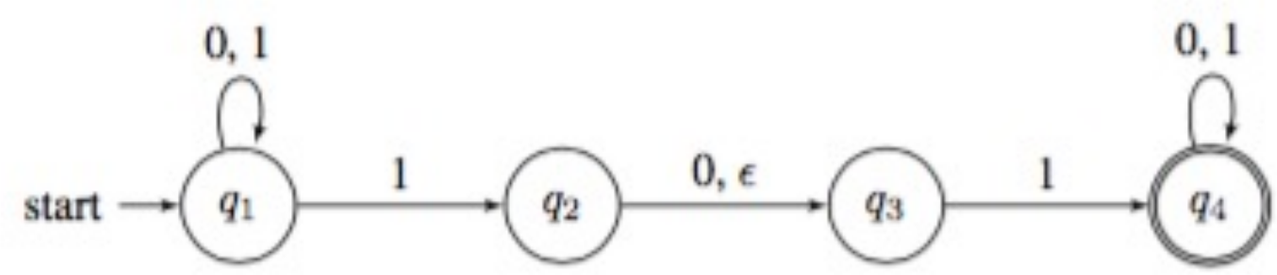
  - $L(N_3) = \{x \mid x \in \Sigma^* 1\ \Sigma^2 \cup \Sigma^* 1\ \Sigma^1 \cup \Sigma^* 1\}$

Definition of NFA

- Formal Definition of NFA:
  - A NFA is a 5-tuple (Q, Σ, δ,q_0, F) where:
    1. Q is a finite set of states
    2. Σ is a finite set called the alphabet
    3. δ: $Q \times \Sigma_\varepsilon \rightarrow P(Q)$ is the transition function
       - $\Sigma_\varepsilon = \Sigma \cup \varepsilon$
       - P(Q) is the power set of Q (collection of all subsets of Q including the empty string)
    4. $q_0 \in Q$ is the start state
    5. $F \subseteq Q$ is a set of accept states

Definition of NFA, cont.

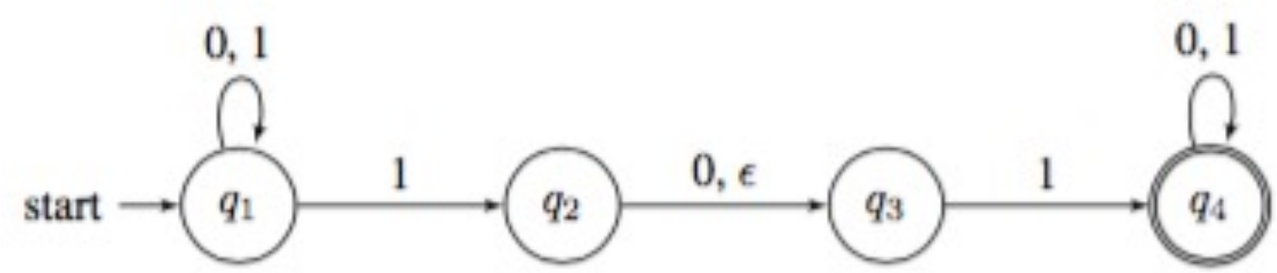
- Ex: N_1:



- Give the formal definition of this NFA

Definition of NFA, cont.

- Ex: N_1:



  - Give the formal definition of this NFA
    - $Q = \{q_1, q_2, q_3, q_4\}$
    - $\Sigma = \{0, 1\}$
    - $\Sigma_\varepsilon = \{0, 1, \varepsilon\}$
    - $q_0 = q_1$
    - $F = \{q_4\}$

δ	0	1	ε
q_1	$\{q_1\}$	$\{q_1, q_2\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$

Formal Definition of Computation

- Let N = (Q, Σ, δ, q_0, F) be a non-deterministic finite automata (NFA) and w $\in \Sigma^*$ (a string)
  - N <u>accepts</u> w if we can write w = $y_1y_2\ldots y_m$ for $y_i \in \Sigma_\varepsilon$ for all $1 \leq i \leq m$, and there exists a sequence of states (r_0, r_1,..., r_m) $\in Q_{m+1}$ with 3 conditions:
    1. r_0 is the start state q_0 ($r_0 = q_0$)
    2. $r_{i+1} \in \delta(r_i, y_{i+1})$ for i $\in$ {0, …, m-1}
    3. $r_m \in F$ (r_m is an accepting state)
- Do NFA's = DFA's?
  - Yes. For all languages L, there exists a NFA recognizing L if and only if there exists a DFA recognizing L.

Equivalence of NFAs and DFAs

- Theorem 1.39: Every NFA has an equivalent DFA
  - Proof: Let L be a language recognized by a NFA N = (Q, Σ, δ,q_0, F), we construct a DFA M = (Q′ , Σ, δ' ,q_0' , F′) recognizing L
    - Idea: At each point in computation, M keeps track of multiple states
      - Q′ = P(Q)
      - (P(Q) = set of subsets of Q)
      - $|Q'| = 2^{|Q|}$

Equivalence of NFAs and DFAs

- Formal Proof Every NFA has an equivalent DFA
  - Assume for now that the NFA N has no ε transitions:

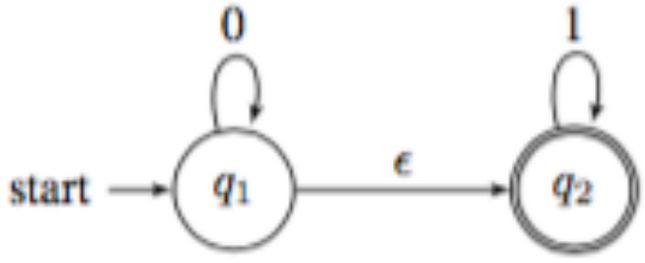
1. $Q' = P(Q)$
2. $\Sigma = \Sigma$
3. For $R \in Q'$ and $a \in \Sigma$, let $\delta'(R, a) = \{q \in Q \mid q \in \delta(r, a)$ for some $r \in R\}$
   - R is a state of M, the DFA, and is a set of states of N, the NFA
   - $\delta'(R, a) = \bigcup_{r \in R} \delta(r, a)$
4. $q_0' = \{q_0\}$
5. $F' = \{R \in Q' \mid R$ contains an accept state of N$\}$

Equivalence of NFA's and DFA's

- Formal Proof Every NFA has an equivalent DFA
  - Now add in ε transitions:
    - For any state R of M, define E(R) = {q | q can be reached from R by traveling ε transitions}
    - Modify $\delta'(R, a) = \{q \in E(\delta(r, a)) \text{ for some } r \in R\}$
    - Need to modify q_0' to be $E(\{q_0\})$ so can handle ε transitions from q_0

Equivalence of NFA's and DFA's

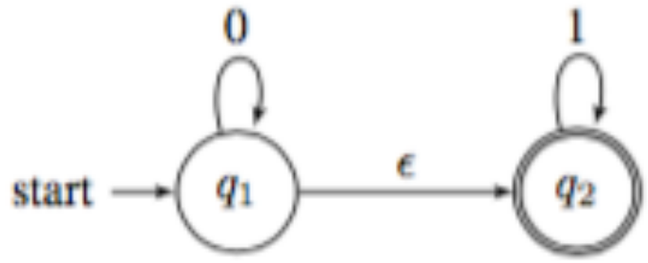
- Ex: NFA → DFA
  - NFA N
    - Q = $\{q_1, q_2\}$
    - $\Sigma = \{0, 1\}$
    - $q_0 = q_1$
    - $F = \{q_2\}$



δ	0	1	ε
$\{q_1\}$	$\{q_1\}$	$\emptyset$	$\{q_2\}$
$\{q_2\}$	$\emptyset$	$\{q_2\}$	$\emptyset$

Equivalence of NFA's and DFA's

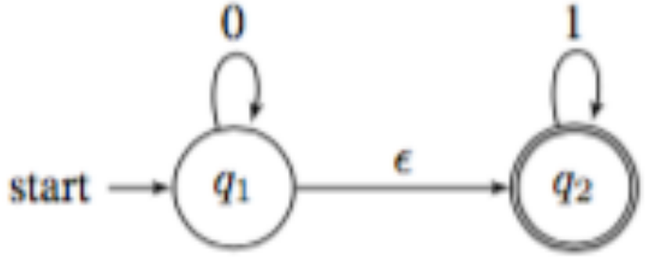
- Ex: NFA → DFA
  - NFA N:



  - 2 states, so $2^Q = 2^2 = 4$
  - DFA M:
    - Q' = power set which includes the empty set and any combination of states from Q
      - $Q' = \{\emptyset, q_1, q_2, q_{12}\}$
    - $\Sigma = \{0, 1\}$ (the alphabet does not change)
    - $q_0' = E(\{q_0\})$
      - q_0' would equal q_1 but there is an ε transition from q_1 to q_2,
      - $q_0' = q_{12}$ since can slide to state q_2 at the start or stay at state q_1
    - F' = any state containing an original accept state
      - $F' = \{q_2, q_{12}\}$

Equivalence of NFA's and DFA's

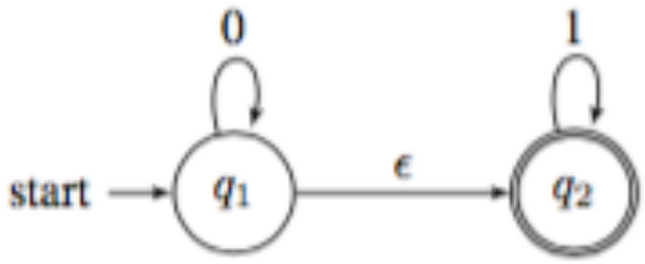
- Ex: NFA → DFA
  - NFA N:



  - DFA M:
    - $Q' = \{\emptyset, q_1, q_2, q_{12}\}$ $\Sigma = \{0, 1\}$ $q_0' = q_{12}$ $F' = \{q_2, q_{12}\}$
    - δ transitions:

Equivalence of NFA's and DFA's

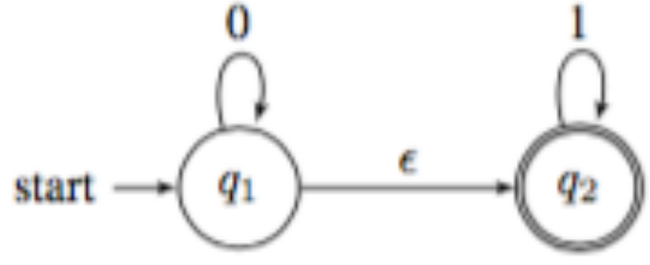
- Ex: NFA → DFA
  - NFA N:



  - DFA M:
    - $Q' = \{\emptyset, q_1, q_2, q_{12}\}$ $\Sigma = \{0, 1\}$ $q_0' = q_{12}$ $F' = \{q_2, q_{12}\}$
    - δ transitions:
      - At state q_1 with a 0 input, can stay at q_1 or slide to q_2 (union of both = q_{12})
      - At state q_1 with a 1 input, there is no transition, so go to $\emptyset$
      - At state q_2 with a 0 input, there is no transition, so go to $\emptyset$
      - At state q_2 with a 1 input, stay at q_2

Equivalence of NFA's and DFA's

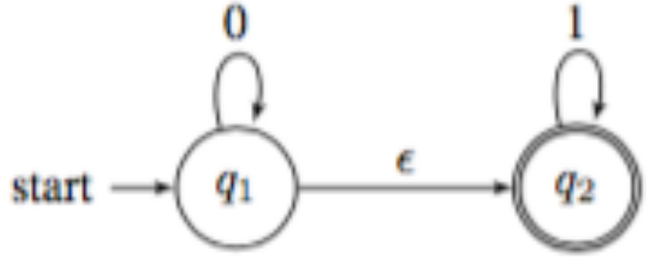
- Ex: NFA → DFA
  - NFA N:



  - DFA M:
    - $Q' = \{\emptyset, q_1, q_2, q_{12}\}$ $\Sigma = \{0, 1\}$ $q_0' = q_{12}$ $F' = \{q_2, q_{12}\}$
    - δ transitions:
      - At state $\emptyset$ with an input of 0 or 1 stay at state $\emptyset$
      - At state q_{12} (look at state q_1 and state q_2) with an input of 1, q_1 goes to $\emptyset$ and q_2 stays at q_2 (union of both = q_2)
      - At state q_{12}, (look at state q_1 and state q_2) with an input of 0, q_1 stays at q_1 or could slide to q_2 and q_2 goes to $\emptyset$ (union of both = q_{12})

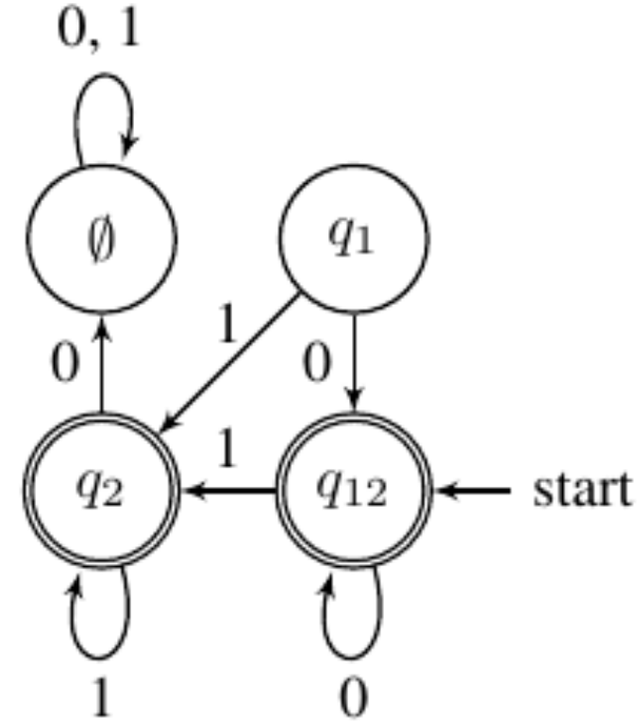
Equivalence of NFA's and DFA's

- Ex: NFA → DFA, cont.
  - NFA N:



  - 2 states, so $2^2 = 4$
  - DFA M
    - $Q' = \{\emptyset, q_1, q_2, q_{12}\}$
    - $\Sigma = \{0, 1\}$
    - $q_0 = q_{12}$
    - $F' = \{q_2, q_{12}\}$

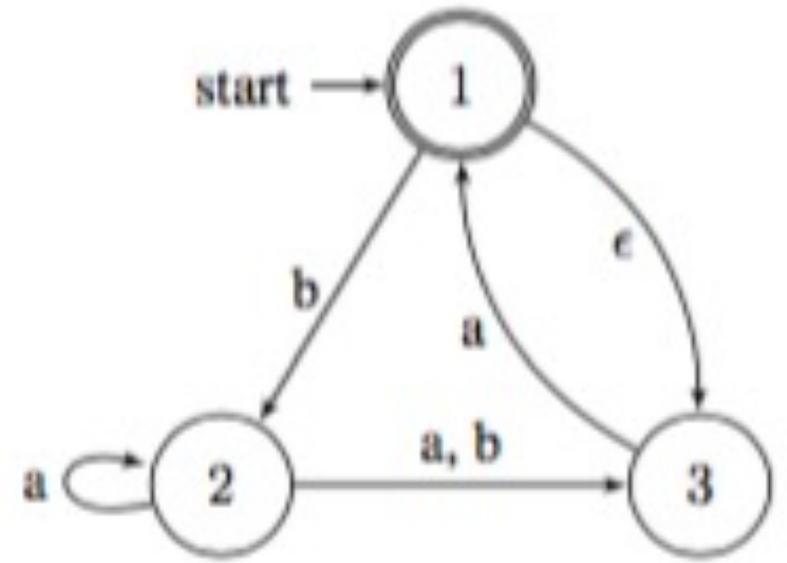
δ'	0	1
q_1	q_{12}	$\emptyset$
q_2	$\emptyset$	q_2
q_{12}	q_{12}	q_2
$\emptyset$	$\emptyset$	$\emptyset$



- Thus, DFA's and NFA's are equivalent in power

Equivalence of NFA's and DFA's

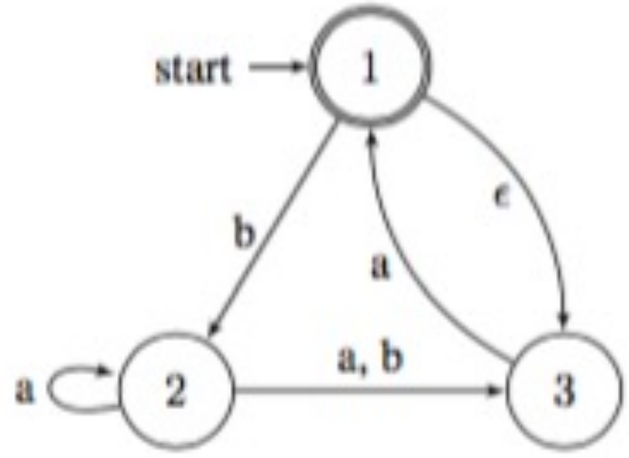
- Ex 2: NFA → DFA
  - NFA N:
    - Q = {1, 2, 3}
    - Σ = {a, b}
    - q_0 = 1
    - F = {1}



δ	a	b	ε
1	∅	{2}	{3}
2	{2,3}	{3}	∅
3	{1,3}	∅	∅

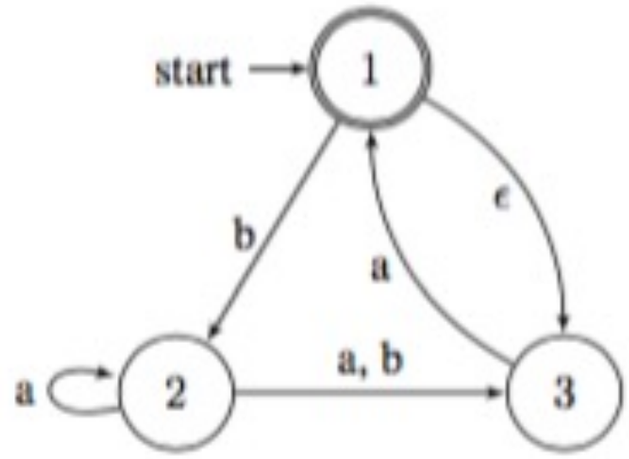
Equivalence of NFA's and DFA's

- Ex 2: NFA → DFA
  - NFA N:
  - 3 states, so 2^Q =
  - DFA M:
    - Q' =
    - Σ =
    - q_0' =
    - F' =



Equivalence of NFA's and DFA's

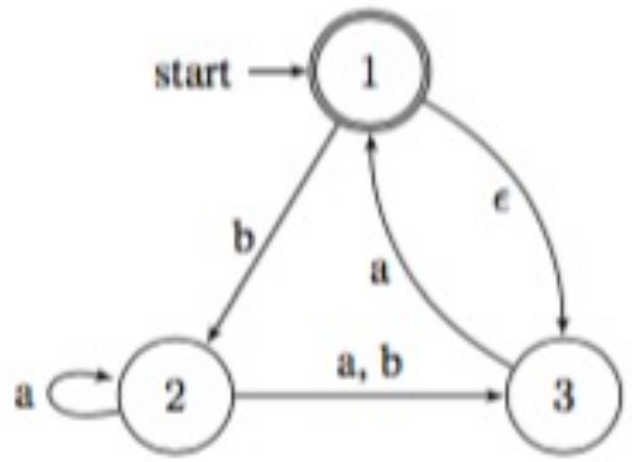
- Ex 2: NFA → DFA
  - NFA N:
  - 3 states, so $2^Q = 2^3 = 8$
  - DFA M:



  - Q' = power set which includes the empty set and any combination of states from Q
    - Q' = {∅, 1, 2, 3, 12, 13, 23, 123}
  - Σ = {a, b} (the alphabet does not change)
  - q_0' = E({q_0})
    - q_0' would equal 1 but there is an ε transition from 1 to 3,
    - q_0' = 13 since can slide to state 3 at the start or stay at state 1
  - F' = any state containing an original accept state
    - F' = {1, 12, 13, 123}

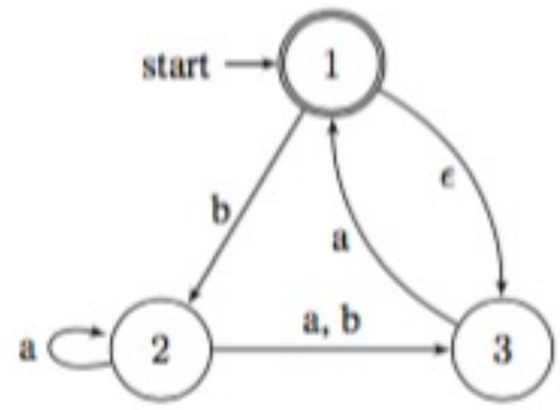
Equivalence of NFA's and DFA's

- Ex 2: NFA → DFA



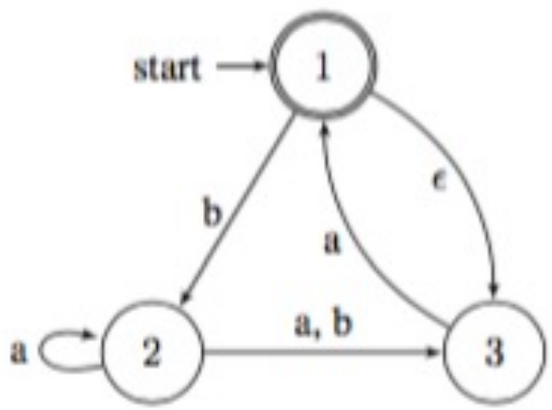
  - DFA M δ transitions: {∅, 1, 2, 3, 12, 13, 23, 123}

Equivalence of NFA's and DFA's



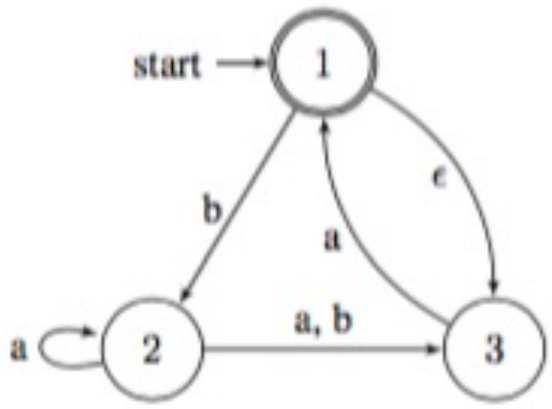
- Ex 2: NFA → DFA
  - DFA M δ transitions: {ø, 1, 2, 3, 12, 13, 23, 123}
    - At state 1 with an input of a, there is no transition, go to ø
    - At state 1 with an input of b, move to state 2
    - At state 2 with an input of a, stay at state 2 or move to state 3 (union of both = 23)
    - At state 2 with an input of b, move to state 3
    - At state 3 with an input of a, move to state 1 or slide back to state 3 (union of both = 13)
    - At state 3 with an input of b, there is no transition, go to ø
    - At state ø with an input of a or b stay at state ø

Equivalence of NFA's and DFA's



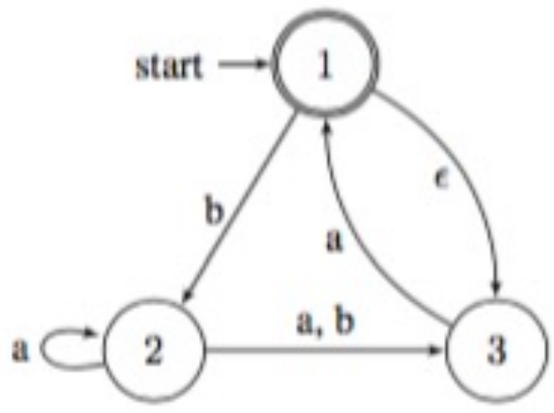
- Ex 2: NFA → DFA
  - DFA M δ transitions: {∅, 1, 2, 3, 12, 13, 23, 123}
    - At state 12 (look at state 1 and state 2) with an input of a, 1 goes to ∅ and 2 moves to 23 (union of both = 23)
    - At state 12 (look at state 1 and state 2) with an input of b, 1 goes to 2 and 2 moves to 3 (union of both = 23)
    - At state 13 (look at state 1 and state 3) with an input of a, 1 goes to ∅ and 3 moves to 1 or slides back to 3 (union of both = 13)
    - At state 13 (look at state 1 and state 3) with an input of b, 1 goes to 2 and 3 goes to ∅ (union of both = 2)

Equivalence of NFA's and DFA's



- Ex 2: NFA → DFA
  - DFA M δ transitions: {∅, 1, 2, 3, 12, 13, 23, 123}
    - At state 23 (look at state 2 and state 3) with an input of a, 2 goes to 23 and 3 moves to 1 or slides back to 3 (union of both = 123)
    - At state 23 (look at state 2 and state 3) with an input of b, 2 goes to 3 and 3 goes to ∅ (union of both = 3)
    - At state 123 (look at state 1, state 2 and state 3) with an input of a, 1 goes to ∅, 2 goes to 23 and 3 moves to 1 or slides back to 3 (union of all = 123)
    - At state 123 (look at state 1, state 2 and state 3) with an input of b, 1 goes to 2, 2 goes to 3 and 3 goes to ∅ (union of all = 23)

Equivalence of NFA's and DFA's

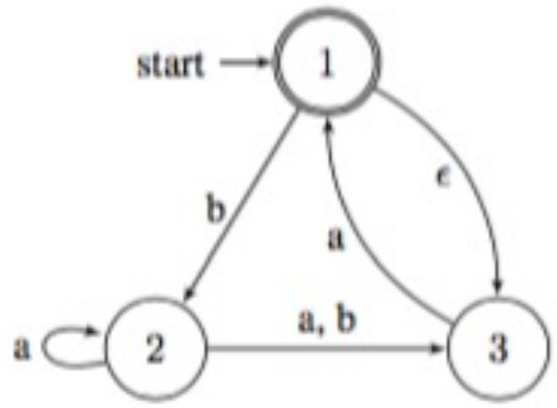


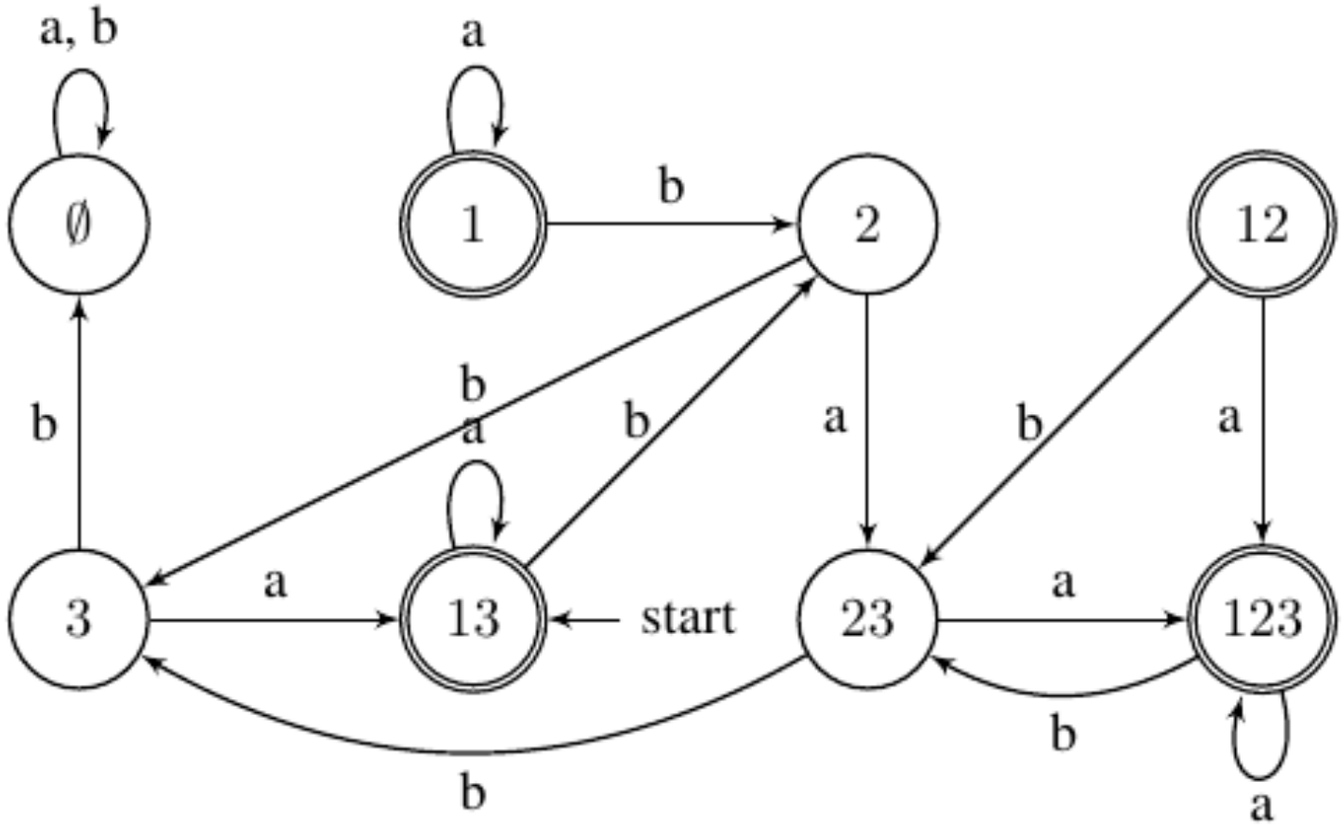
- Ex 2: NFA → DFA, cont.
  - NFA N:
  - 2 states, so $2^3 = 8$
  - DFA M:
    - $Q' = \{\emptyset, 1, 2, 3, 12, 13, 23, 123\}$
    - $\Sigma = \{a, b\}$
    - $q_0 = 13$
    - $F' = \{1, 12, 13, 123\}$

δ'	a	b
1	∅	2
2	23	3
3	13	∅
12	23	23
13	13	2
23	123	3
123	123	23
∅	∅	∅

Equivalence of NFA's and DFA's

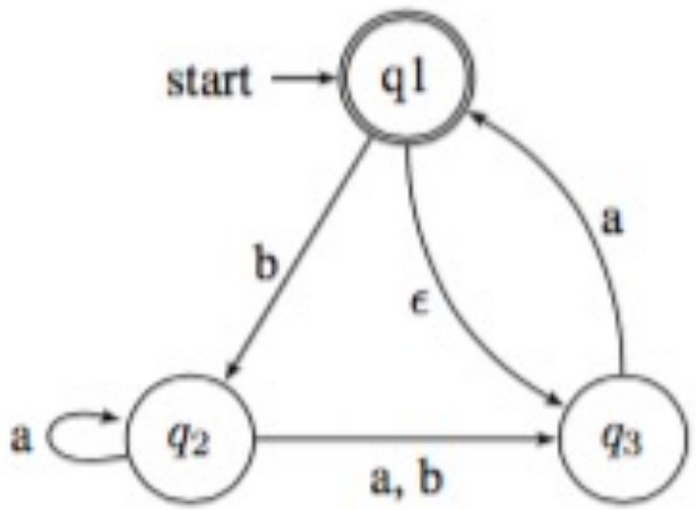
- Ex 2: NFA → DFA, cont.
  - NFA N:
  - DFA M:





Try It

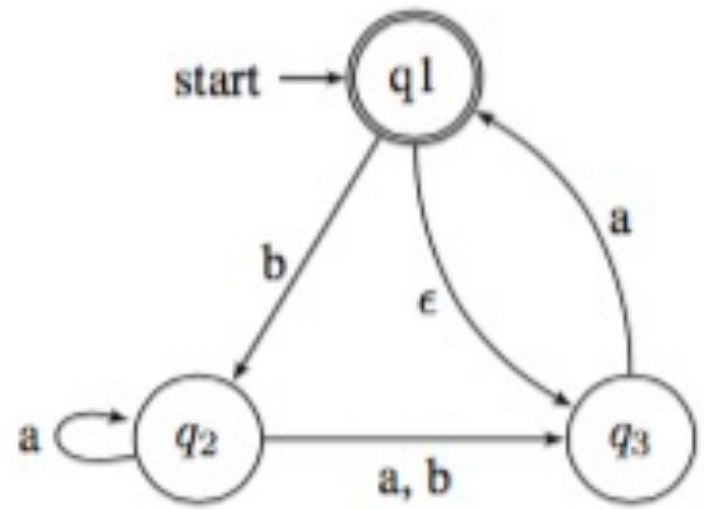
- Formally describe this NFA:



- What are some strings it will accept?

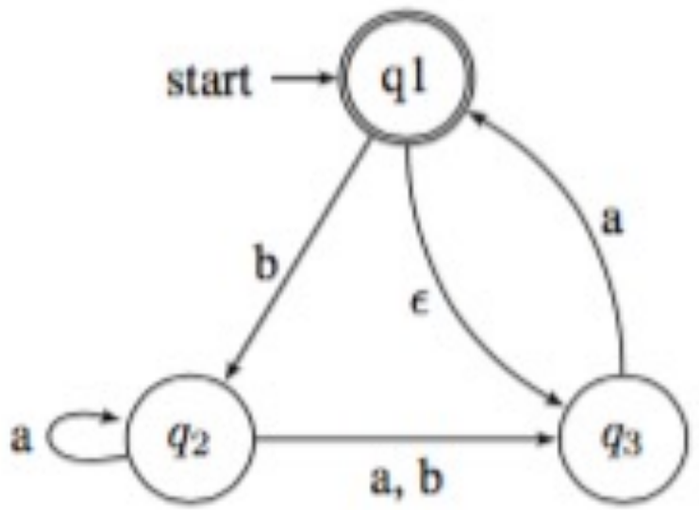
Try It

- Formally describe this NFA:
  - $Q = \{q_1, q_2, q_3\}$
  - $\Sigma = \{a, b\}$
  - $\Sigma_\varepsilon = \{a, b, \varepsilon\}$
  - $q_0 = q_1$
  - $F = \{q_1\}$



δ	a	b	ε
q_1	$\emptyset$	$\{q_2\}$	$\{q_3\}$
q_2	$\{q_2, q_3\}$	$\{q_3\}$	$\emptyset$
q_3	$\{q_1\}$	$\emptyset$	$\emptyset$

Try It



- What are some strings it will accept?
  - ε, a, bba, baa
  - Officially it accepts: ((ba*(a∪b) ∪ ε)(a*a)) ∪ ε

Theory of Computation
Chapter 1

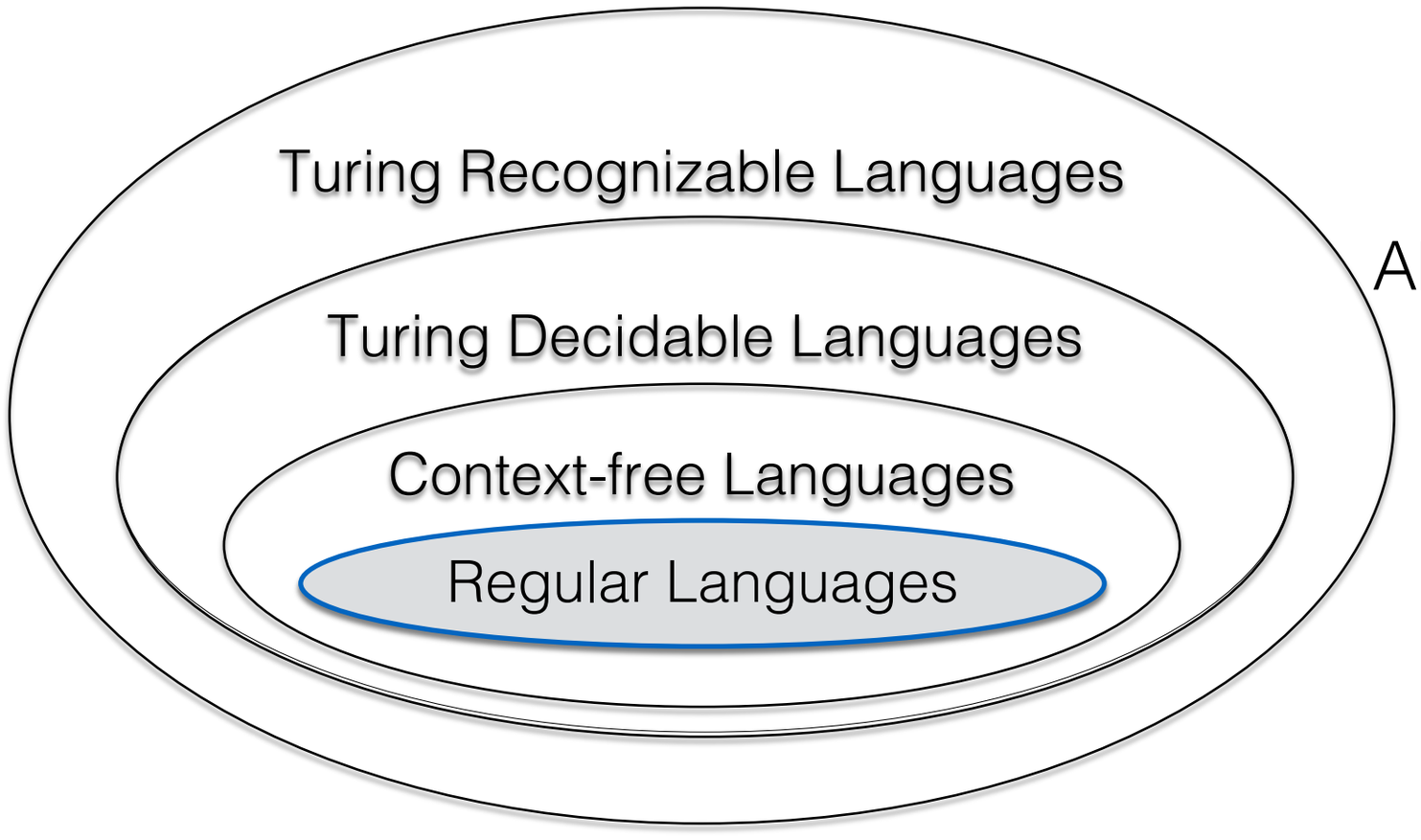
Regular Languages Part 2

School of Engineering | Computer Science

Gordon Moore
1929-2023

- The “Moore” of “Moore’s Law”
- Co-founder (1968) and longtime chairman and CEO of Intel Corp.
- Endowed Gordon and Betty Moore Foundation (with wife)
- One of original 8 founders (1957) of Fairchild Semiconductor, manufacturer of first cost-effective silicon integrated circuit

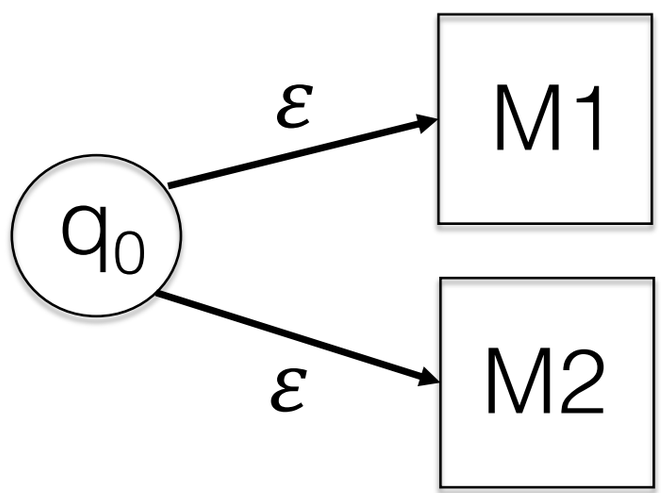
Regular Languages



Regular Languages
DFA = NFA = RE
Closed under union, $\cup$, concatenation, $\circ$, and star, $*$.

Closure Under Union

- Closure is much easier to show with NFA's verses DFA's
  - Union: $A \cup B = \{ x \mid x \in A \text{ or } x \in B\}$
    - M1 is a DFA that recognizes A and M2 is a DFA that recognizes B,
    - The union is an NFA, recognizing both of the languages.



Closure Under Union

- Union Formally Defined:
  - Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$
  - Construct $N_3 = (Q_3, \Sigma, \delta_3, q_3, F_3)$ as follows:

1. $Q_3 = Q_1 \cup Q_2 \cup \{q_0\}$
2. $q_3 = q_0$
3. $F_3 = F_1 \cup F_2$
4. δ_3 is defined such that for any $q \in Q$ and any $a \in \Sigma_\varepsilon$:

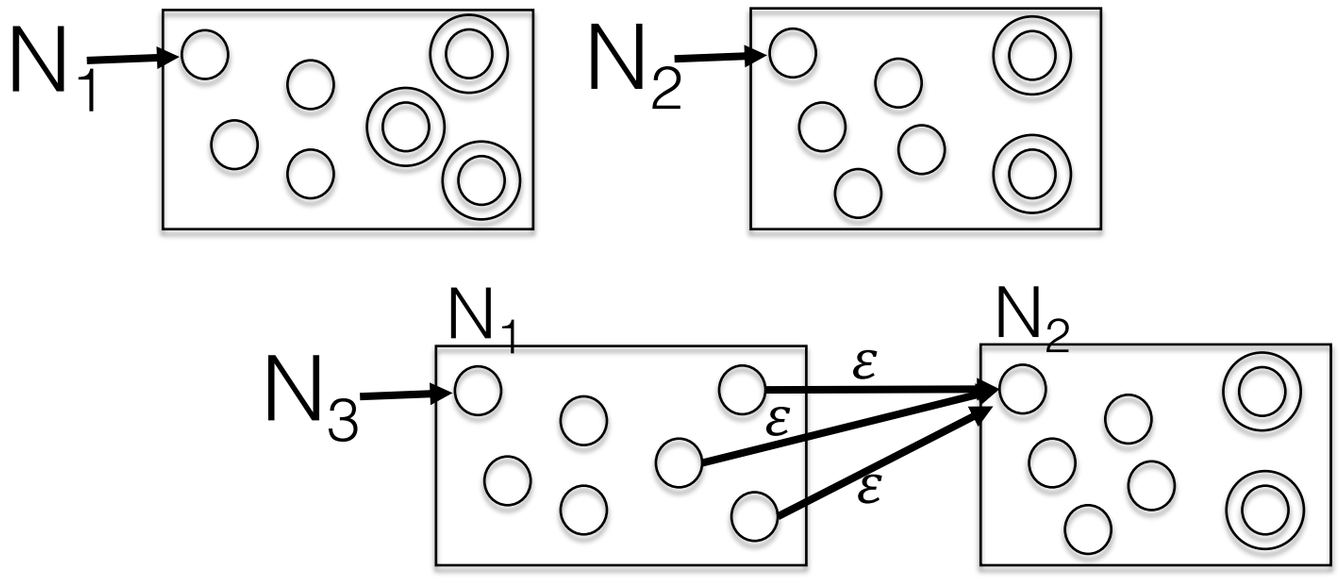
$$\delta_3(q, a) = \begin{cases} \delta_1(q, a) & q \in Q_1 \\ \delta_2(q, a) & q \in Q_2 \\ \{q_1, q_2\} & q = q_0 \text{ and } a = \varepsilon \\ \emptyset & q = q_0 \text{ and } a \neq \varepsilon \end{cases}$$

Closure Under Concatenation

- Theorem 1.47: Regular languages are closed under concatenation.
  - Let A and B be regular languages. Then $A \circ B$ is a regular language $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$

- Proof Idea: Let N_1 and N_2 be NFAs for A and B respectively

N_1 no longer has its accept states. They instead have ε transitions to N_2's start state.



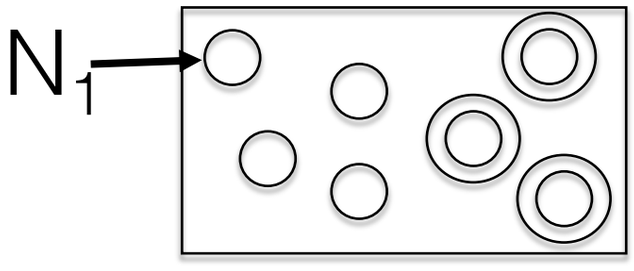
Closure Under Concatenation

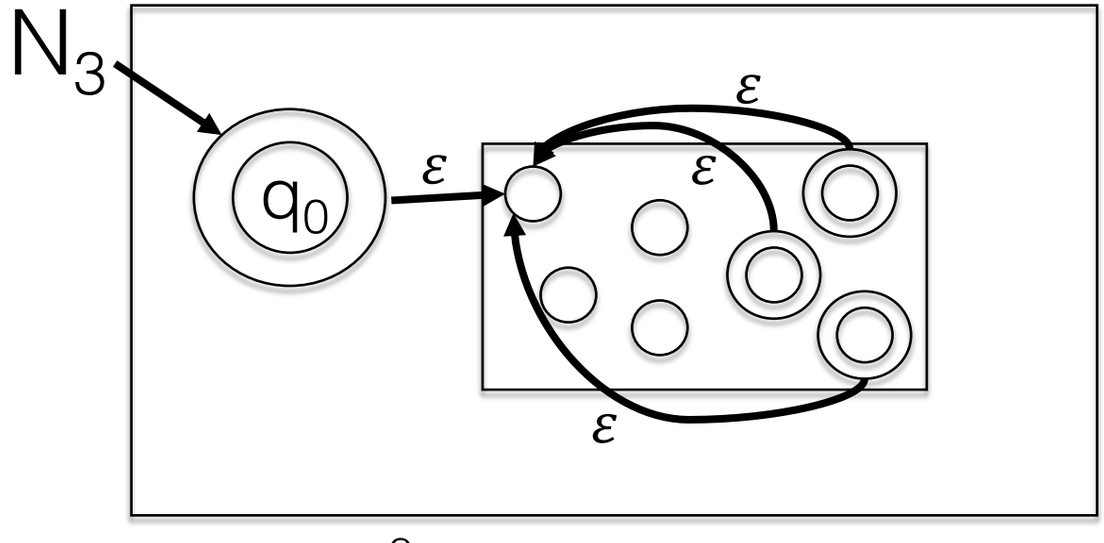
- Theorem 1.47: Regular languages are closed under concatenation.
  - Formally: Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$. Construct $N_3 = (Q_3, \Sigma, \delta_3, q_3, F_3)$ as follows:
    1. $Q_3 = Q_1 \cup Q_2$
    2. $q_3 = q_1$
    3. $F_3 = F_2$
    4. Define δ_3 such that for any $q \in Q_3$ and any $a \in \Sigma_\varepsilon$

$$\delta_3 \begin{cases} \delta_1(q, a) & q \in Q_1 \text{ and } q \notin F_1 \\ \delta_2(q, a) & q \in Q_2 \\ \delta_1(q, a) \cup \{q_2\} & q \in F_1 \text{ and } a = \varepsilon \\ \delta_1(q, a) & q \in F_1 \text{ and } a \neq \varepsilon \end{cases}$$

Closure Under Star

- Theorem 1.49: Regular languages are closed under star.
  - Let A be a regular language. Then A* is a regular language
  - $A^* = \{x_1x_2\ldots x_k \mid k \geq 0 \text{ and } x_i \in A \text{ for all } 1 \leq i \leq k\}$
  - Proof Idea: Let N_1 be a NFA for A





There is a new start state that is an accept state. Each accept state transitions to the old start state.

Closure Under Star

- Theorem 1.49: Regular languages are closed under star.
  - Formally: Let $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$. Construct $N_3 = (Q_3, \Sigma, \delta_3, q_3, F_3)$ as follows:
    1. $Q_3 = Q_1 \cup \{q_0\}$
    2. $q_3 = q_0$
    3. $F_3 = F_1 \cup \{q_0\}$
    4. Define δ_3 such that for any $q \in Q_3$ and any $a \in \Sigma_\varepsilon$

$$\delta_3 \begin{cases} \delta_1(q, a) & q \in Q_1 \text{ and } q \notin F_1 \\ \delta_1(q, a) & q \in F_1 \text{ and } a \neq \varepsilon \\ \delta_1(q, a) \cup \{q_1\} & q \in F_1 \text{ and } a = \varepsilon \\ \{q_1\} & q = q_0 \text{ and } a = \varepsilon \\ \emptyset & q = q_0 \text{ and } a \neq \varepsilon \end{cases}$$

Regular Expressions

- We can use regular operations to build up expressions describing languages.
- They are a sequence of characters and are used to describe patterns in text
  - Ex: list all words that start with the letters “pre”
    - Use “pre”Σ^* (where Σ is the English alphabet)

Regular Expressions

- R is a regular expression (RE) if R is one of the following:
  1. a for some a $\in \Sigma$
  2. ε
  3. $\emptyset$
  4. $R_1 \cup R_2$ for regular expressions R_1 and R_2
  5. $R_1 \circ R_2$ for regular expressions R_1 and R_2
  6. R_1^* for regular expression R_1
- These are the possible cases of regular expressions. You can build the regular expressions for this course from just these 6 cases.

Regular Expressions

- Given the possible cases of regular expressions, the language of these regular expressions corresponds to:

1. $L(R) = \{a\}$
2. $L(R) = \{\varepsilon\}$
3. $L(R) = \emptyset$
4. $L(R_1 \cup R_2) = L(R_1) \cup L(R_2)$
5. $L(R_1 \circ R_2) = L(R_1) \circ L(R_2)$
6. $L(R_1^*) = (L(R_1))^*$

Regular Expressions

- Precedence (Order of Operations)
  1. Parentheses
  2. Star (exponent)
  3. Concatenation (multiplication)
  4. Union (addition)

- Convention: $R^+ = R^1R^*$ (At least one copy of R then * copies of R afterwards)

Regular Expressions

- Examples: Describe the language elaborated by the following regular expressions. Let $\Sigma = \{0, 1\}$

1. $0^*10^* =$
2. $\Sigma^*1\Sigma^* =$
3. $1^*(01^+)^* =$
4. $(\Sigma\Sigma\Sigma)^* =$
5. $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1 =$
6. $(0 \cup \varepsilon) \circ (1 \cup \varepsilon) =$

Regular Expressions

- Examples: : Describe the language elaborated by the following regular expressions. Let $\Sigma = \{0, 1\}$
  1. $0^*10^* = \{w \mid w \text{ contains a single 1}\}$
  2. $\Sigma^*1\Sigma^* = \{w \mid w \text{ contains at least one 1}\}$
  3. $1^*(01^+)^* = \{w \mid \text{every 0 in w is followed by at least one 1}\}$
  4. $(\Sigma\Sigma\Sigma)^* = \{w \mid \text{length of w is a multiple of 3}\}$
  5. $0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1 = \{w \mid w \text{ starts and ends with the same symbol, with length at least 1}\}$
  6. $(0 \cup \varepsilon) \circ (1 \cup \varepsilon) = \{\varepsilon, 0, 1, 01\}$

Regular Expressions

- Examples: Build a regular expression for the following. Let $\Sigma = \{0, 1\}$
  1. {w | w ends with a 1}
  2. {w | w contains 1001}
  3. {w | every 1 in w is followed by at least one 0}
  4. {w | length of w is a multiple of 2}
  5. {w | w has length at least 3 and starts and ends with a different symbol}
  6. {w | w contains at least one 0 and exactly two 1s}

Regular Expressions

- Examples: Build a regular expression for the following. Let $\Sigma = \{0, 1\}$

1. $\{w \mid w \text{ ends with a } 1\}$ $\quad \Sigma^*1$
2. $\{w \mid w \text{ contains } 1001\}$ $\quad \Sigma^*1001\Sigma^*$
3. $\{w \mid \text{every 1 in w is followed by at least 1 0}\}$ $\quad 0^*(10^+)^*$
4. $\{w \mid \text{length of w is a multiple of 2}\}$ $\quad (\Sigma\Sigma)^*$
5. $\{w \mid w \text{ has length at least 3 and starts and ends with a different symbol}\}$ $\quad 0\Sigma^+1 \cup 1\Sigma^+0$
6. $\{w \mid w \text{ contains at least one 0 and exactly two 1s}\}$ $\quad 0^*10^+10^* \cup 0^*10^*10^+ \cup 0^+10^*10^*$

Regular Expressions

- Boundary Cases
  1. $1^*\emptyset =$
  2. $\emptyset^* =$
  3. $R \cup \emptyset =$
  4. $R \cup \varepsilon =$
  5. $R \circ \varepsilon =$
  6. $R \circ \emptyset =$

Regular Expressions

- Boundary Cases
  1. $1^*\emptyset = \emptyset$ ($1^* \circ \emptyset$ - any set concatenated with the empty set gives the empty set)
  2. $\emptyset^* = \{\varepsilon\}$ (can only generate 0 strings from an empty language)
  3. $R \cup \emptyset = R$ (adding the empty set does not change R)
  4. $R \cup \varepsilon \neq R$ (cannot assume ε is in R)
  5. $R \circ \varepsilon = R$ (joining the empty string to any string will not change the string)
  6. $R \circ \emptyset = \emptyset$ (ex: R = 0, L(R) = {0}, $L(R \circ \emptyset) = \emptyset$, can also use case 1 above)

Regular Expressions

- Regular Languages are closed under
  - Union
  - Concatenation
  - Star
- Regular expressions are equivalent to deterministic finite automata which are equivalent to non-deterministic finite automata
  - RE = DFA = NFA

Try It

- Write a regular expression for the language of strings (with $\Sigma = \{0, 1\}$) for the following:
  - {w | w either ends in 01 or has a length divisible by 4 (or both)}
  - {w | w contains exactly one 0s and at least three 1s}

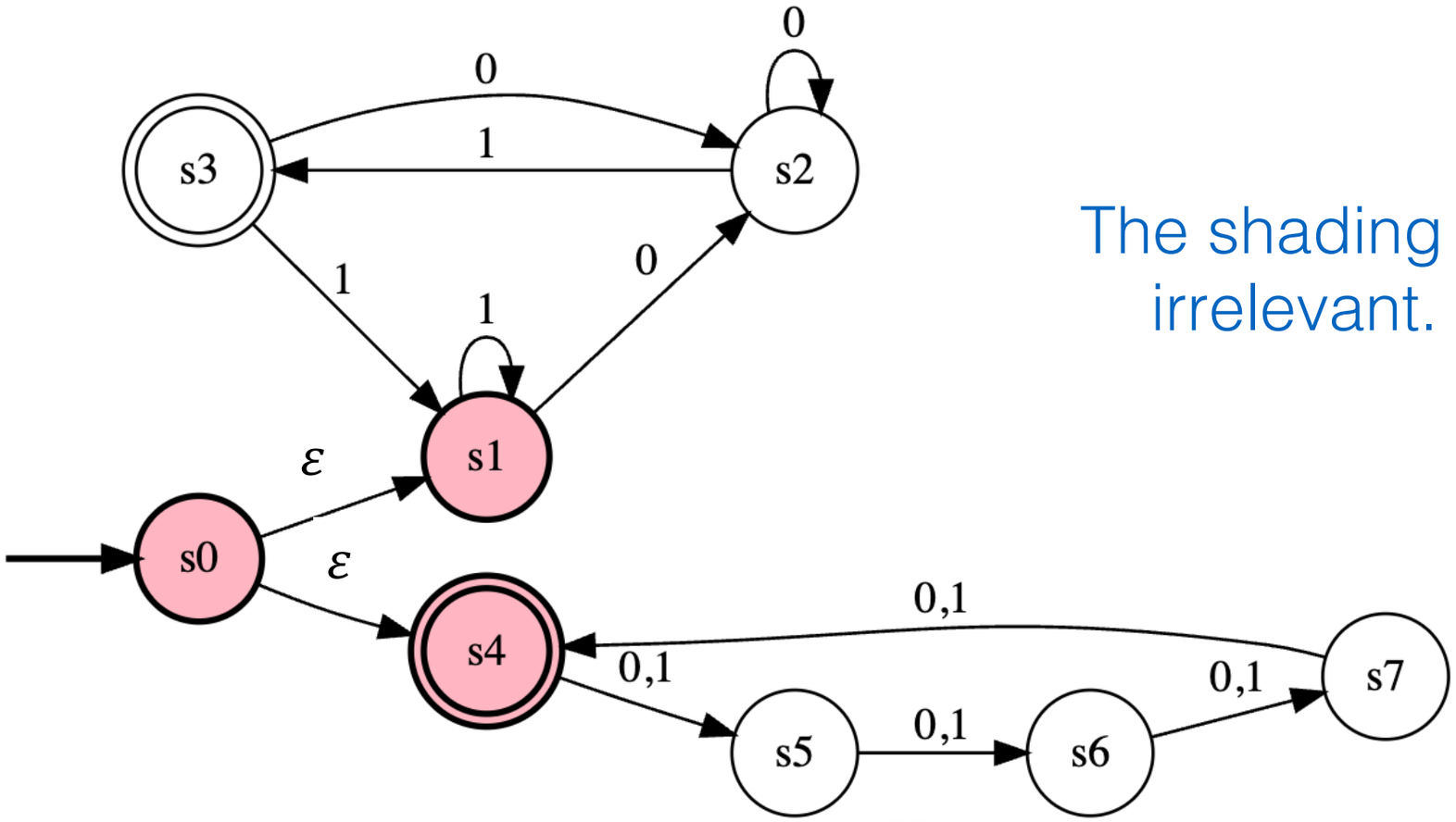
- Construct two separate DFAs (with $\Sigma = \{0, 1\}$) that accept (i) strings that end in 01 and (ii) strings with length divisible by 4. Combine them to construct an NFA that accepts the union of both.

Try It

- Write a regular expression for the language of strings (with $\Sigma = \{0, 1\}$) for the following:
  - {w | w either ends in 01 or has a length divisible by 4 (or both)} $\Sigma^*01 \cup (\Sigma\Sigma\Sigma\Sigma)^*$
  - {w | w contains exactly one 0s and at least three 1s} $1^+1^+1^+0 \cup 1^+1^+01^+ \cup 1^+01^+1^+ \cup 01^+1^+1^+$

Try It

- Construct two separate DFAs (with $\Sigma = \{0, 1\}$) that accept (i) strings that end in 01 and (ii) strings with length divisible by 4. Combine them to construct an NFA that accepts the union of both.



The shading is irrelevant.

Theory of Computation
Chapter 1

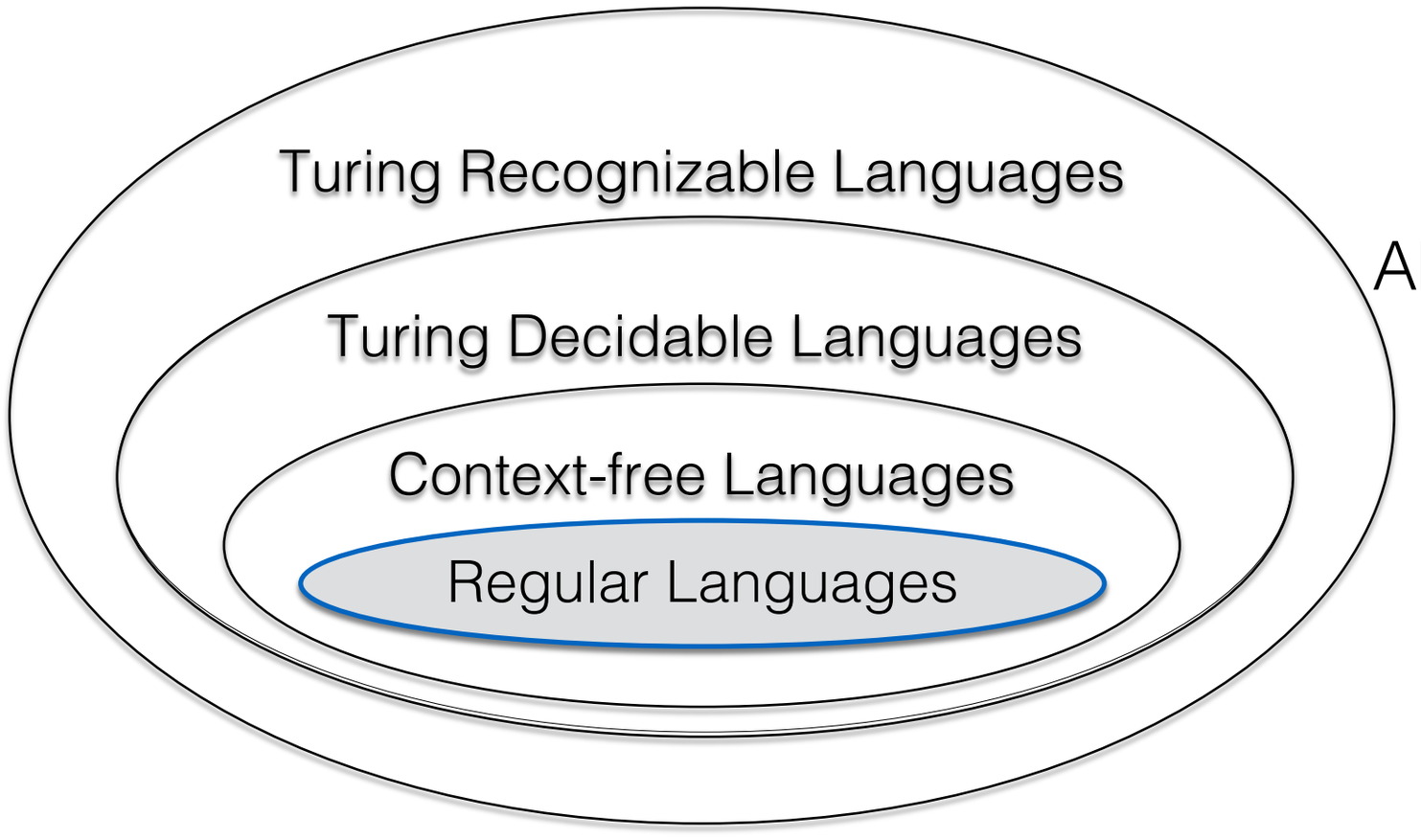
Regular Languages Part 3

School of Engineering | Computer Science

Dennis Ritchie
1941 - 2011

- Creator of C programming language
  - co-author of *The C Programming Language* (K&R)
- Key player in creation of Unix operating system
- Winner (with K. Thompson) of 1983 Turing Award

Regular Languages



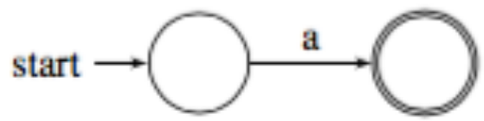
Regular Languages
DFA = NFA = RE
Closed under union, $\cup$, concatenation, $\circ$, and star, $*$.

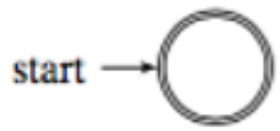
Regular Language

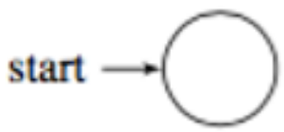
- Theorem 1.54: A language is regular if and only if some regular expression describes it (if and only if means two directions)
  - Forward Direction: Lemma 1.55
    - Claim: If a language L is described by a regular expression, then the language L is regular
    - Proof: Given a regular expression R, we will convert it into a NFA N such that the language $L(R) = L(N)$

Regular Language, Forward

- Theorem 1.54: A language is regular if and only if some regular expression describes it
  - Forward: Lemma 1.55
    - There are 6 cases from the definition of regular expressions. Here are the NFAs for each case:

1. R = a for some $a \in \Sigma$ 

2. R = ε 

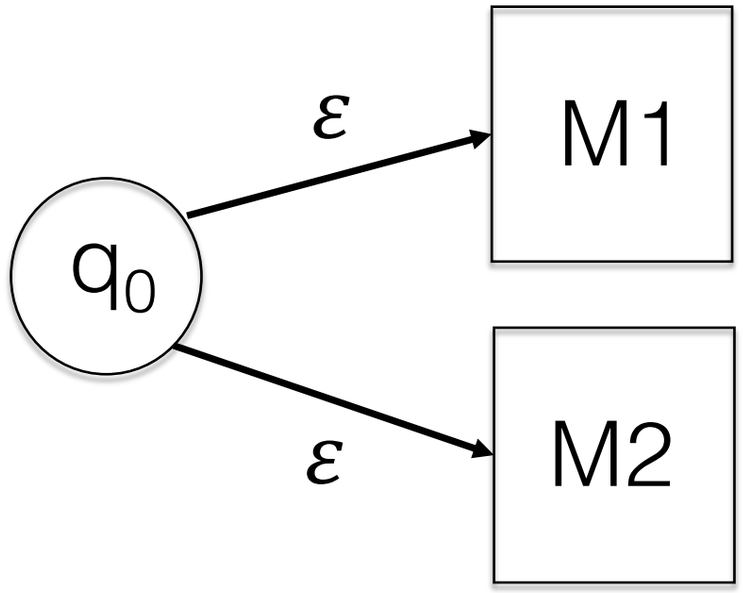
3. R = $\emptyset$ 

4. R = $R_1 \cup R_2$
5. R = $R_1 \circ R_2$
6. R = R_1^*

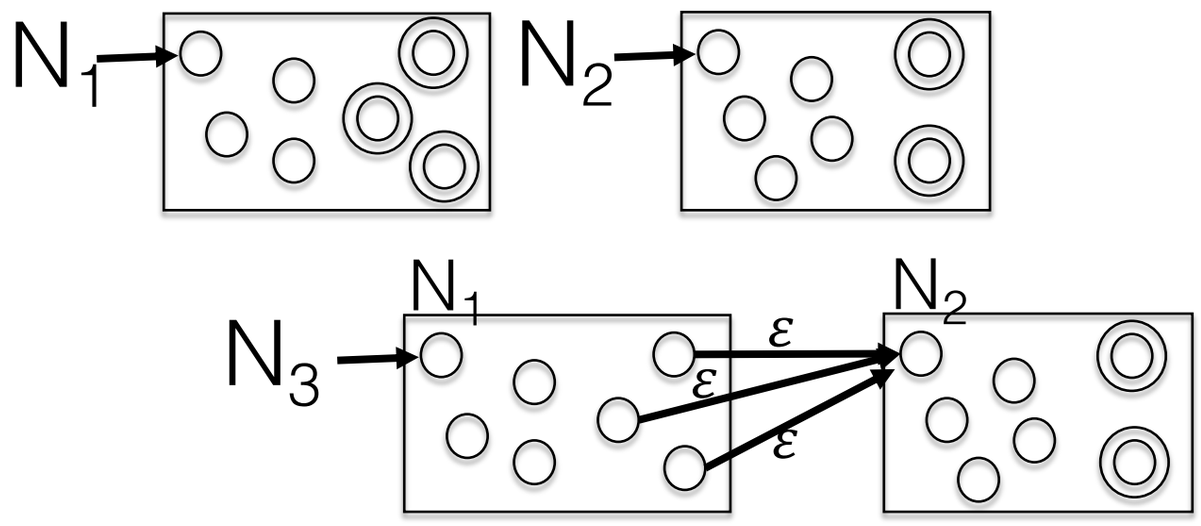
For these cases, we build the NFAs as we did in previous slides.

Regular Language Operations

- Remember:
  - Union: $R = R_1 \cup R_2$

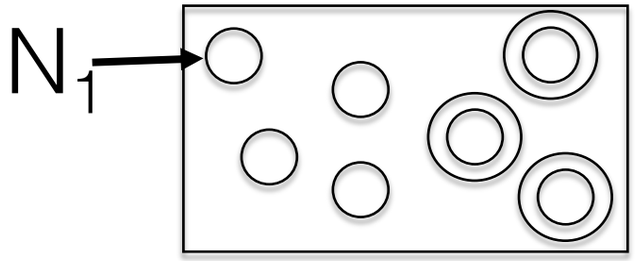


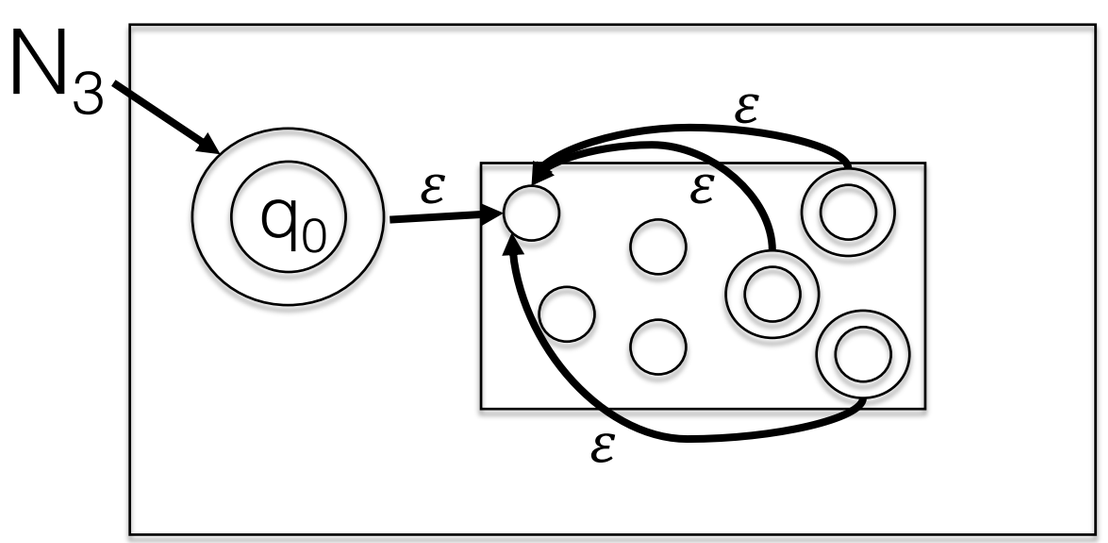
- Concatenation: $R = R_1 \circ R_2$



Regular Language Operations

- Remember:
  - Star: R = R_1*





Regular Language, Forward

- Forward: Lemma 1.55
  - Ex: Given regular expression (01 ∪ 0)* find the NFA
    - 0
    - 1
    - 01 or 0 ∘1
    - 01 ∪ 0
    - (01 ∪ 0)*

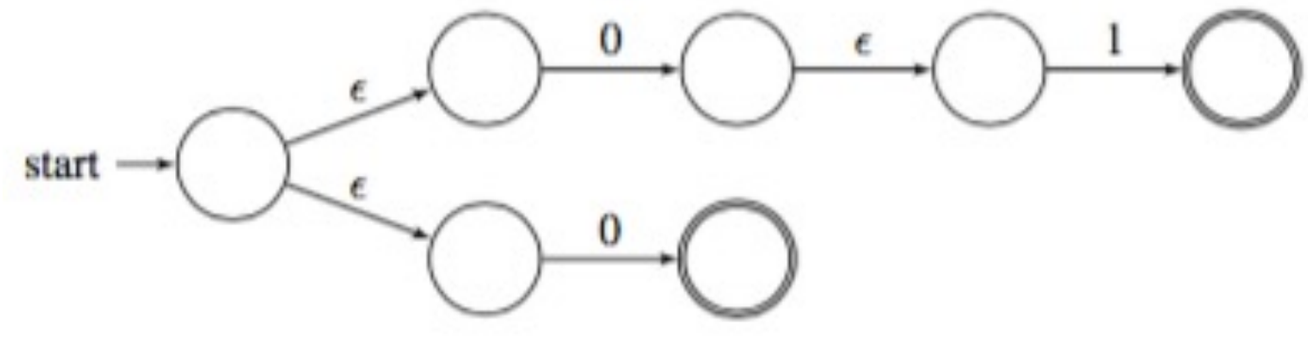
Regular Language, Forward

- Forward: Lemma 1.55
  - Ex: Given regular expression (01 ∪ 0)* find the NFA
    - 0

    - 1

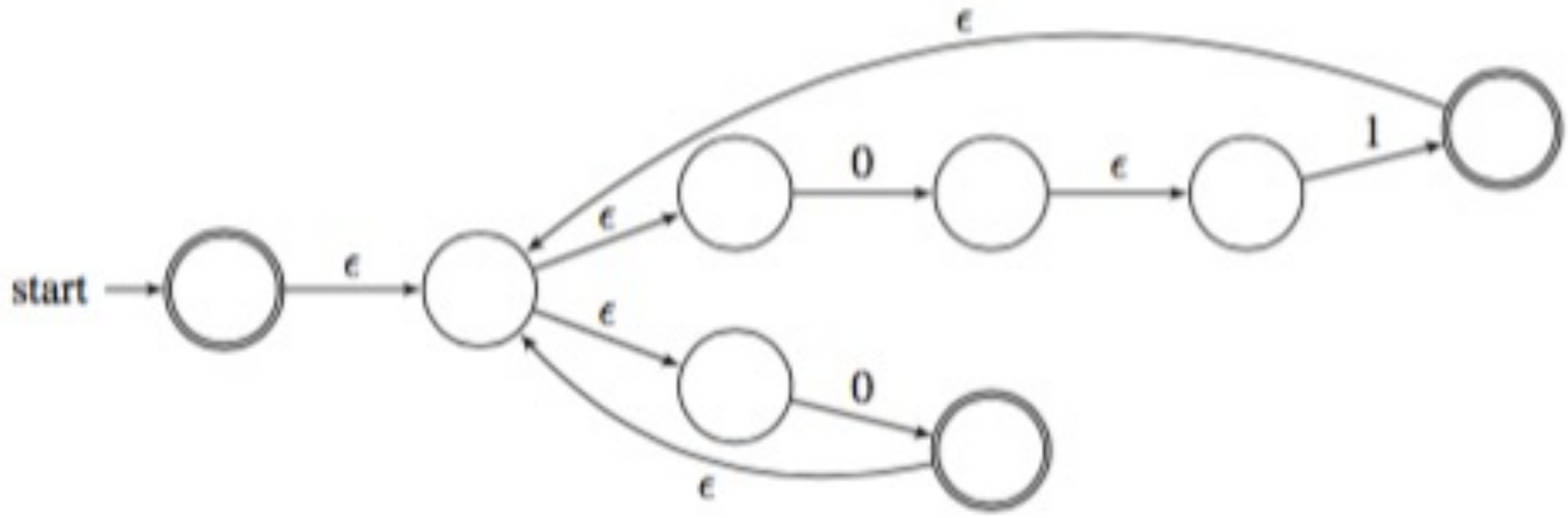
    - 01 or 0 ∘1

    - 01 ∪ 0



Regular Language, Forward

- Forward: Lemma 1.55
  - Ex: Given regular expression (01 ∪ 0)* find the NFA, cont.
  - (01 ∪ 0)*



Regular Language, Forward

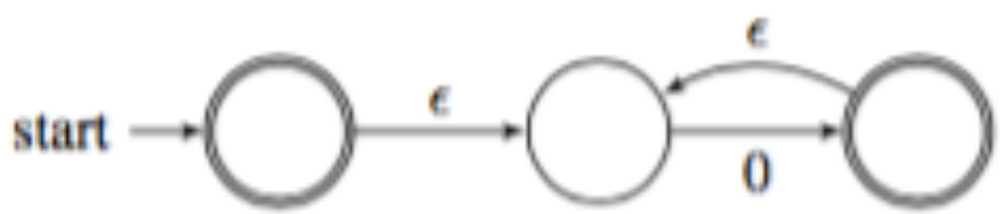
- Forward: Lemma 1.55
  - Ex 2: Given regular expression (0*11) ∪ (01)* find the NFA
    - 0
    - 1
    - 0*
    - 11

Regular Language, Forward

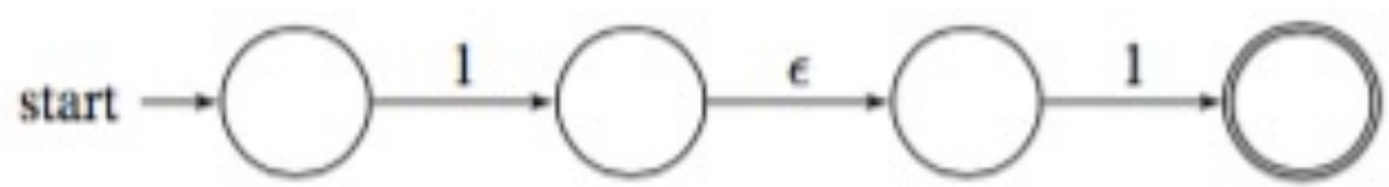
- Forward: Lemma 1.55
  - Ex 2: Given regular expression (0*11) ∪ (01)* find the NFA
    - 0

    - 1

    - 0*



    - 11

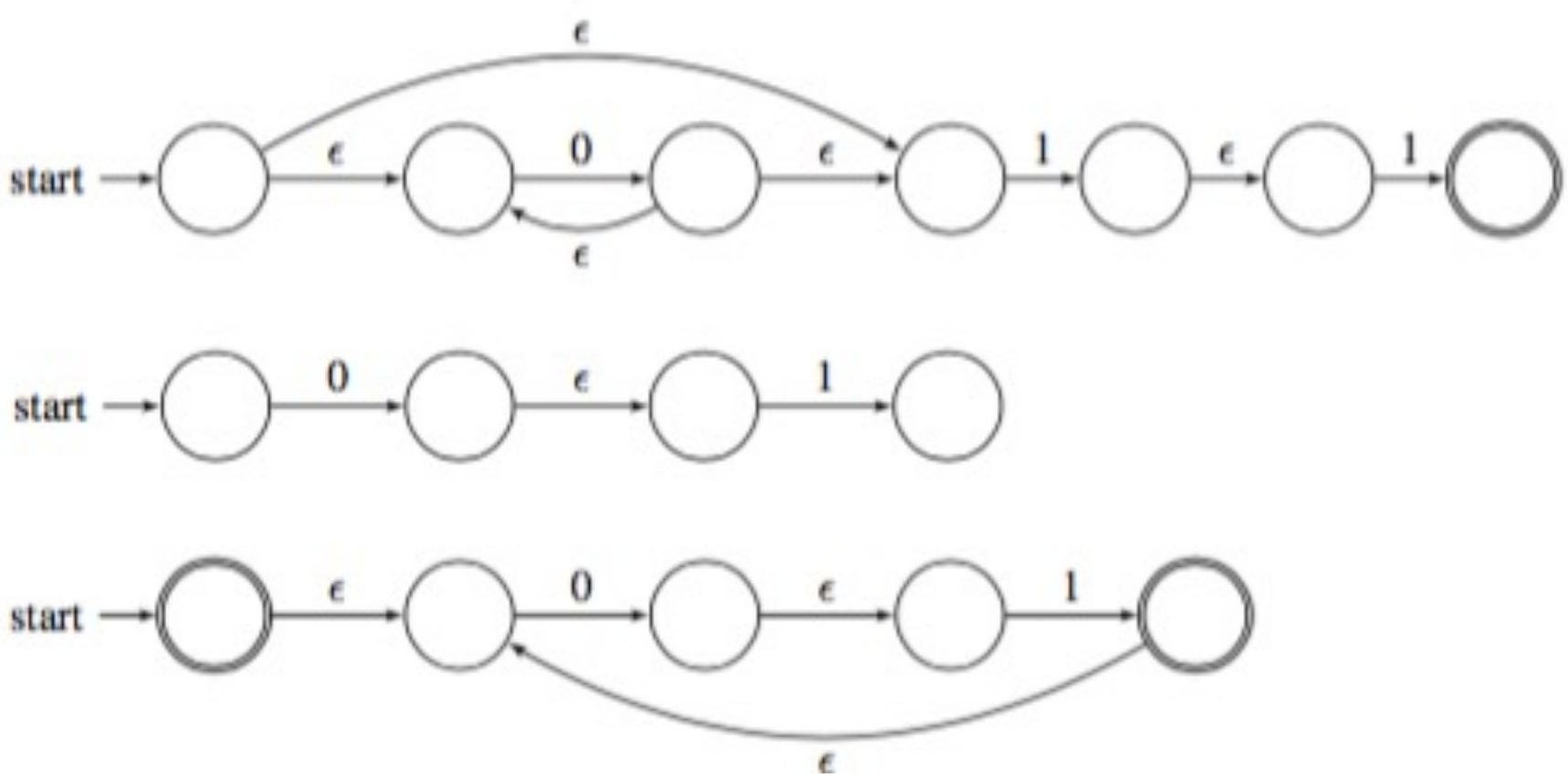


Regular Language, Forward

- Forward: Lemma 1.55
  - Ex 2: Given regular expression (0*11) ∪ (01)* find the NFA, cont,
    - 0*11
    - 01
    - (01)*

Regular Language, Forward

- Forward: Lemma 1.55
  - Ex 2: Given regular expression (0*11) ∪ (01)* find the NFA, cont,
    - 0*11
    - 01
    - (01)*



Regular Language, Forward

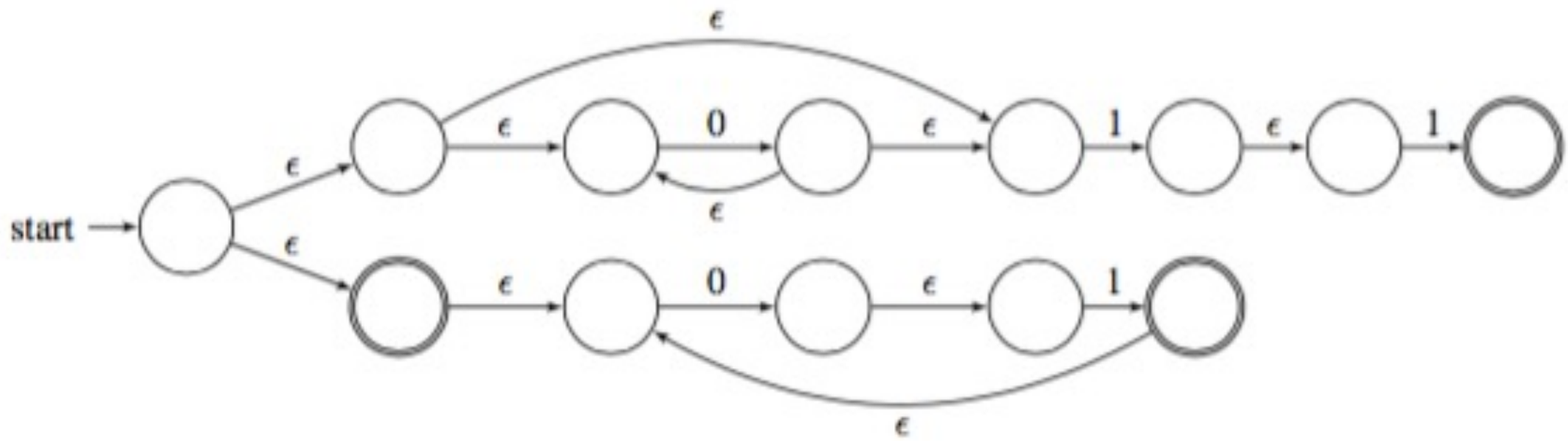
- Forward: Lemma 1.55
  - Ex 2: Given regular expression (0*11) ∪ (01)* find the NFA, cont,

- (0*11) ∪ (01)*

Regular Language, Forward

- Forward: Lemma 1.55
  - Ex 2: Given regular expression (0*11) ∪ (01)* find the NFA, cont,

- (0*11) ∪ (01)*



Regular Language, Backwards

- Theorem 1.54: A language is regular if and only if some regular expression describes it
  - Backwards: Lemma 1.60
  - Claim: If a language is regular, then it can be described by a regular expression, R
  - Proof: Assume L is a regular language with a DFA $D = (Q, \Sigma, \delta, q, F)$, we can construct a regular expression R for L
    - $L(R) = L(D) = L$

Regular Language

- Theorem 1.54: A language is regular if and only if some regular expression describes it
  - Backwards: Lemma 1.60
  - Idea: Describe language with a regular expression
    1. Start with a DFA recognizing L
    2. Convert L into a "generalized NFA" G whose transitions are labeled by regular expressions
    3. Recursively reduce the number of states until G has two remaining states and a single transition labeled by some regular expression R

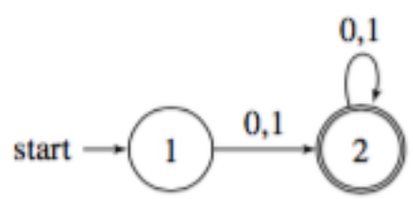
Regular Language

- Lemma 1.60: ("has DFA" ⇒ "has regular expression")
  - Step 1: Setting up the generalized NFA
    - Let $D = \{Q, \Sigma, \delta, q_0, F\}$
    1. Add a new start state q_{start} with an ε-transition to q_0
    2. Add a new end state (accept state) q_{end} with ε-transitions from all $q_i \in F$ and mark all q_i as non-accept states
    3. For each transition in D with multiple labels, replace each transition's label with the union of all old labels
    4. Between states with no transitions add arrows labeled by ∅ EXCEPT that there are no incoming ∅-transitions to q_{start} and no outgoing ∅-transitions from q_{end}

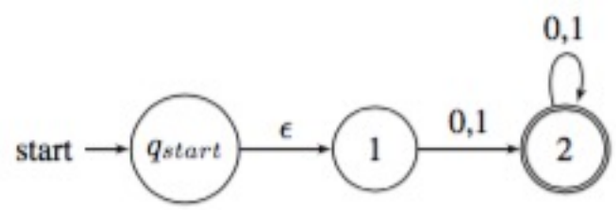
Regular Language

- Lemma 1.60: ("has DFA" ⇒ "has regular expression")
  - Step 1: Setting up the generalized NFA - Example

- DFA

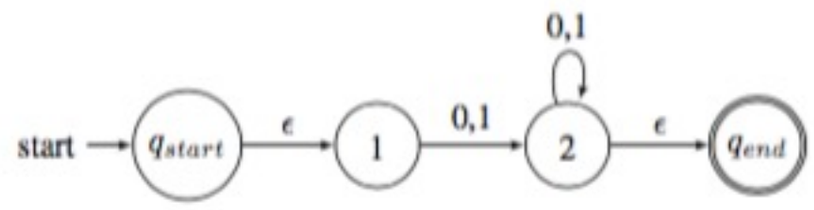


1. NFA



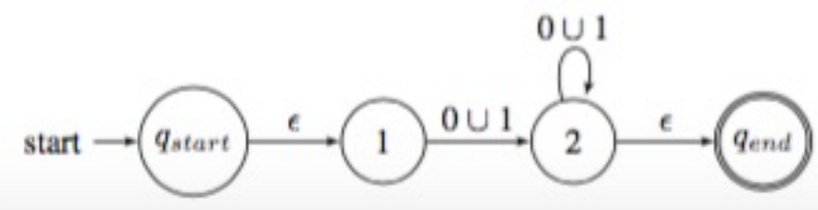
New start state

2. NFA



New end state

3. NFA

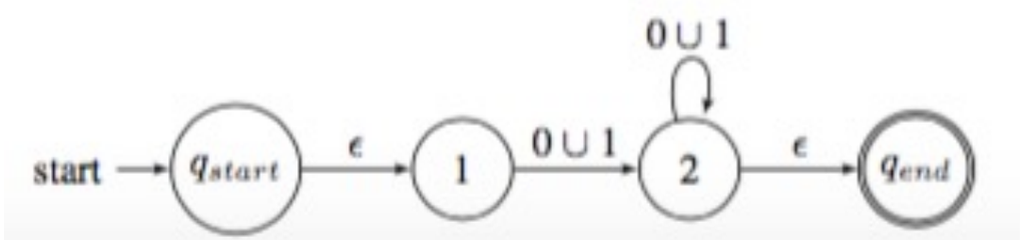


Multiple labels replaced by union

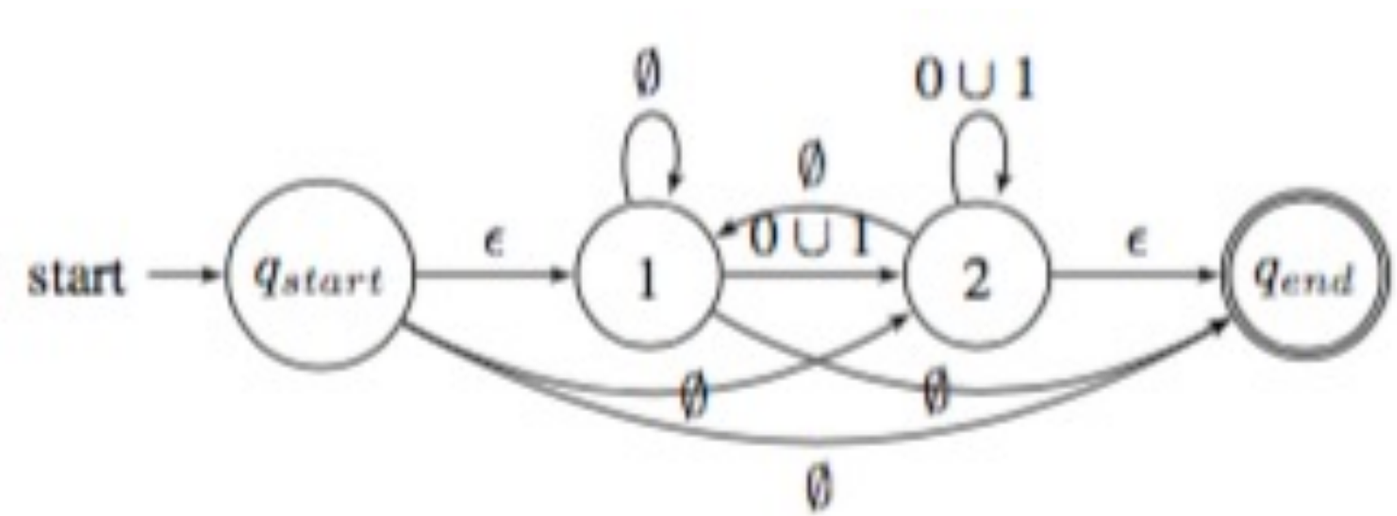
Regular Language

- Lemma 1.60: ("has DFA" ⇒ "has regular expression")
  - Step 1: Setting up the generalized NFA - Example

3. NFA



4. NFA with added transitions



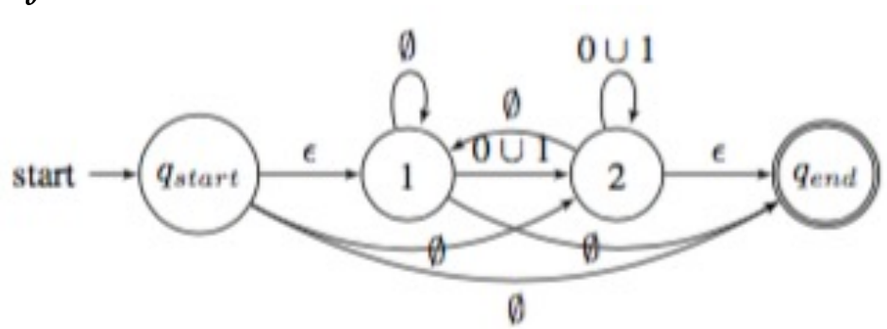
Regular Language

- Lemma 1.60: (“has DFA” ⇒ “has regular expression”)
  - Step 2: Reduce the NFA down to two states
    - Goal: k = 2

    - Let k be the number of states in the generalized NFA G
    - Base Case: If $k = 2$, return the regular expression on the arrow from q_{start} to q_{end}
    - Recursive Case: $k > 2$ states:
      - Pick any state (q_{rip}) to remove, $q_{rip} \in Q$, such that $q_{rip} \notin \{q_{start}, q_{end}\}$
      - Remove q_{rip} from G (continued)

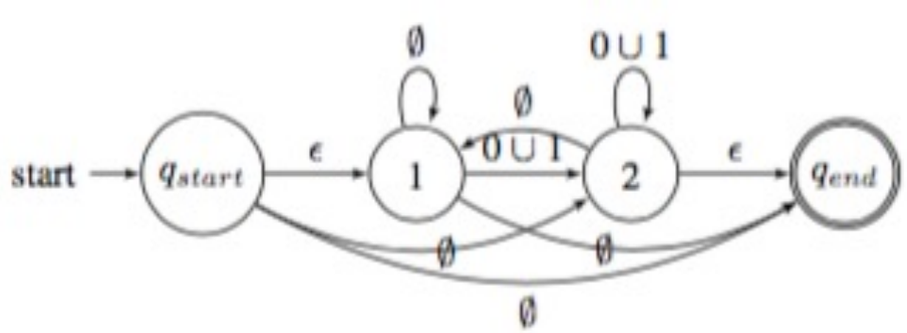
Regular Language

- Lemma 1.60: ("has DFA" ⇒ "has regular expression")
  - Step 2: Reduce the NFA down to two states
    - Recursive Case: $k > 2$ states:
      - Remove q_{rip} from G
        - For all pairs of states, $q_i \in Q$, but not q_{rip} or q_{end} and $q_j \in Q$, but not q_{rip} or q_{start} , set $\delta(q_i, q_j) = R_1 R_2^* R_3 \cup R_4$
          - $R_1 = \delta(q_i, q_{rip})$ q_i goes to q_{rip}
          - $R_2 = \delta(q_{rip}, q_{rip})$ q_{rip} goes to itself
          - $R_3 = \delta(q_{rip}, q_j)$ q_{rip} goes to q_j
          - $R_4 = \delta(q_i, q_j)$ q_i goes to q_j



Regular Language

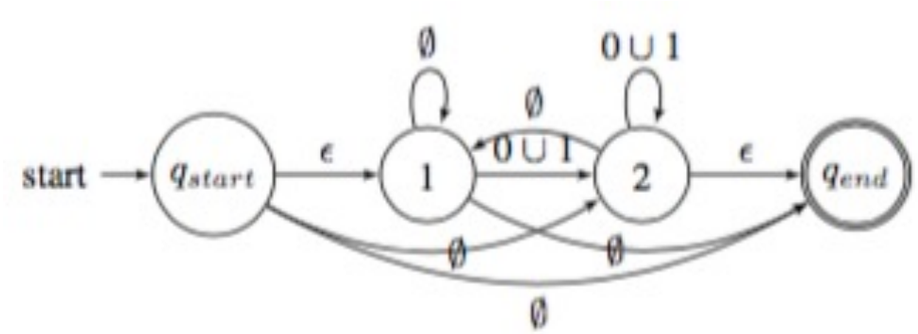
- Lemma 1.60: ("has DFA" ⇒ "has regular expression")
  - Step 2: Reduce the NFA down to two states – Example
    - Recursive Case: $k > 2$ states:



- Ex: Let q_{rip} = state 1, must consider two pairs (q_i, q_j):
  - (q_i, q_j) = (q_{start}, state 2)
  - (q_i, q_j) = (state 2, state 2)

Regular Language

- Lemma 1.60: ("has DFA" ⇒ "has regular expression")
  - Step 2: Reduce the NFA down to two states – Example
    - Recursive Case: k > 2 states:

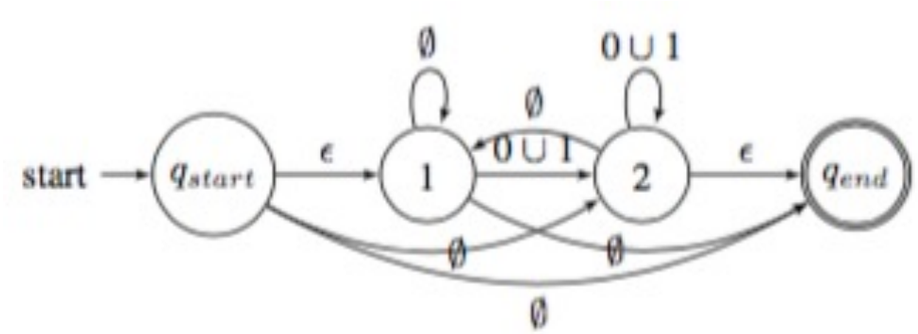


$R_1 = \delta(q_i, q_{rip})$
q_i goes to q_{rip}
$R_2 = \delta(q_{rip}, q_{rip})$
q_{rip} goes to itself
$R_3 = \delta(q_{rip}, q_j)$
q_{rip} goes to q_j
$R_4 = \delta(q_i, q_j)$
q_i goes to q_j

- Ex: Let q_{rip} = state 1:
  - (q_i, q_j) = (q_{start}, state 2)
    - (R1 ∘ R2* ∘ R3) ∪ R4
    - = (ε ∘ ∅* ∘ (0 ∪ 1)) ∪ ∅ = 0 ∪ 1

Regular Language

- Lemma 1.60: ("has DFA" ⇒ "has regular expression")
  - Step 2: Reduce the NFA down to two states – Example
    - Recursive Case: k > 2 states:

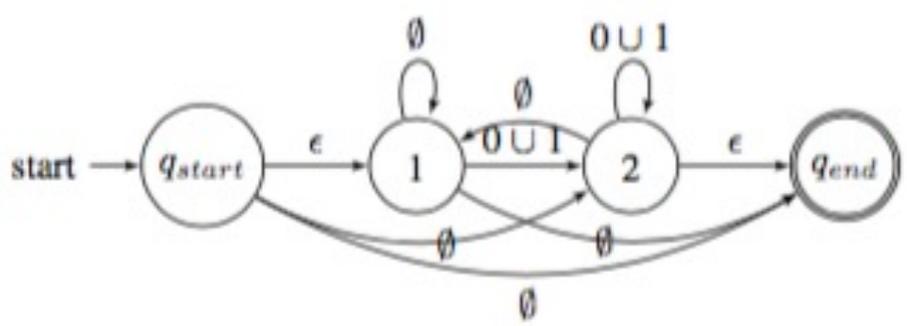


$R_1 = \delta(q_i, q_{rip})$
q_i goes to q_{rip}
$R_2 = \delta(q_{rip}, q_{rip})$
q_{rip} goes to itself
$R_3 = \delta(q_{rip}, q_j)$
q_{rip} goes to q_j
$R_4 = \delta(q_i, q_j)$
q_i goes to q_j

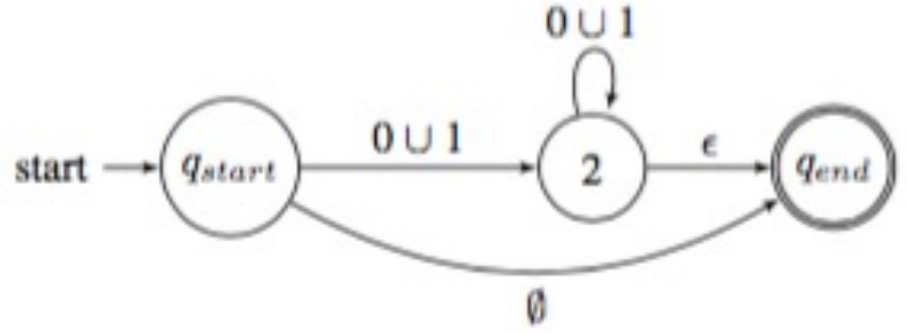
- Ex: Let q_{rip} = state 1:
  - (q_i, q_j) = (state 2, state 2)
    - (R1 ∘ R2* ∘ R3) ∪ R4
    - = (∅ ∘ ∅* ∘ (0 ∪ 1)) ∪ (0 ∪ 1) = 0 ∪ 1

Regular Language

- Lemma 1.60: ("has DFA" ⇒ "has regular expression")
  - Step 2: Reduce the NFA down to two states – Example
    - Recursive Case: k > 2 states:

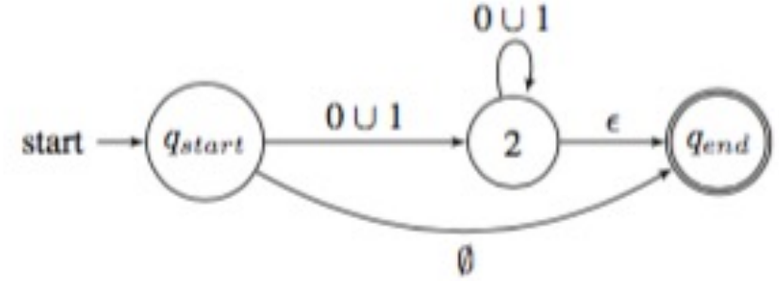


- Ex: Let q_{rip} = state 1:
  - (q_i, q_j) = (q_{start}, state 2) = 0 ∪ 1
  - (q_i, q_j) = (state 2, state 2) = 0 ∪ 1

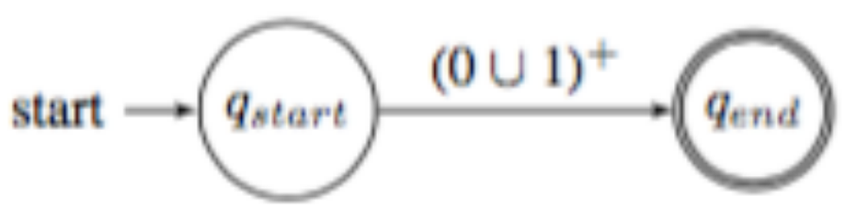


Regular Language

- Lemma 1.60: (“has DFA” ⇒ “has regular expression”)
  - Step 2: Reduce the NFA down to two states – Example
    - Recursive Case: k > 2 states:

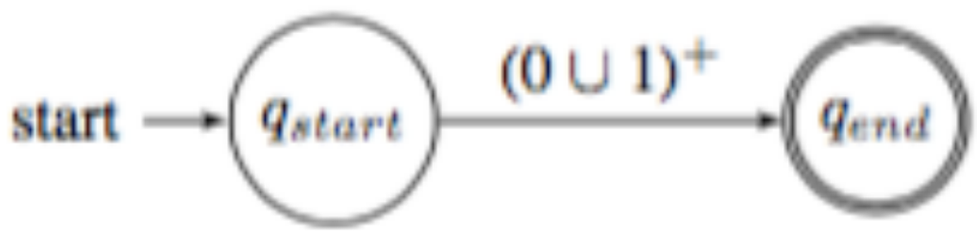


- Ex: Let q_{rip} = state 2, must consider one pair (q_i, q_j):
  - (q_i, q_j) = (q_{start}, q_{end})
    - = (R1 ∘ R2* ∘ R3) ∪ R4
    - = ((0 ∪ 1) ∘ (0 ∪ 1)* ∘ ε) ∪ ∅
    - = (0 ∪ 1)(0 ∪ 1)* = (0 ∪ 1)$^+$



Regular Language

- Lemma 1.60: ("has DFA" ⇒ "has regular expression")
  - Step 2: Reduce the NFA down to two states – Example
    - Recursive Case: k = 2 states:



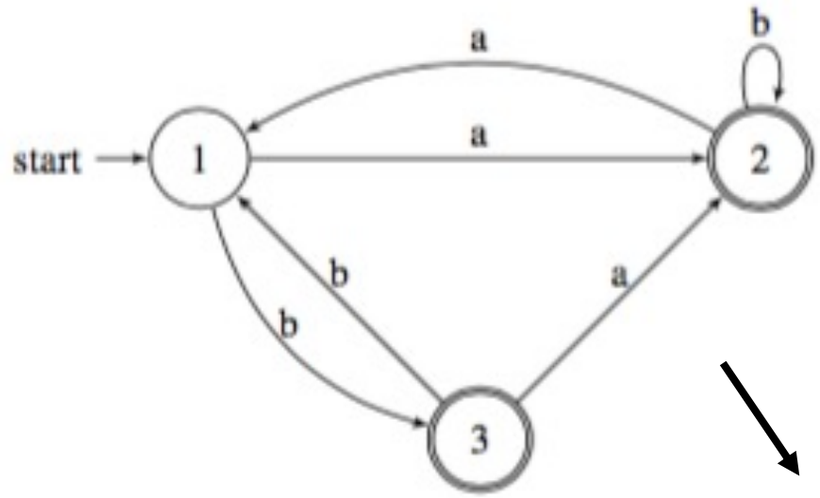
- No other reductions needed, so $(0 \cup 1)^+$ is the regular expression

Regular Language

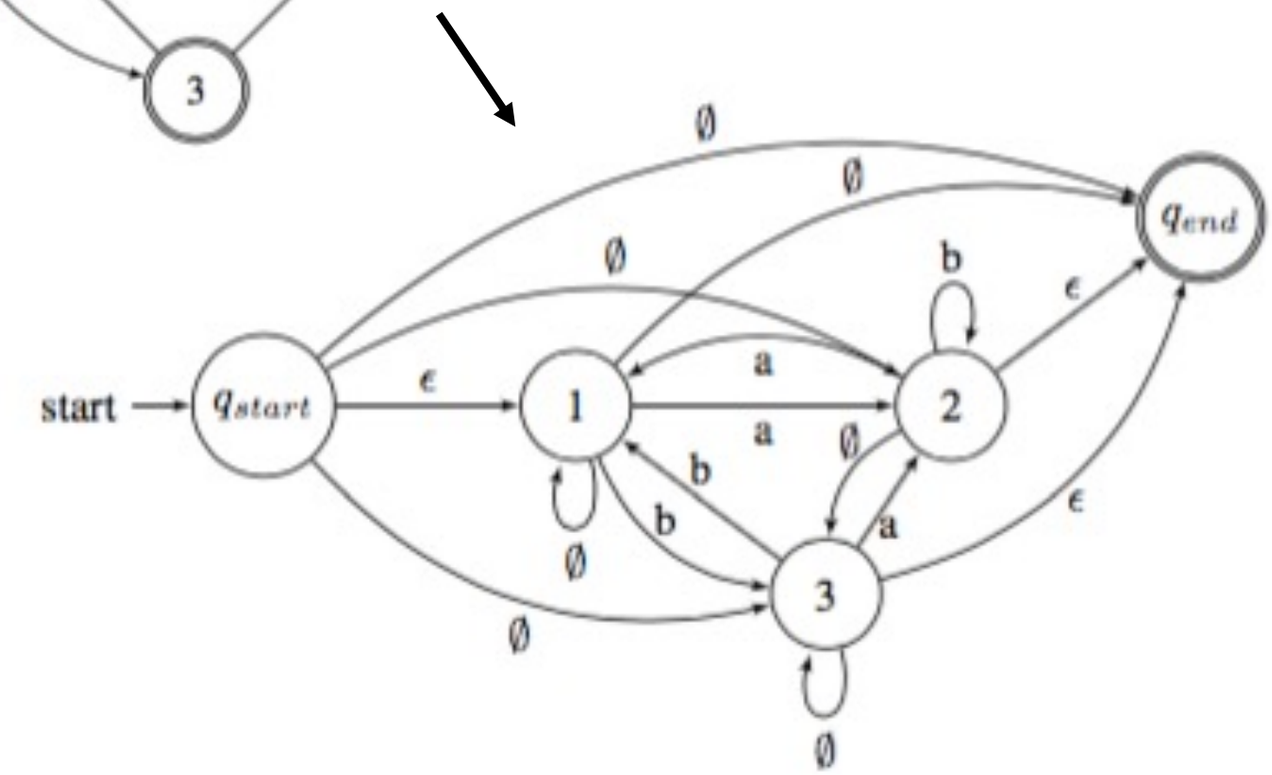
- Theorem 1.54: A language is regular if and only if some regular expression describes it.
  - Forward: Lemma 1.55
    - Claim: If a language L is described by a regular expression, then the language L is regular.
  - Backwards: Lemma 1.60
    - Claim: If a language is regular, then it can be described by a regular expression.
  - We have proved both directions so have proved that a language is regular if and only if some regular expressions describes it.

Regular Language

- Lemma 1.60: Example 2
  - Step 1: Setting up the generalized NFA
    - DFA

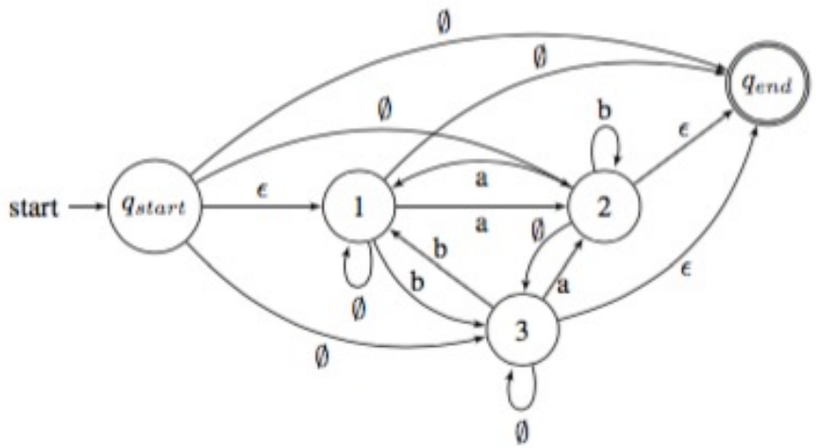


    - NFA



Regular Language

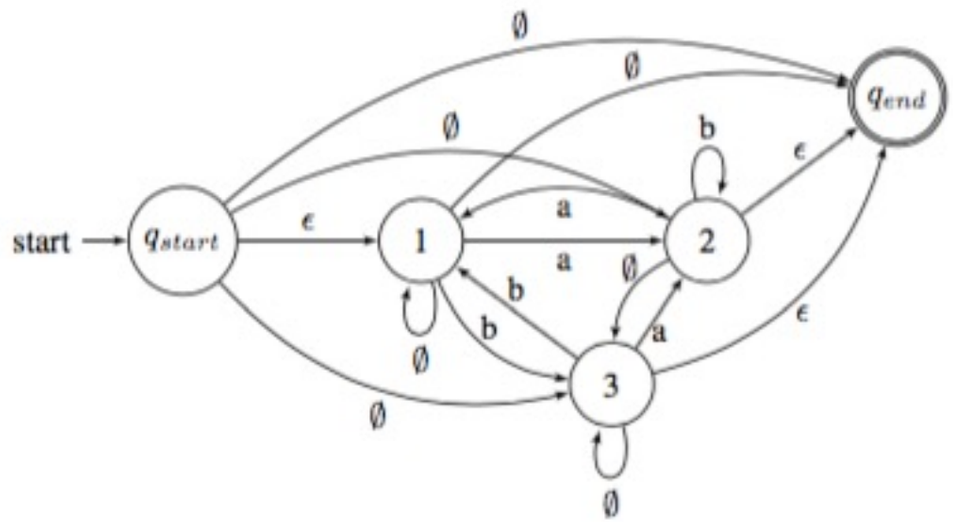
- Lemma 1.60: Example 2
  - Step 2: Reduce the NFA down to two states
    - NFA



- Let q_{rip} = state 1, so have six transitions we need to create:
  - (q_i, q_j) = (q_{start}, state 2), (state 2, state 2), (state 2, state 3), (q_{start}, state 3), (state 3, state 3), (state 3, state 2)

Regular Language

- Lemma 1.60: Example 2
  - Step 2: Reduce the NFA down to two states
    - NFA



$R_1 = \delta(q_i, q_{rip})$

q_i goes to q_{rip}

$R_2 = \delta(q_{rip}, q_{rip})$

q_{rip} goes to itself

$R_3 = \delta(q_{rip}, q_j)$

q_{rip} goes to q_j

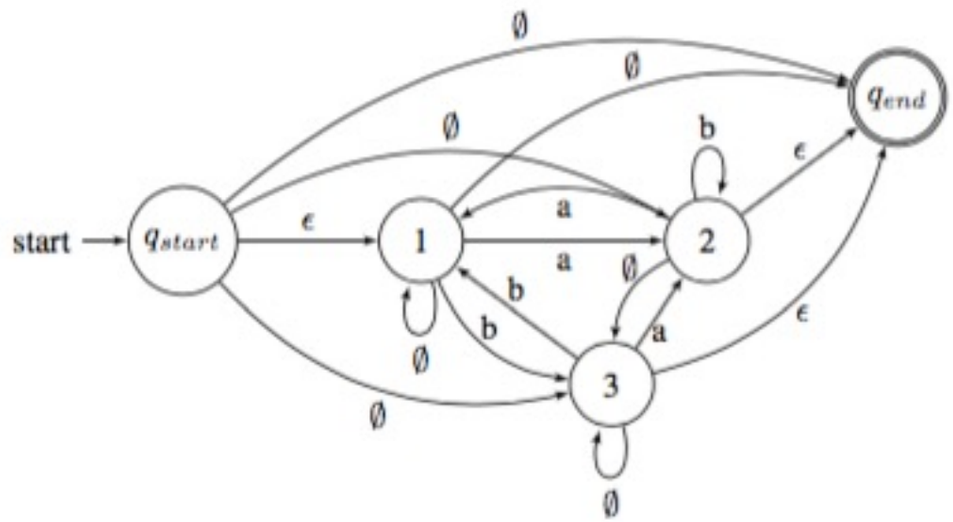
$R_4 = \delta(q_i, q_j)$

q_i goes to q_j

- Let q_{rip} = state 1, (q_{start}, state 2),
  - (R1 ∘ R2* ∘ R3) ∪ R4
  - = (ε ∘ ∅* ∘ a) ∪ ∅ = a

Regular Language

- Lemma 1.60: Example 2
  - Step 2: Reduce the NFA down to two states
    - NFA



$R_1 = \delta(q_i, q_{rip})$

q_i goes to q_{rip}

$R_2 = \delta(q_{rip}, q_{rip})$

q_{rip} goes to itself

$R_3 = \delta(q_{rip}, q_j)$

q_{rip} goes to q_j

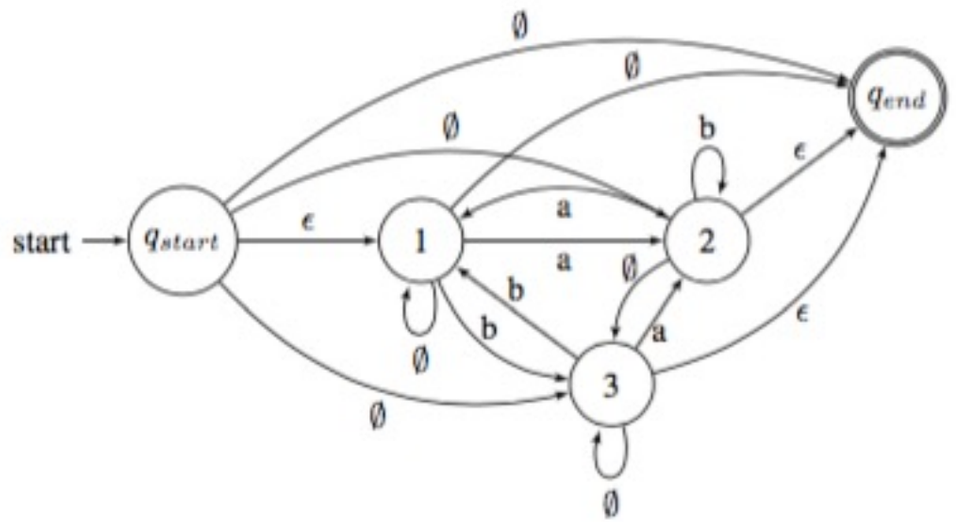
$R_4 = \delta(q_i, q_j)$

q_i goes to q_j

- Let q_{rip} = state 1, (state 2, state 2)
  - (R1 ∘ R2* ∘ R3) ∪ R4
  - = (a ∘ ∅* ∘ a) ∪ b = aa ∪ b

Regular Language

- Lemma 1.60: Example 2
  - Step 2: Reduce the NFA down to two states
    - NFA



$R_1 = \delta(q_i, q_{rip})$

q_i goes to q_{rip}

$R_2 = \delta(q_{rip}, q_{rip})$

q_{rip} goes to itself

$R_3 = \delta(q_{rip}, q_j)$

q_{rip} goes to q_j

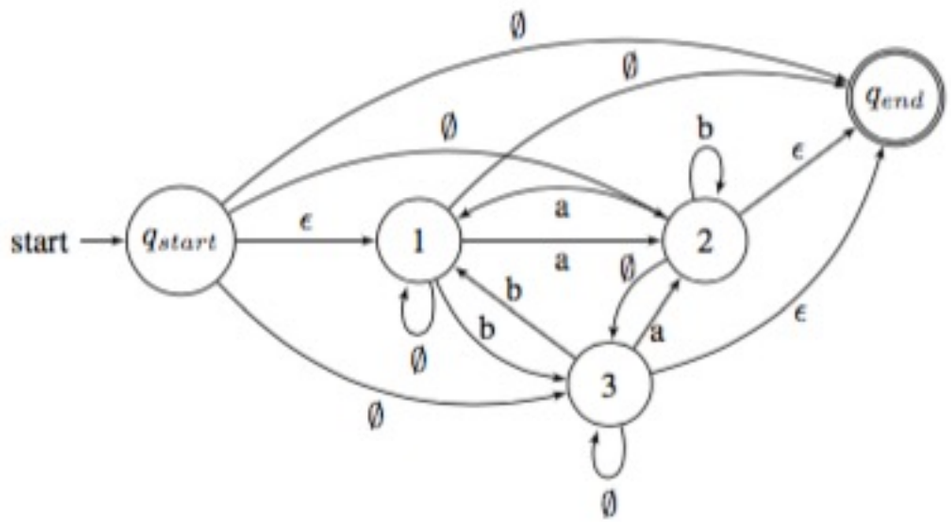
$R_4 = \delta(q_i, q_j)$

q_i goes to q_j

- Let q_{rip} = state 1, so (state 2, state 3)
  - (R1 ∘ R2* ∘ R3) ∪ R4
  - = (a ∘ ∅* ∘ b) ∪ ∅ = ab

Regular Language

- Lemma 1.60: Example 2
  - Step 2: Reduce the NFA down to two states
    - NFA



$R_1 = \delta(q_i, q_{rip})$

q_i goes to q_{rip}

$R_2 = \delta(q_{rip}, q_{rip})$

q_{rip} goes to itself

$R_3 = \delta(q_{rip}, q_j)$

q_{rip} goes to q_j

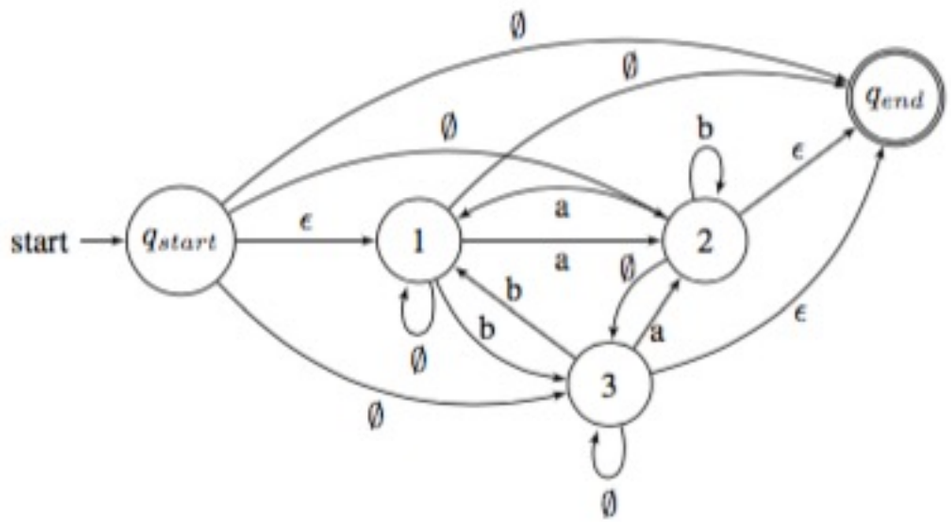
$R_4 = \delta(q_i, q_j)$

q_i goes to q_j

- Let q_{rip} = state 1, (q_{start}, state 3)
  - (R1 ∘ R2* ∘ R3) ∪ R4
  - = (ε ∘ ∅* ∘ b) ∪ ∅ = b

Regular Language

- Lemma 1.60: Example 2
  - Step 2: Reduce the NFA down to two states
    - NFA



$R_1 = \delta(q_i, q_{rip})$

q_i goes to q_{rip}

$R_2 = \delta(q_{rip}, q_{rip})$

q_{rip} goes to itself

$R_3 = \delta(q_{rip}, q_j)$

q_{rip} goes to q_j

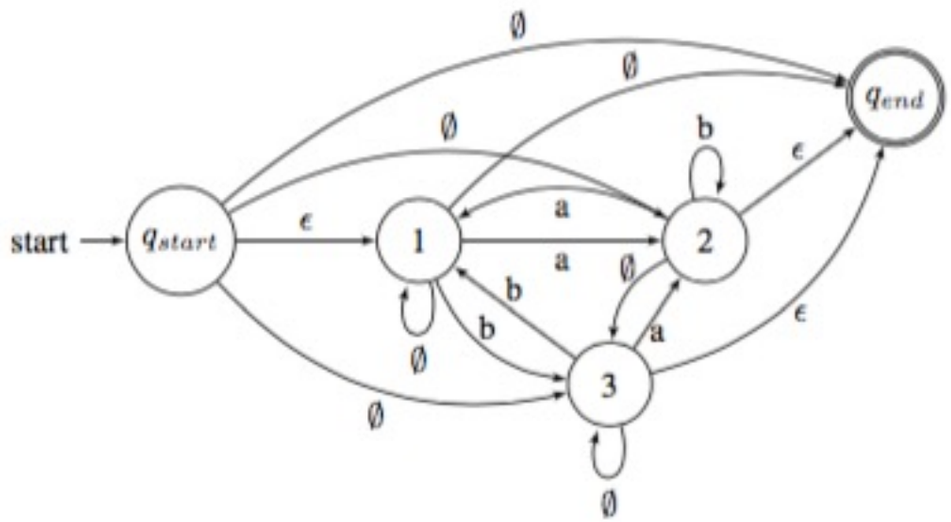
$R_4 = \delta(q_i, q_j)$

q_i goes to q_j

- Let q_{rip} = state 1, (state 3, state 3)
  - (R1 ∘ R2* ∘ R3) ∪ R4
  - = (b ∘ ∅* ∘ b) ∪ ∅ = bb

Regular Language

- Lemma 1.60: Example 2
  - Step 2: Reduce the NFA down to two states
    - NFA



$R_1 = \delta(q_i, q_{rip})$

q_i goes to q_{rip}

$R_2 = \delta(q_{rip}, q_{rip})$

q_{rip} goes to itself

$R_3 = \delta(q_{rip}, q_j)$

q_{rip} goes to q_j

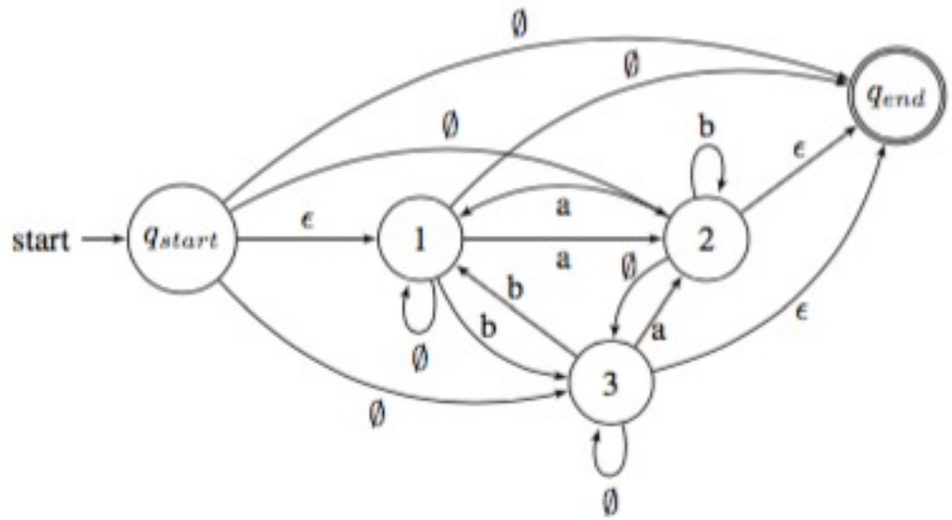
$R_4 = \delta(q_i, q_j)$

q_i goes to q_j

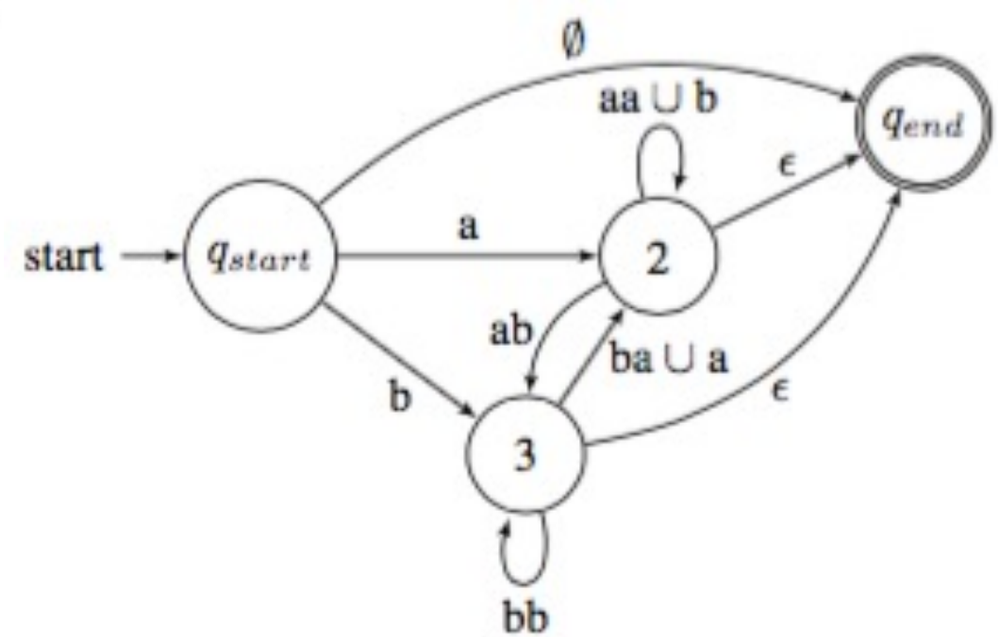
- Let q_{rip} = state 1, (state 3, state 2)
  - (R1 ∘ R2* ∘ R3) ∪ R4
  - = (b ∘ ∅* ∘ a) ∪ a = ba ∪ a

Regular Language

- Lemma 1.60: Example 2
  - Step 2: Reduce the NFA down to two states
    - NFA

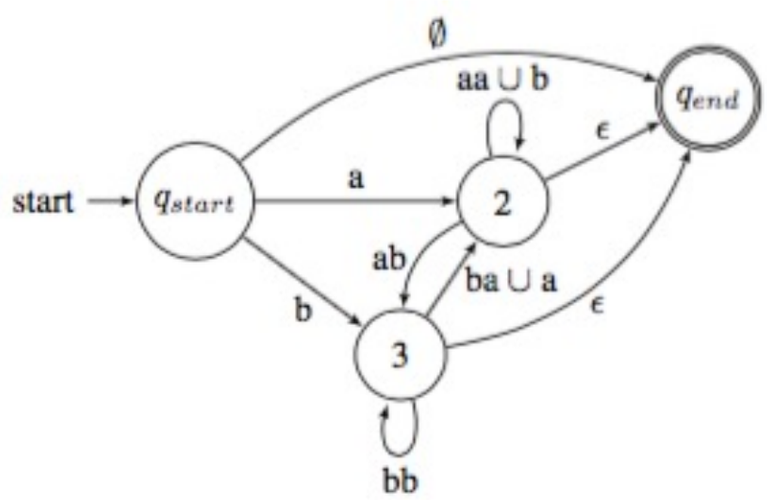


- Let q_{rip} = state 1, so



Regular Language

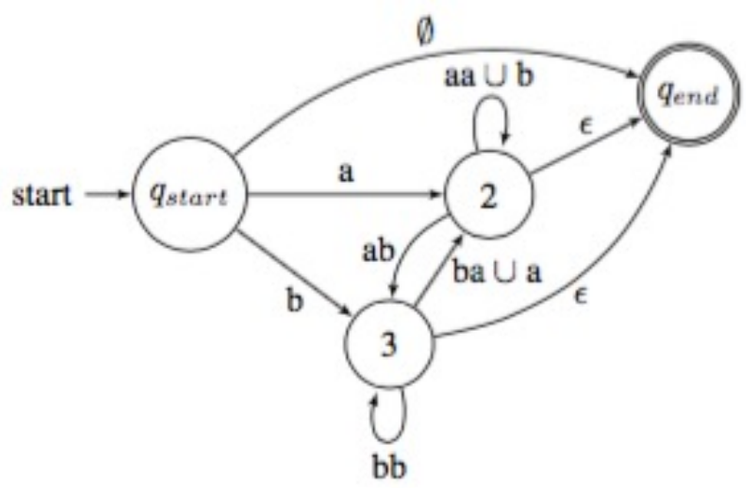
- Lemma 1.60: Example 2
  - Step 2: Reduce the NFA down to two states
  - NFA



  - Let q_{rip} = state 2, so have four transitions we need to create:
    - (q_i, q_j) = (q_{start}, q_{end}), (q_{start}, state 3), (state 3, state 3), (state 3, q_{end})

Regular Language

- Lemma 1.60: Example 2
  - Step 2: Reduce the NFA down to two states
    - NFA

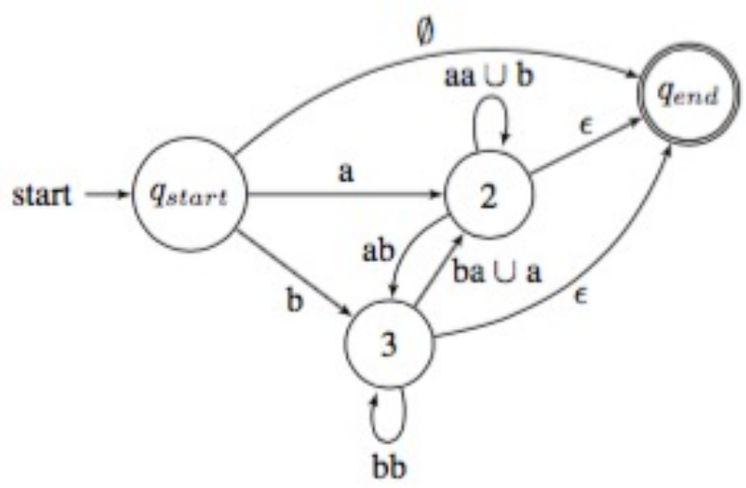


- Let q_{rip} = state 2, (q_{start}, q_{end})
  - (R1 ∘ R2* ∘ R3) ∪ R4
  - = (a ∘ (aa ∪ b)* ∘ ε) ∪ ∅ = a(aa ∪ b)*

$R_1 = \delta(q_i, q_{rip})$
q_i goes to q_{rip}
$R_2 = \delta(q_{rip}, q_{rip})$
q_{rip} goes to itself
$R_3 = \delta(q_{rip}, q_j)$
q_{rip} goes to q_j
$R_4 = \delta(q_i, q_j)$
q_i goes to q_j

Regular Language

- Lemma 1.60: Example 2
  - Step 2: Reduce the NFA down to two states
    - NFA



- Let q_{rip} = state 2, (q_{start}, state 3)
  - (R1 ∘ R2* ∘ R3) ∪ R4
  - = (a ∘ (aa ∪ b)* ∘ ab) ∪ b
  - = (a (aa ∪ b)* ab) ∪ b

$R_1 = \delta(q_i, q_{rip})$

q_i goes to q_{rip}

$R_2 = \delta(q_{rip}, q_{rip})$

q_{rip} goes to itself

$R_3 = \delta(q_{rip}, q_j)$

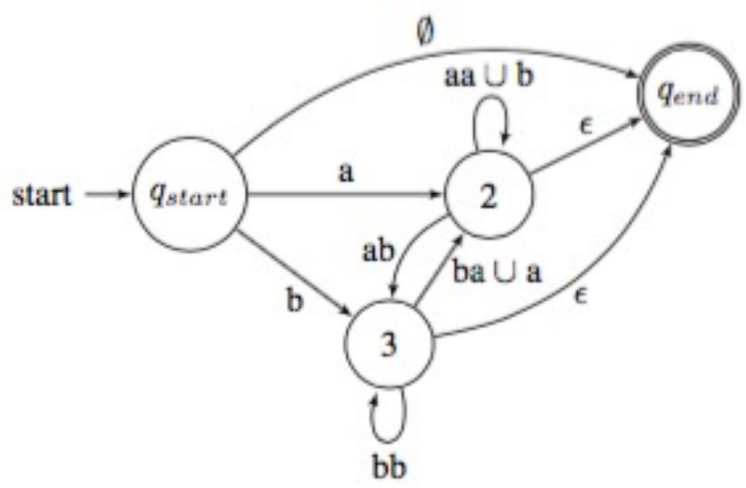
q_{rip} goes to q_j

$R_4 = \delta(q_i, q_j)$

q_i goes to q_j

Regular Language

- Lemma 1.60: Example 2
  - Step 2: Reduce the NFA down to two states
    - NFA



- Let q_{rip} = state 2, (state 3, state 3)
  - (R1 ∘ R2* ∘ R3) ∪ R4
  - = ((ba ∪ a) ∘ (aa ∪ b)* ∘ ab) ∪ bb
  - = ((ba ∪ a)(aa ∪ b)* ab) ∪ bb

$R_1 = \delta(q_i, q_{rip})$

q_i goes to q_{rip}

$R_2 = \delta(q_{rip}, q_{rip})$

q_{rip} goes to itself

$R_3 = \delta(q_{rip}, q_j)$

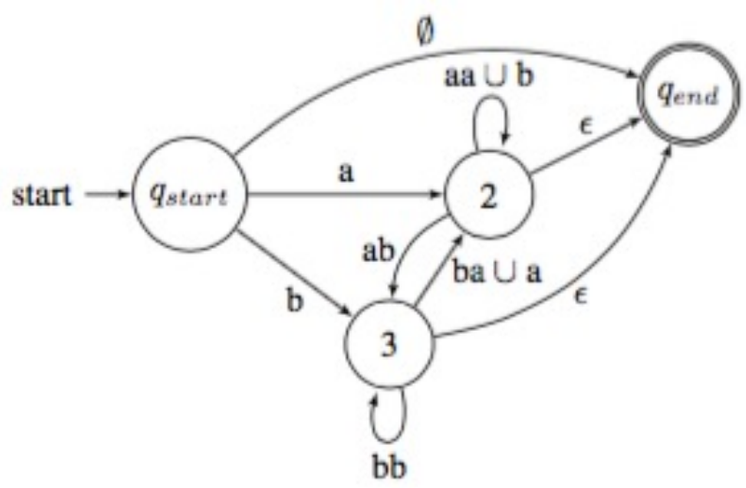
q_{rip} goes to q_j

$R_4 = \delta(q_i, q_j)$

q_i goes to q_j

Regular Language

- Lemma 1.60: Example 2
  - Step 2: Reduce the NFA down to two states
    - NFA



- Let q_{rip} = state 2, (state 3, q_{end})
  - (R1 ∘ R2* ∘ R3) ∪ R4
  - = ((ba ∪ a) ∘ (aa ∪ b)* ∘ ε) ∪ ε
  - = (ba ∪ a)(aa ∪ b)* ∪ ε

$R_1 = \delta(q_i, q_{rip})$

q_i goes to q_{rip}

$R_2 = \delta(q_{rip}, q_{rip})$

q_{rip} goes to itself

$R_3 = \delta(q_{rip}, q_j)$

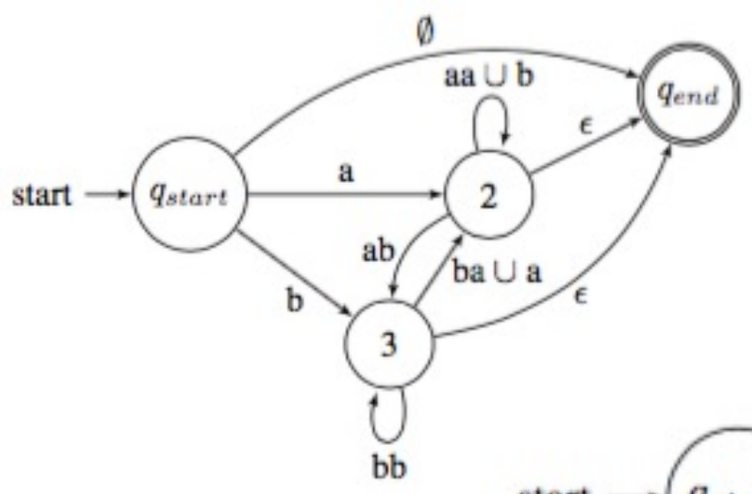
q_{rip} goes to q_j

$R_4 = \delta(q_i, q_j)$

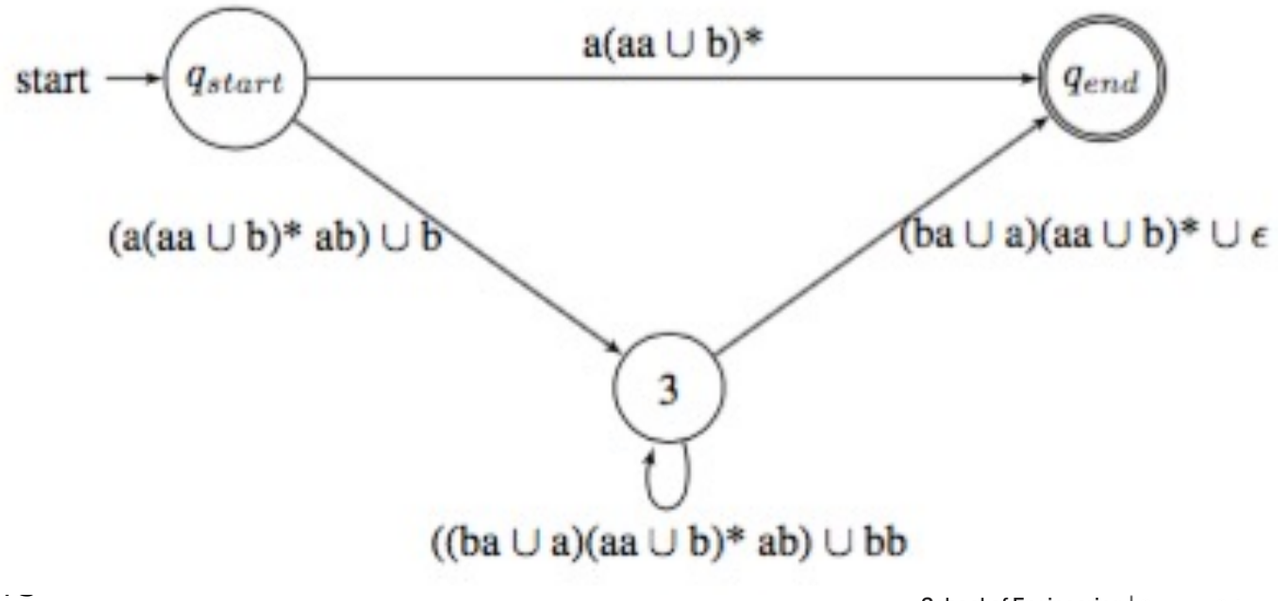
q_i goes to q_j

Regular Language

- Lemma 1.60: Example 2
  - Step 2: Reduce the NFA down to two states
    - NFA

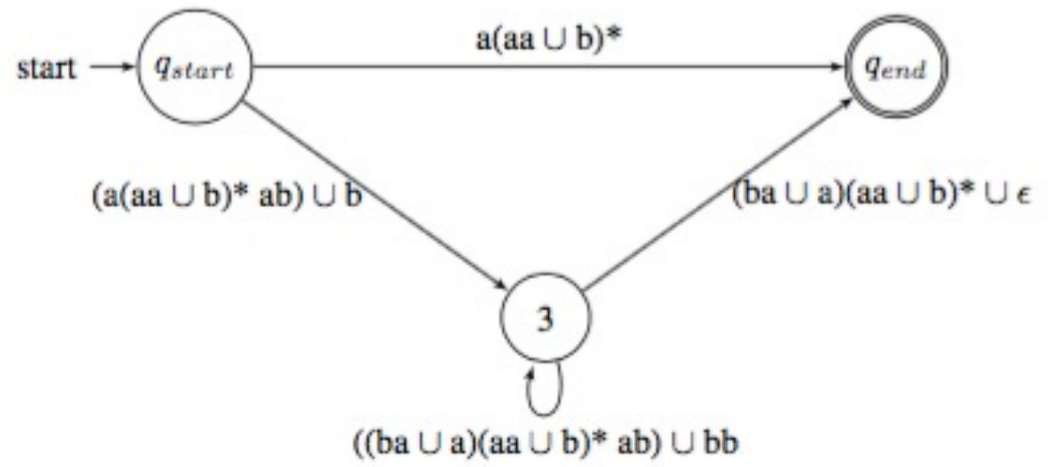


- Let q_{rip} = state 2, so



Regular Language

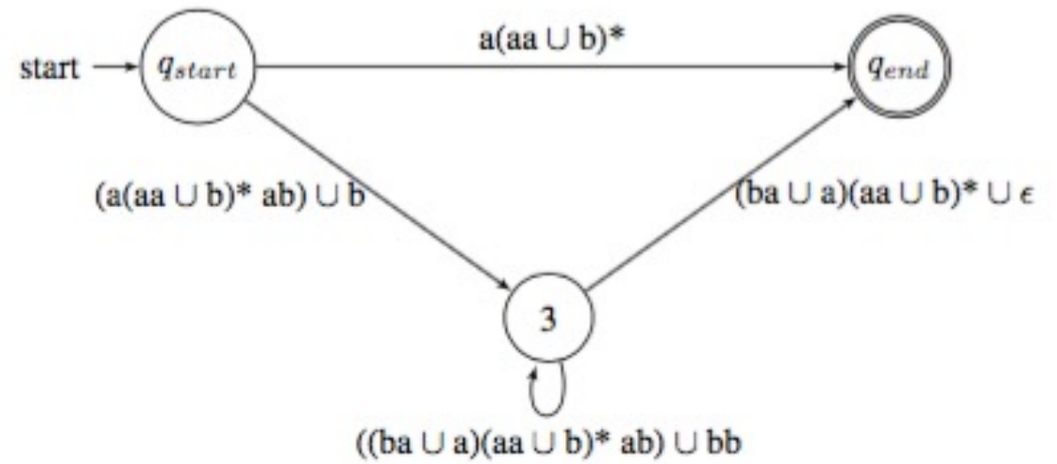
- Lemma 1.60: Example 2
  - Step 2: Reduce the NFA down to two states
    - NFA



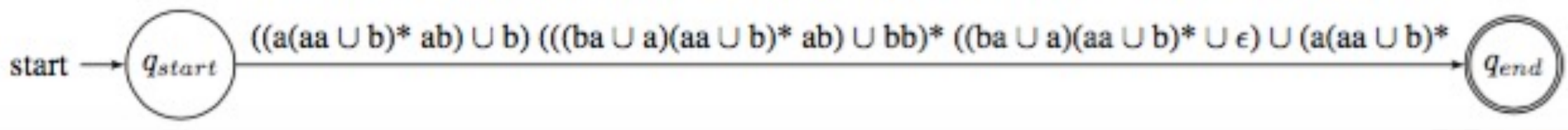
- Let q_{rip} = state 3, so only 1 transition: (q_{start}, q_{end})
  - (R1 ∘ R2* ∘ R3) ∪ R4
  - = ((a(aa ∪ b)* ab) ∪ b) ∘ (((ba ∪ a)(aa ∪ b)* ab) ∪ bb)* ∘ ((ba ∪ a)(aa ∪ b)* ∪ ε) ∪ (a(aa ∪ b)*)

Regular Language

- Lemma 1.60: Example 2
  - Step 2: Reduce the NFA down to two states
    - NFA



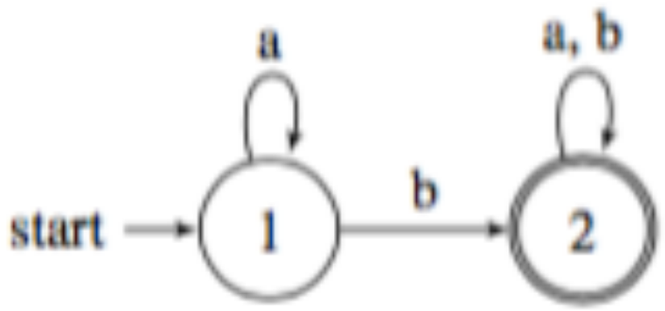
- k = 2, so R = ((a(aa ∪ b)* ab) ∪ b) ∘ (((ba ∪ a)(aa ∪ b)* ab) ∪ bb)* ∘ ((ba ∪ a)(aa ∪ b)* ∪ ε) ∪ (a(aa ∪ b)*)



Try It

1. Construct an NFA from the Regular Expression ((00)*11) ∪ 01

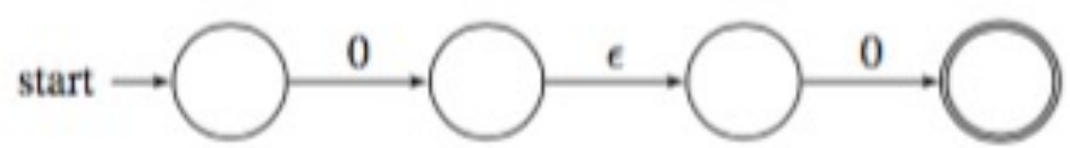
2. Convert the DFA below into a Regular Expression.



Try It

1. Construct an NFA from the Regular Expression ((00)*11) ∪ 01

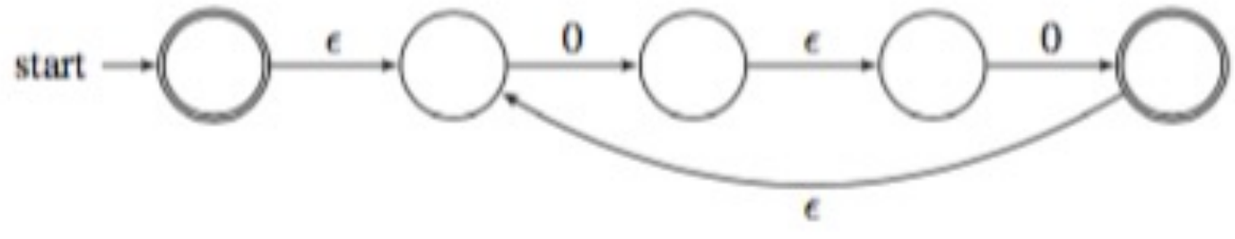
- 00



- 11

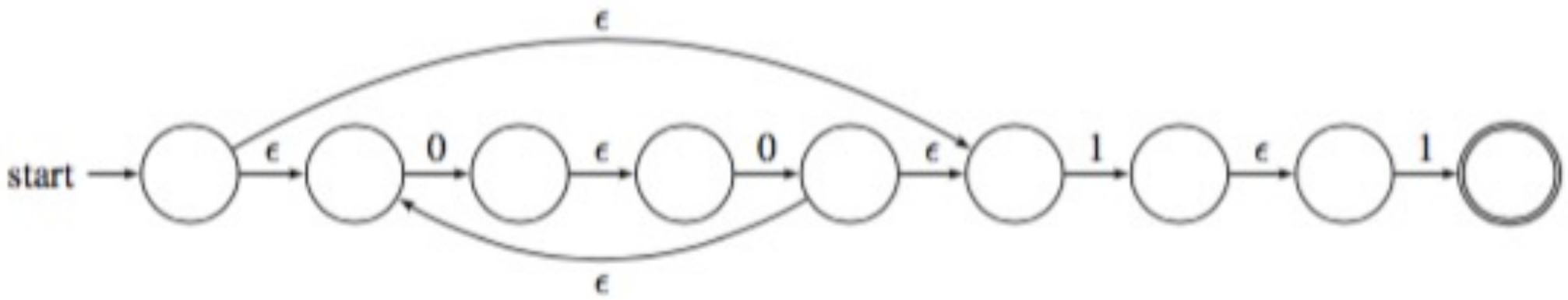
- 01

- (00)*



Try It

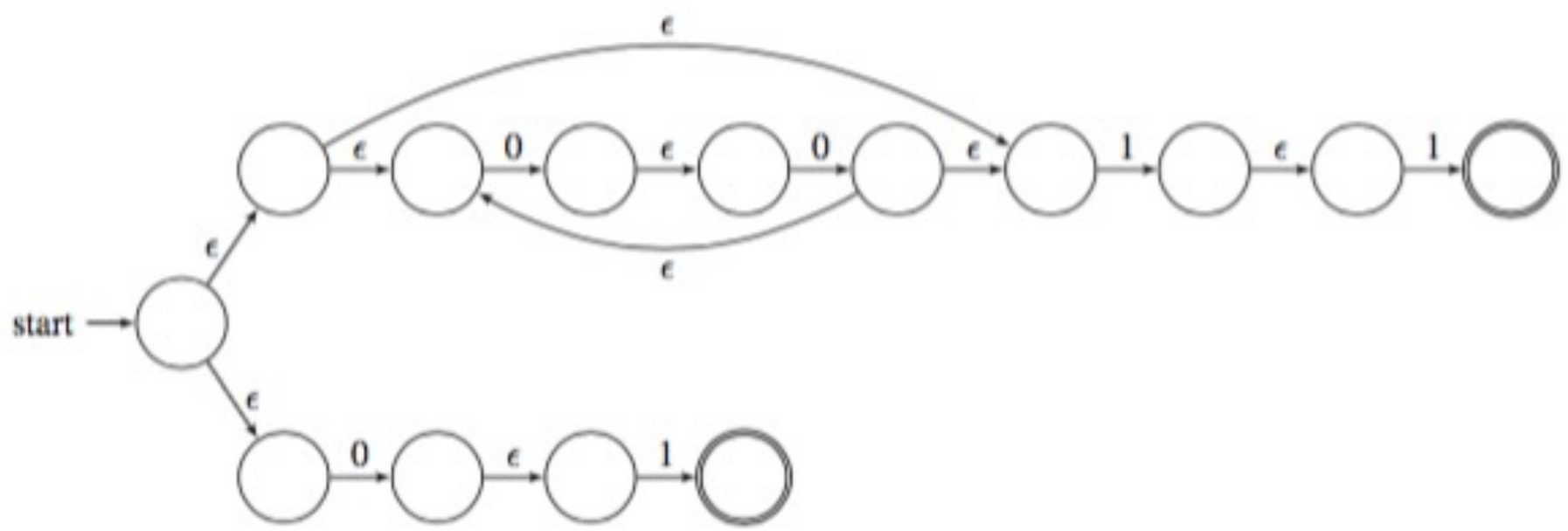
1. Construct an NFA from the Regular Expression ((00)*11) ∪ 01, cont.
   - (00)*11



Try It

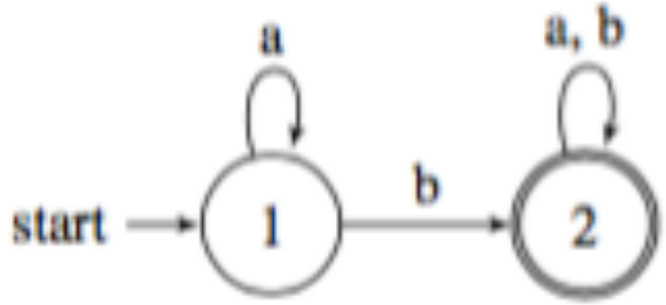
1. Construct an NFA from the Regular Expression ((00)*11) ∪ 01, cont.

- ((00)*11) ∪ 01

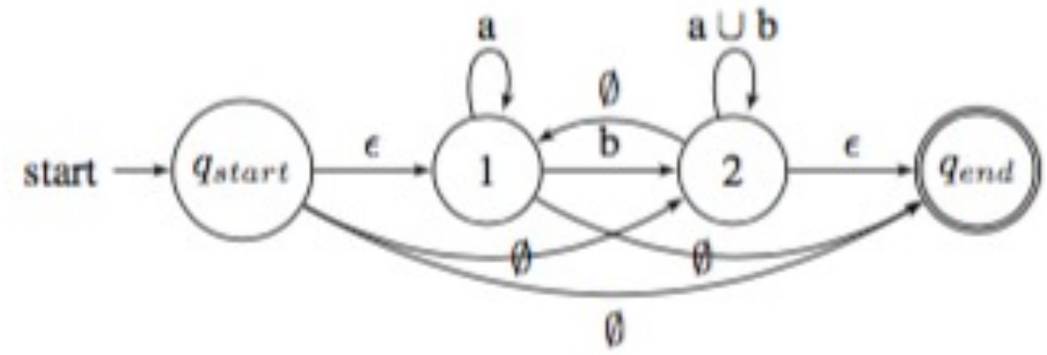


Try It

2. Convert the DFA below into a Regular Expression.

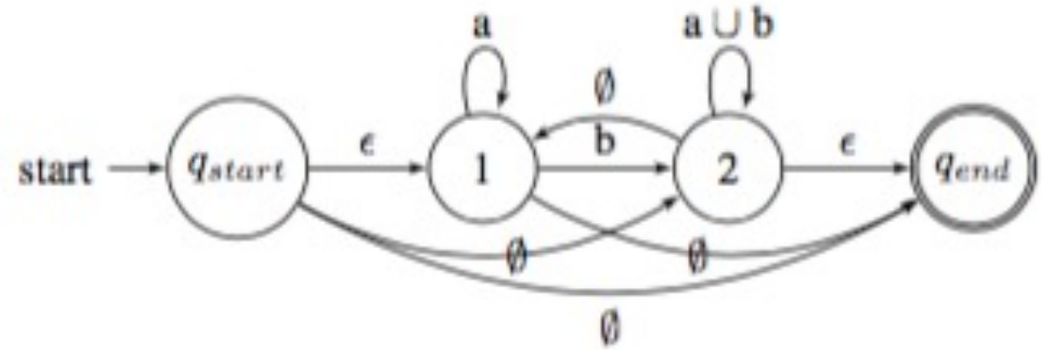


- Set up the generalized NFA



Try It

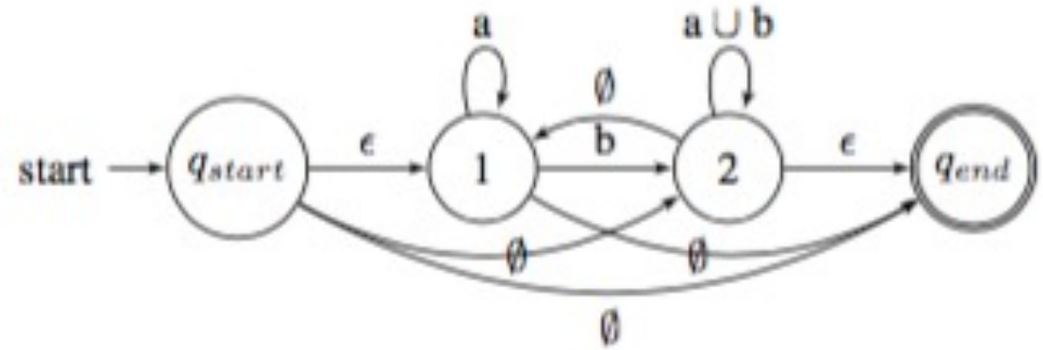
2. Convert the NFA below into a Regular Expression.



- Let q_{rip} = state 1, so have two transitions:
  - (q_i, q_j) = (q_{start}, state 2), (state 2, state 2)

Try It

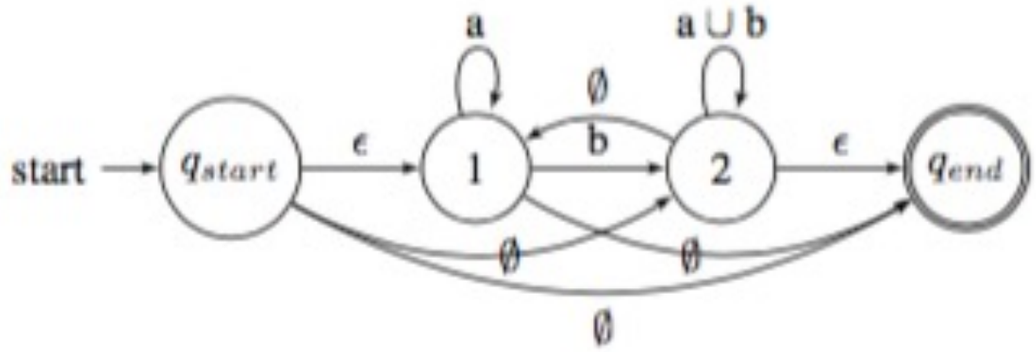
2. Convert the NFA below into a Regular Expression.



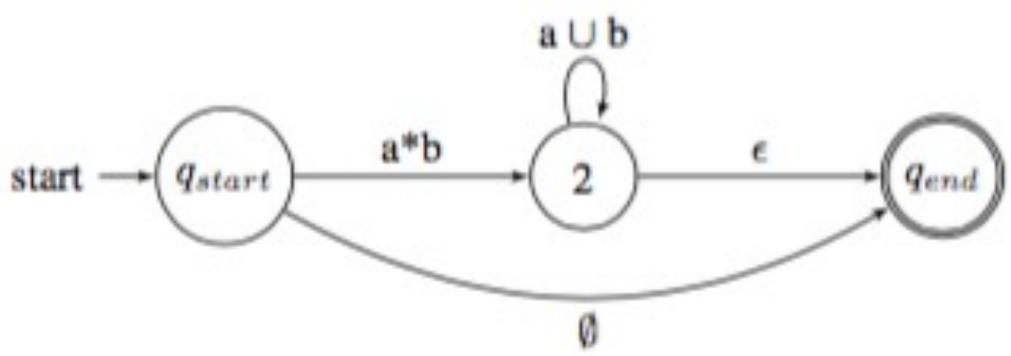
- Let q_{rip} = state 1, (q_{start}, state 2)
  - (R1 ∘ R2* ∘ R3) ∪ R4
  - = (ε ∘ a* ∘ b) ∪ ∅ = a*b
- Let q_{rip} = state 1, (state 2, state 2)
  - (R1 ∘ R2* ∘ R3) ∪ R4
  - = (∅ ∘ a* ∘ b) ∪ (a ∪ b) = a ∪ b

Try It

2. Convert the NFA below into a Regular Expression.

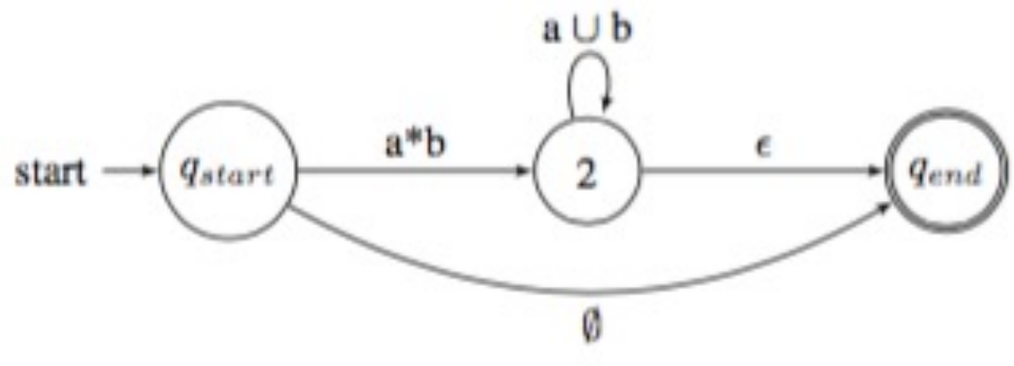


- Let q_{rip} = state 1, so get:

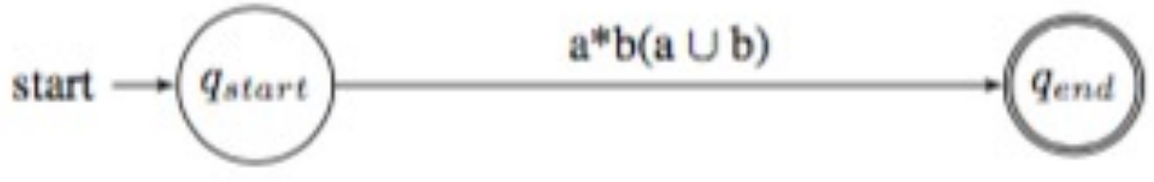


Try It

2. Convert the NFA below into a Regular Expression.



- Let q_{rip} = state 2, $(q_i, q_j) = (q_{start}, q_{end})$
  - $(R1 \circ R2^* \circ R3) \cup R4$
  - $= (a^*b \circ (a \cup b)^* \circ \varepsilon) \cup \emptyset = \mathbf{a^*b(a \cup b)^*} = R$



Theory of Computation
Chapter 1

Pumping Lemma to Prove Non-regular Languages

School of Engineering | Computer Science

Linus Torvalds

- Started creation of Linux operating system core (1991); still coordinates new versions
- Linux is a free, open-source operating system used worldwide
- Since then, thousands have contributed
- In 2005 began development on Git, version control software

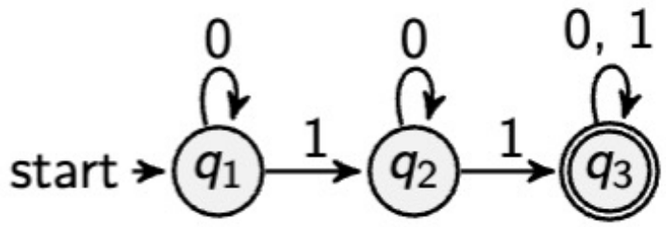
Proving a Languages is Not Regular

- Sometimes we would like to easily prove that a language is not part of the set of regular languages
- We can use the pumping lemma to prove a language is not part of this set
- Here is how it works:
  - If your parents gave you a bicycle and tell you it is a motorbike, how can you prove it is not a motorbike?
  - You must think of a feature that a motorbike has that a bicycle does not, e.g. a motor

Proving Languages to be Non-regular

- Fact: Every regular language has some property, P, such that if a language L does not have this property, then L is not regular
- Contrapositive: (A ⇒ B) → (!B ⇒ !A)
    - If you have condition A, then condition B is true as well
    - If you do not have condition B, then you also do not have condition A
- What property P does every regular language have?

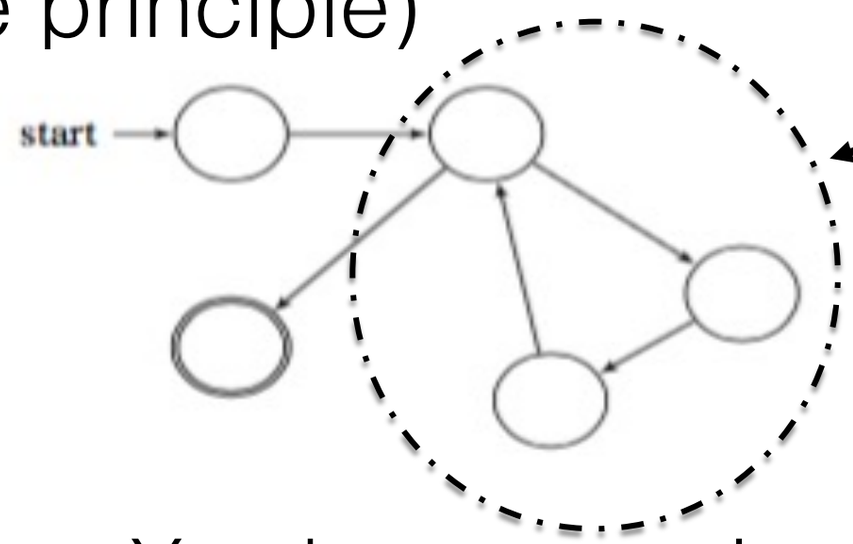
Property P – Pigeonhole Principle

- Observation: Let M be a DFA with k states. Let $w \in \Sigma^*$ be an input string of length $|w| \geq k$
- Conclusion 1: There exists a state q that has been visited at least twice when running M on w (pigeonhole principle)
- Ex: 

$k = 3, w = 011, |w| = 3$ move through the DFA from $q_1 -> q_1 -> q_2 -> q_3$, travelled to state q_1 two times
- No matter what the input string is, you will <u>always</u> travel to one state at least twice with input the size of the number of states or larger if it is a regular language.

Property P – Pigeonhole Principle

- Observation: Let M be a DFA with k states. Let $w \in \Sigma^*$ be an input string of length $|w| \geq k$
- Conclusion 1: There exists a state q that has been visited at least twice when running M on w (pigeonhole principle)
- Generic:



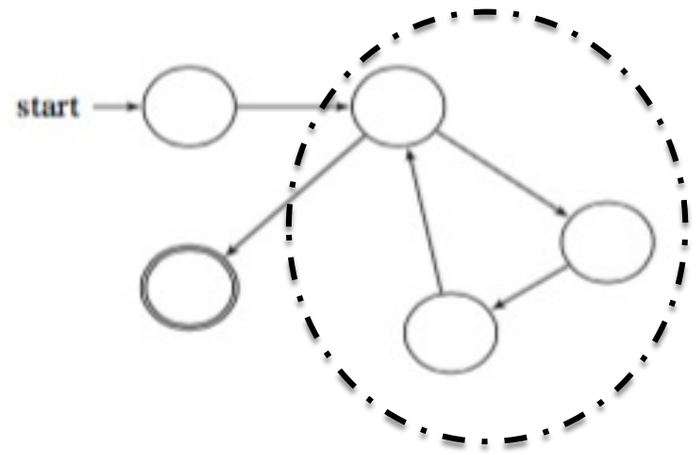
- Pigeonhole – You have more input characters in your string than you have states to place them into

Property P – Pigeonhole Principle

- Observation: Let M be a DFA with k states. Let $w \in \Sigma^*$ be an input string of length $|w| \geq k$
- Conclusion 2: The substring of w read while following a cycle C can be repeated as many times as needed and the resulting string will still be accepted
- Ex: $w = \underbrace{\ldots\ldots}_{x}\underbrace{\ldots\ldots}_{y}\underbrace{\ldots\ldots}_{z}$ is accepted, so

$w' = \underbrace{\ldots\ldots}_{x}\underbrace{(\ldots\ldots)^*}_{y^*}\underbrace{\ldots\ldots}_{z}$ is accepted

Cycle C



Pumping Lemma

- Theorem 1.70: Let A be a regular language. There exists a "pumping length" $p > 0$, such that for all strings $s \in$ A of length $|s| \geq p$, there exists $x, y, z \in \Sigma^*$ such that $s = xyz$ and:

1. For all $i \geq 0, xy^iz \in A$ $\in$ A means accepted by A

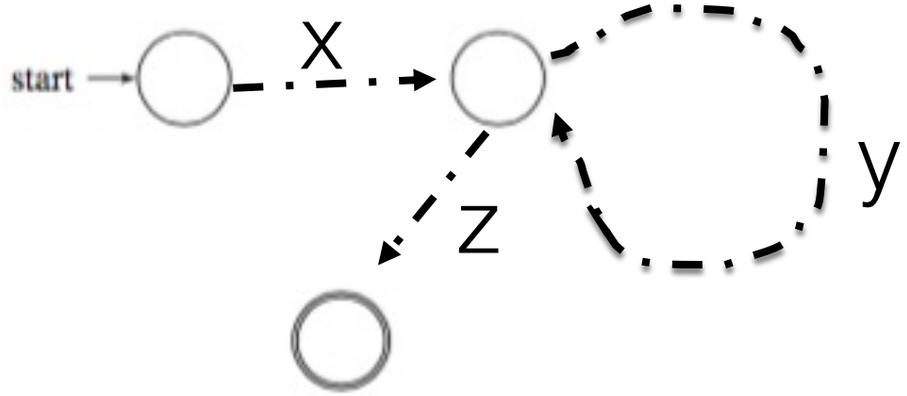
2. $|y| > 0$ y can't be empty, but x, z can be
3. $|xy| \leq p$ xy is the concatenation of x and y

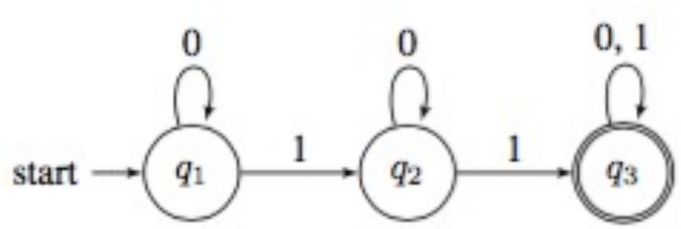
Pumping Lemma

- Theorem 1.70 Proof:
  - Let M = (Q, Σ, δ, q_1, F) be a DFA recognizing A with k states.
  - Let $s = s_1 s_2 \ldots s_n \in A$ such that $|s| = n \geq k$, suppose M enters states $r_1, r_2, \ldots, r_{n+1}$ when processing s, by the pigeonhole principle, there exists $i < j$ such that $r_i = r_j$
  - Pick the smallest such j. Note that $j \leq k + 1$, since M has only k states and at least one state was repeated.

Pumping Lemma

- Theorem 1.70 Proof:
  - Therefore, let $x = s_1 s_2 \ldots s_{i\text{-}1} \mid y = s_i \ldots s_{j\text{-}1} \mid z = s_j \ldots s_n$
  - Let p (pumping length) = k (# of states)





Ex: 101 = q_1 –> q_2 -> q_2 -> q_3

1. Since x takes M from r_1 to r_i, y takes M from r_i to r_j, and z takes M from r_j to r_{n+1}, M accepts $xy^l z$ for all $l \geq 0$
2. Since $i \neq j, |y| > 0$
3. Since $j \leq k + 1, |xy| \leq k = p$

Pumping Lemma

- Example 1.73: Let $B = \{0^n 1^n \mid n \geq 0\}$
  - Claim: B is not regular
  - Proof by contradiction:

Pumping Lemma

- Example 1.73: $Let\ B\ =\ \{0^n1^n \mid n\ \geq\ 0\}$
  - Claim: B is not regular
  - Proof by contradiction:
    - Assume B is regular, so there exists a pumping length $p\ >\ 0$, pick string $s\ =\ 0^p1^p$
    - preconditions: $s\ \in\ B$ and $|s|\ =\ 2p\ \geq\ p$
  - There exists $s\ =\ xyz$ and by:
    3. $|xy|\ \leq\ p$ (x and y contain only 0's)
    2. $|y|\ >\ 0$ (y contains at least one 0)
    1. $xy^2z\ =\ xyyz\ =\ 0^{n'}1^n$ where the result has more 0s than 1s, $n'\ >\ n$, which is a contradiction ($xyz\ \in\ B$, but $xy^2z\ \notin\ B$)

       Or xyz, when $p\ =\ 2$ is 0 0 11, so when y^2 we get 0 00 11 $\notin$ B
       x y z x y^2 z

Pumping Lemma

- Example 1.75: Let $C = \{ww \mid w \in \{0,1\}*\}$
  - Claim: C is not regular
  - Proof by contradiction:

Pumping Lemma

- Example 1.75: Let $C = \{ww \mid w \in \{0,1\}*\}$
  - Claim: C is not regular
  - Proof by contradiction:
    - Assume C is regular, therefore there exists a pumping length $p > 0$, pick string $s = 0^p10^p1$
    - Preconditions: $s \in C$ and $|s| = 2p + 2 \geq p$
  - There exists $s = xyz$ and by:
    3. $|xy| \leq p$ (x and y contain only 0's)
    2. $|y| > 0$ (y contains at least one 0)
    1. $xy^2z = xyyz = 0^{p'}10^p1$, for $p' > p$, so $xyz \in C$, but $xy^2z \notin C$

       Or xyz, $p = 2$ is 0 0 1001, so when y^2 we get 0 00 1001 $\notin C$
       x y z x y^2 z

Pumping Lemma

- Example 1.77: Let $B = \{0^i 1^j \mid i > j\}$
  - Claim: B is not regular
  - Proof by contradiction:

Pumping Lemma

- Example 1.77: $Let\ B = \{0^i 1^j \mid i > j\}$
  - Claim: B is not regular
  - Proof by contradiction:
    - Assume B is regular, therefore there exists a pumping length $p > 0$, pick string $s = 0^{p+1}1^p$
    - Preconditions: $s \in B$ and $|s| = 2p + 1 \geq p$
  - There exists $s = xyz$ and by:
    3. $|xy| \leq p$ (x and y contain only 0's)
    2. $|y| > 0$ (y contains at least one 0)
    1. $xy^2z = xyyz = 0^{p+2}1^p$, which is still in the language

       So, use $xy^0z = xz = 0^{p'}1^p$, for $p' \leq p$, so $i \not> j$ and $xy^0z \notin B$

Remember that Case 1 should hold for for all $i \geq 0$, $xy^iz \in B$

Try It

1. Use the Pumping Lemma to show that D = $\{0^n1^m0^n \mid m, n \geq 0\}$ is not a regular language

2. Construct an NFA from the Regular Expression ((00)*11) ∪ 01

Try It

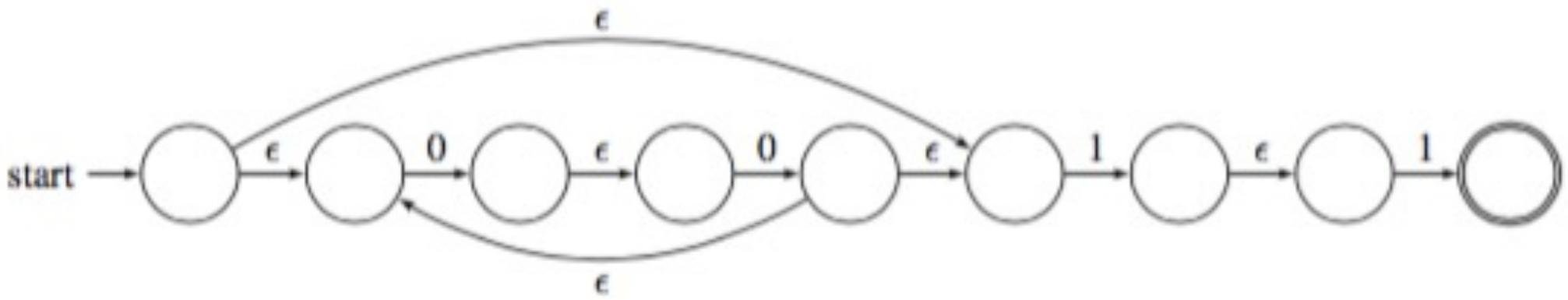
- Use the Pumping Lemma to show that $D = \{0^n 1^m 0^n \mid m, n \geq 0\}$ is not a regular language
  - Claim: D is not regular
  - Proof by contradiction:
    - Assume D is regular, therefore there exists a pumping length $p > 0$, pick string $s = 0^p 1^{p+1} 0^p$
      - preconditions: $s \in B$ and $|s| = 3p + 1 \geq p$
  - There exists $s = xyz$ and by:
    3. $|xy| \leq p$ (x and y contain only 0's)
    2. $|y| > 0$ (y contains at least one 0)
    1. $xy^2z = xyyz = 0^{n'} 1^{n+1} 0^n$ where the result has more 0s on one side than the 0's on the other side, $n' > n$, which is a contradiction ($xyz \in B, but\ xy^2z \notin B$)

       Or $xyz, p = 2$ is 0 0 11100, so when y^2 we get 0 00 11100 $\notin B$

       x y z x y^2 z

Try It

1. Construct an NFA from the Regular Expression ((00)*11) ∪ 01, cont.
   - (00)*11



Try It

1. Construct an NFA from the Regular Expression ((00)*11) ∪ 01, cont.
   - ((00)*11) ∪ 01