Theory of Computation
Chapter 1

Regular Languages Part 3

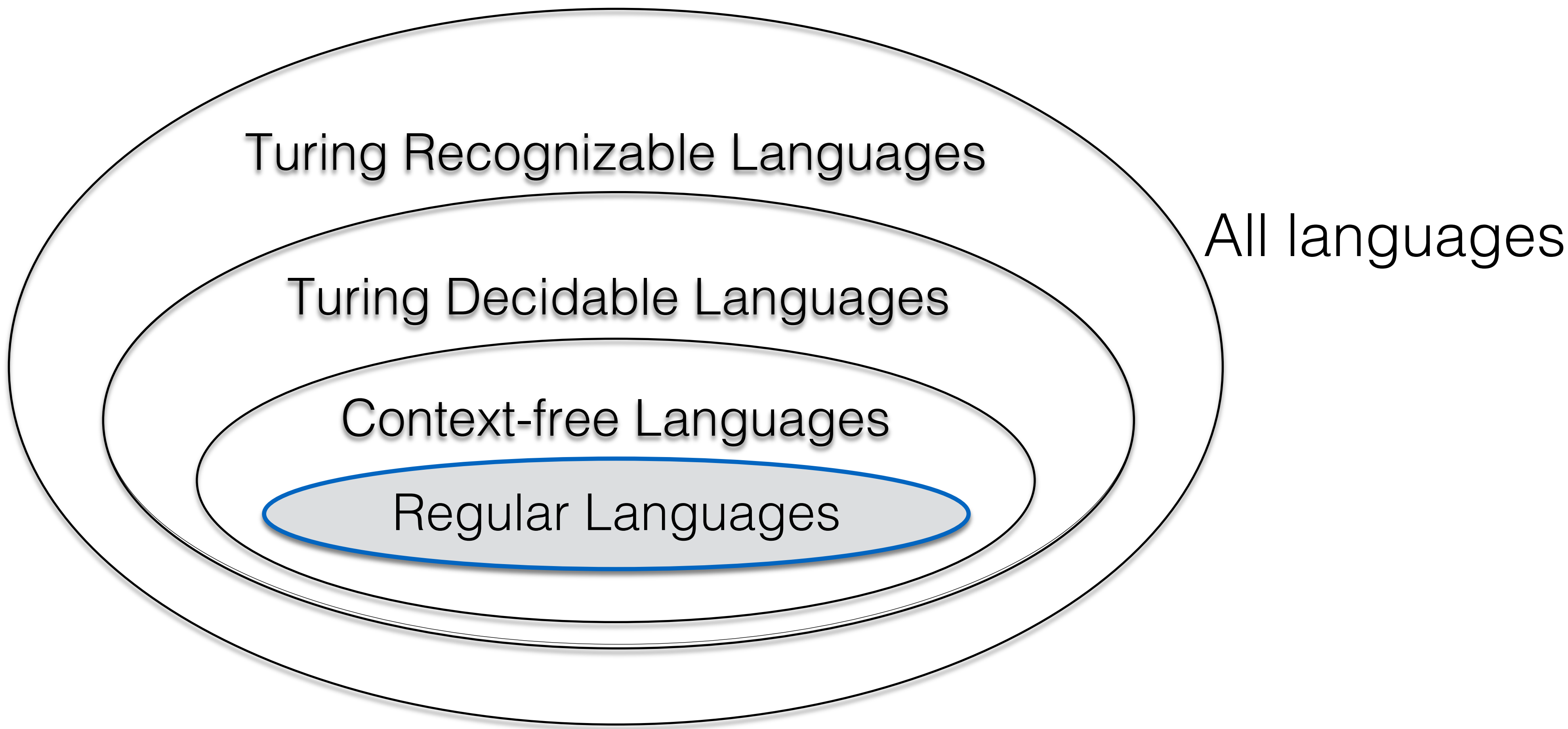
VCU

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, ∪, concatenation, ∘, and star, ∗.

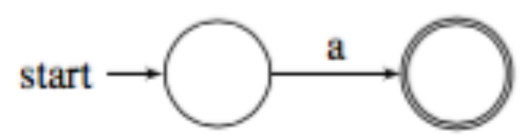
Regular Language

- 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 $L(R) = L(N)$

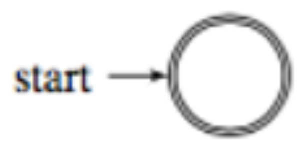
Regular Language, Forward

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

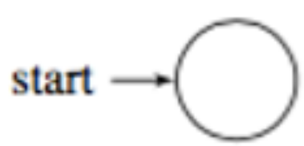
1. R = a for some $a \in \Sigma$



2. R = ε



3. R = $\emptyset$

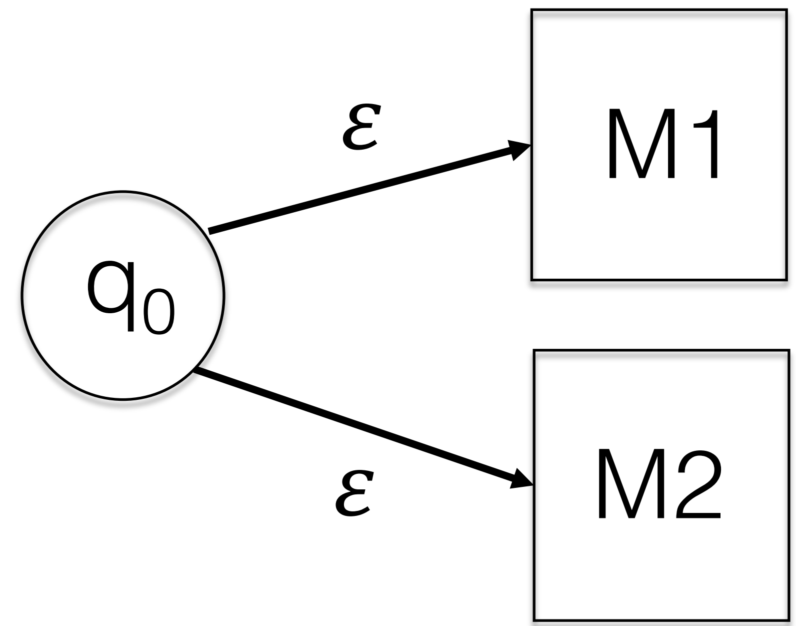


4. R = $R_1 \cup R_2$
5. R = $R_1 \circ R_2$
6. R = R_1^*

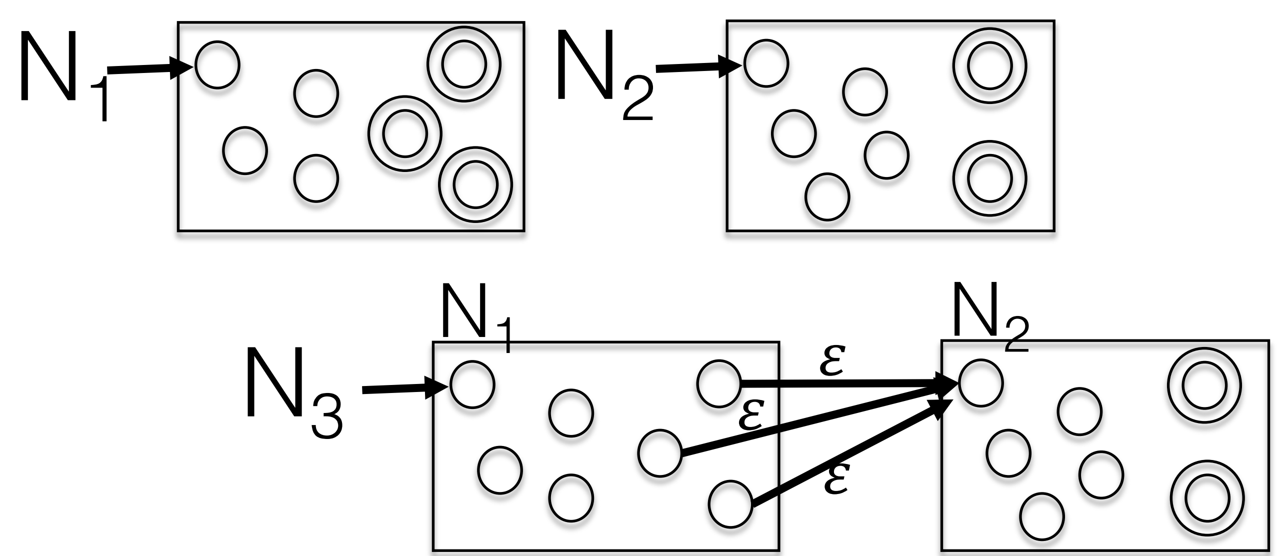
For these cases, we build the NFAs as we did in previous slides.

Regular Language Operations

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

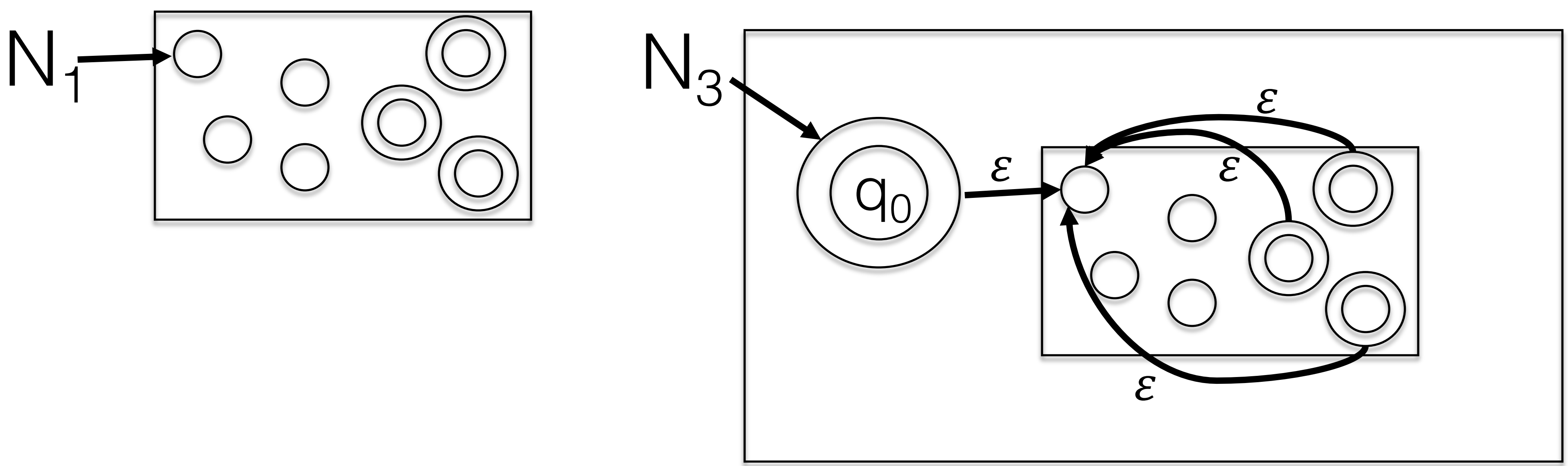


  - Concatenation: $R = R_1 \circ R_2$



Regular Language Operations

- <u>Remember</u>:
  - <u>Star</u>: $R = R_1{}^*$



School of Engineering | Computer Science

Regular Language, Forward

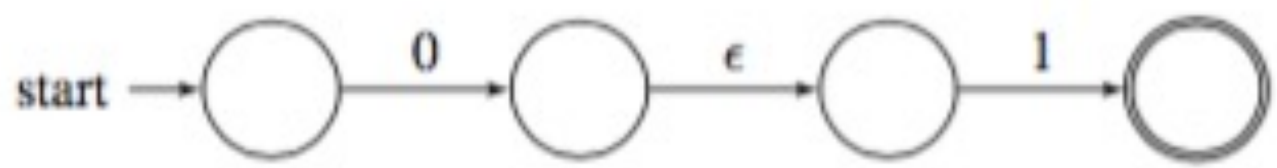
- Forward: Lemma 1.55
  - Ex: Given regular expression $(01 \cup 0)^*$ find the NFA
    - 0
    - 1
    - 01 or $0 \circ 1$
    - $01 \cup 0$
    - $(01 \cup 0)^*$

Regular Language, Forward

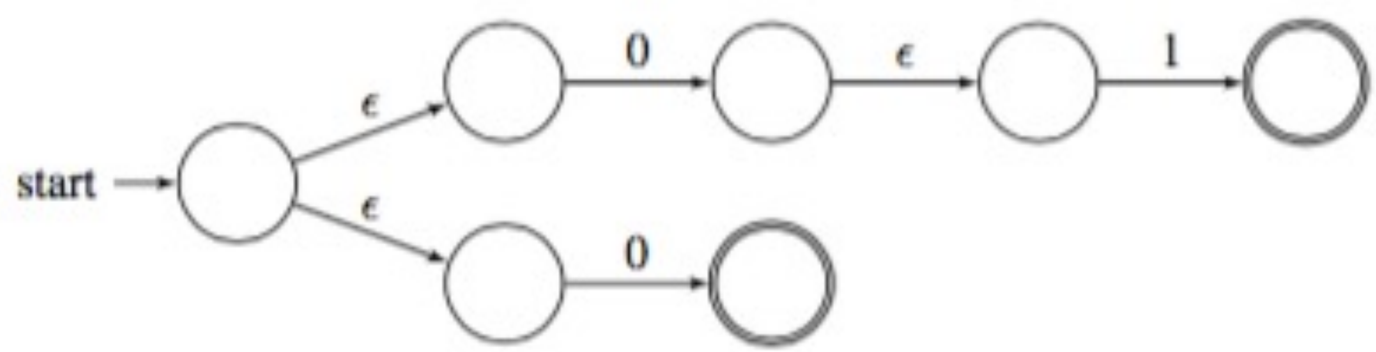
- <u>Forward</u>: Lemma 1.55
  - Ex: Given regular expression (01 ∪ 0)* find the NFA
    - 0

    - 1

    - 01 or 0 ∘1

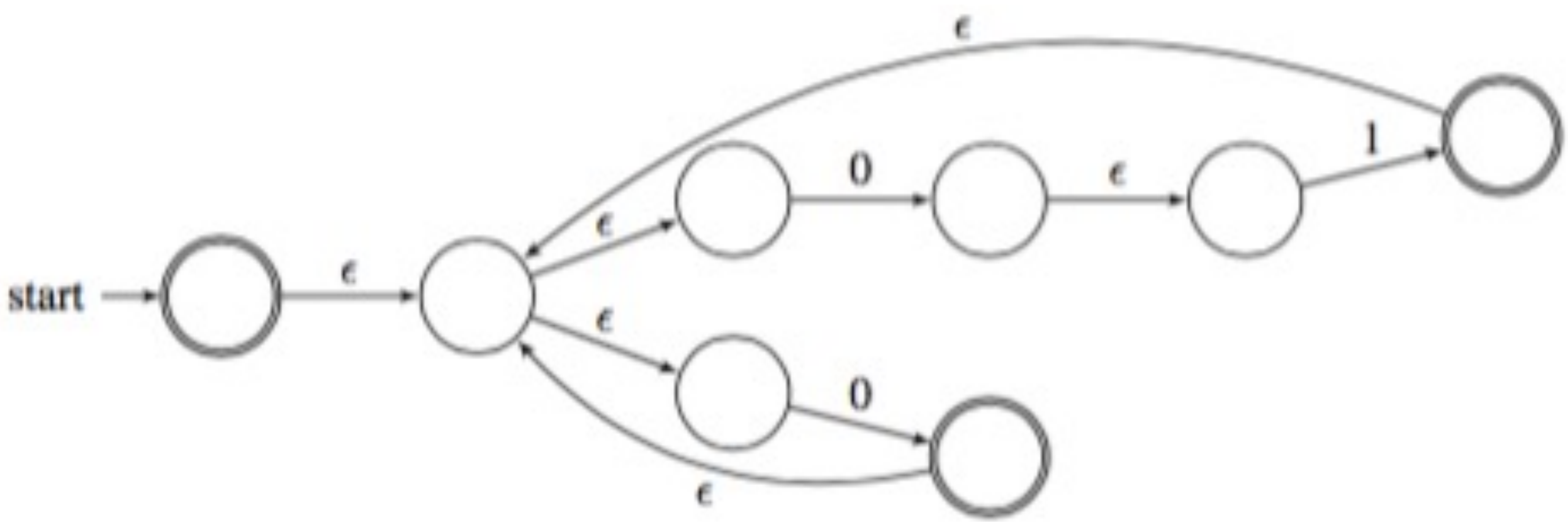


    - 01 ∪ 0



Regular Language, Forward

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

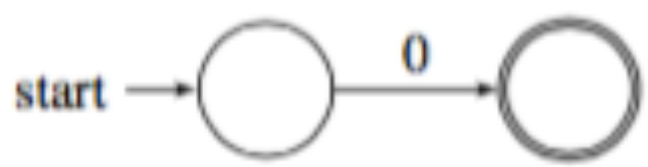


School of Engineering | Computer Science

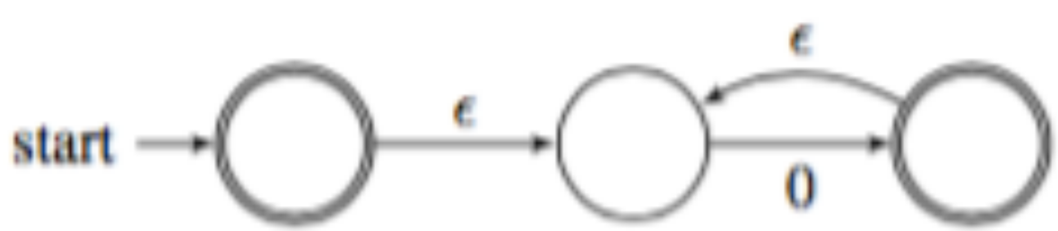
Regular Language, Forward

- Forward: Lemma 1.55
  - Ex 2: Given regular expression $(0^*11) \cup (01)^*$ 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 

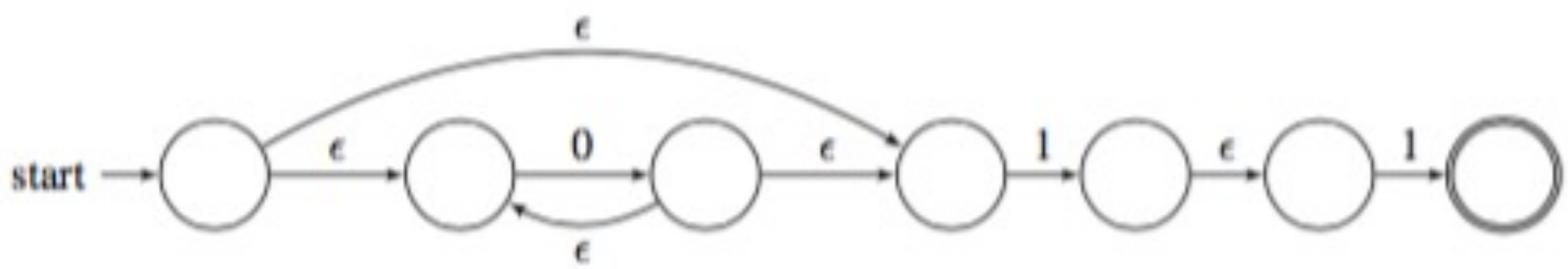
Regular Language, Forward

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

School of Engineering | Computer Science

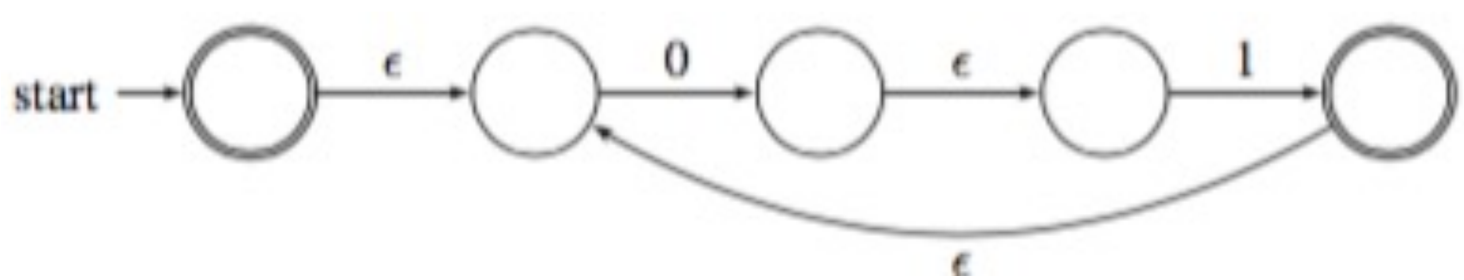
Regular Language, Forward

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



        - 01

        - (01)*



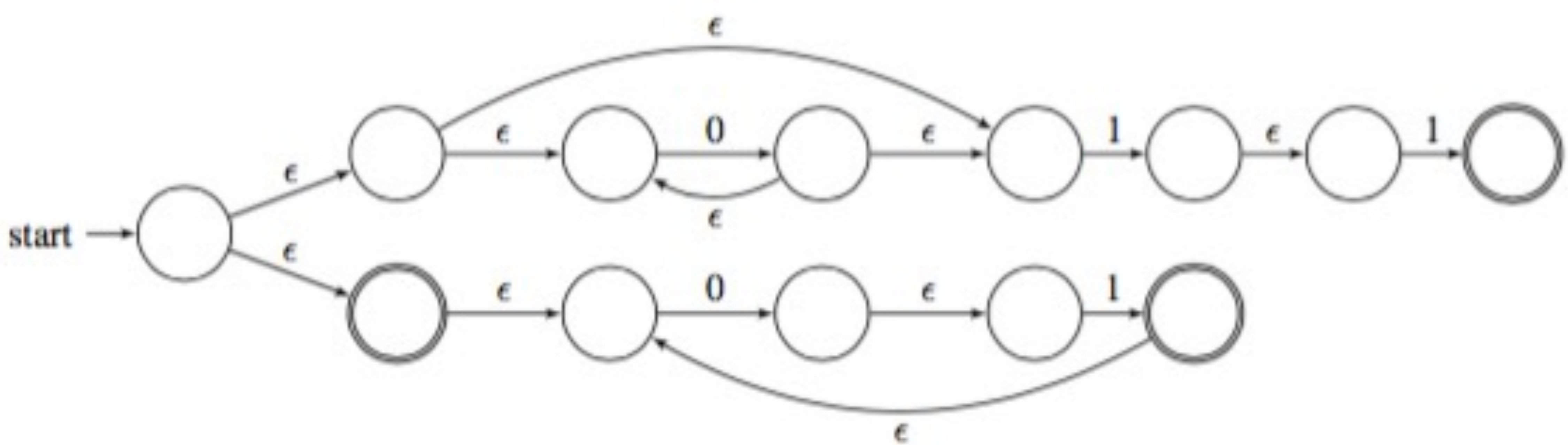
Regular Language, Forward

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

    - $(0^*11) \cup (01)^*$

Regular Language, Forward

- <u>Forward</u>: 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, Σ, δ, 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

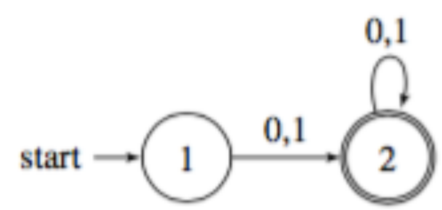
Regular Language

- Lemma 1.60: (“has DFA” ⇒ “has regular expression”)
  - Step 1: Setting up the generalized NFA
    - Let $D = \{Q, \Sigma, \delta, q_0, F\}$

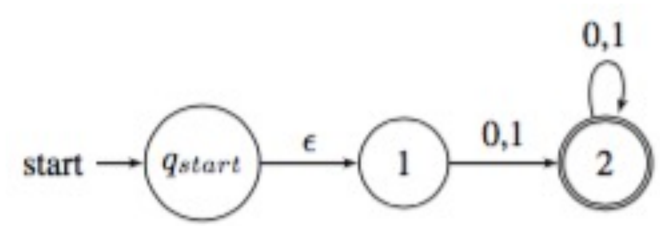
1. Add a new start state q_{start} with an ε-transition to q_0
2. Add a new end state (accept state) q_{end} with ε-transitions from all $q_i \in F$ and mark all q_i as non-accept states
3. For each transition in D with multiple labels, replace each transition's label with the union of all old labels
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

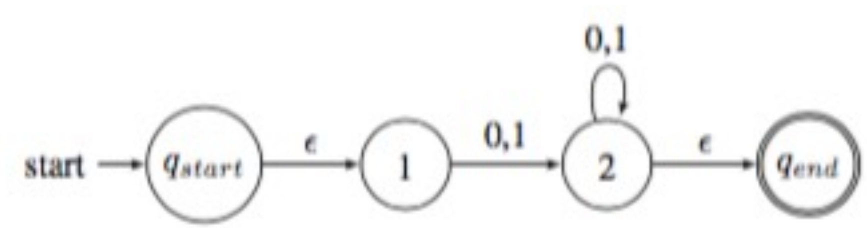


1. NFA



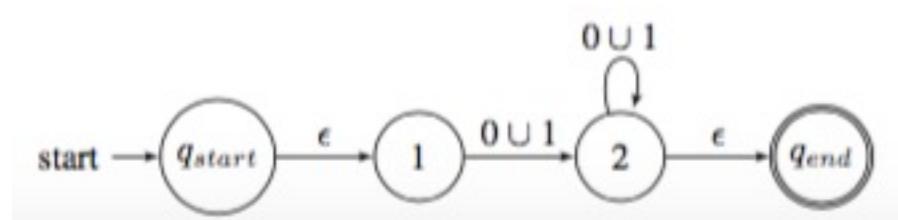
New start state

2. NFA



New end state

3. NFA

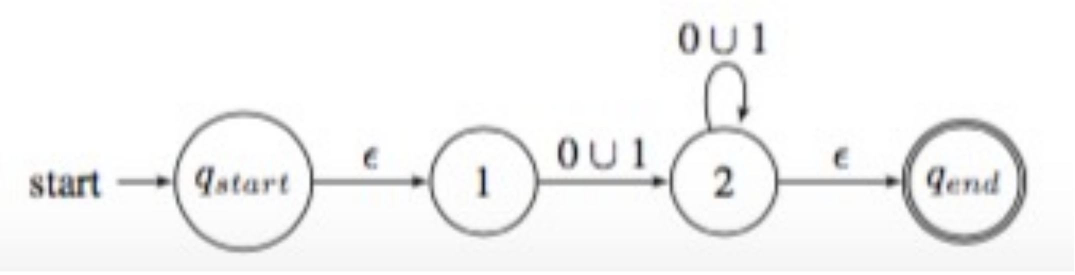


Multiple labels replaced by union

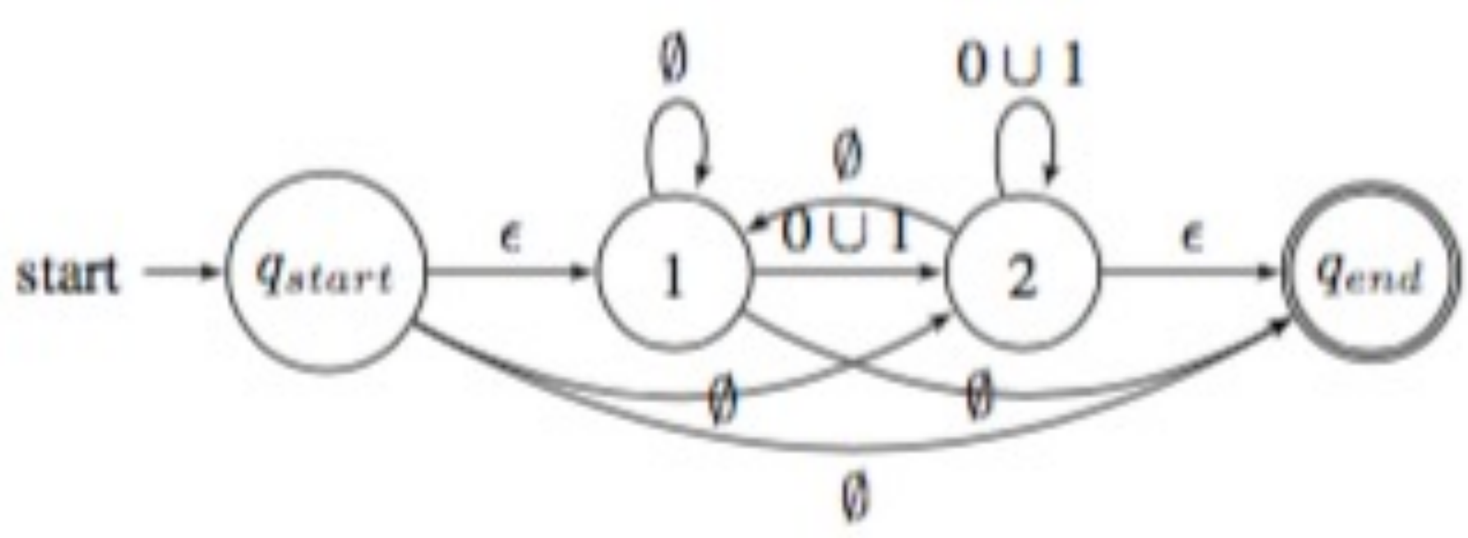
Regular Language

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

3. NFA



4. NFA with added transitions



School of Engineering | Computer Science

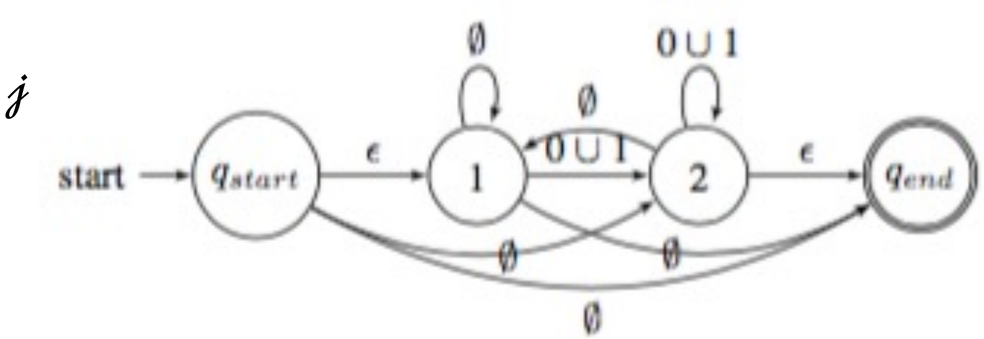
Regular Language

- <u>Lemma 1.60</u>: ("has DFA" ⇒ "has regular expression")
  - <u>Step 2</u>: Reduce the NFA down to two states
    - Goal: k = 2

    - Let k be the number of states in the generalized NFA G
    - <u>Base Case</u>: If $k = 2$, return the regular expression on the arrow from q_{start} to q_{end}
    - <u>Recursive Case</u>: $k > 2$ 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)

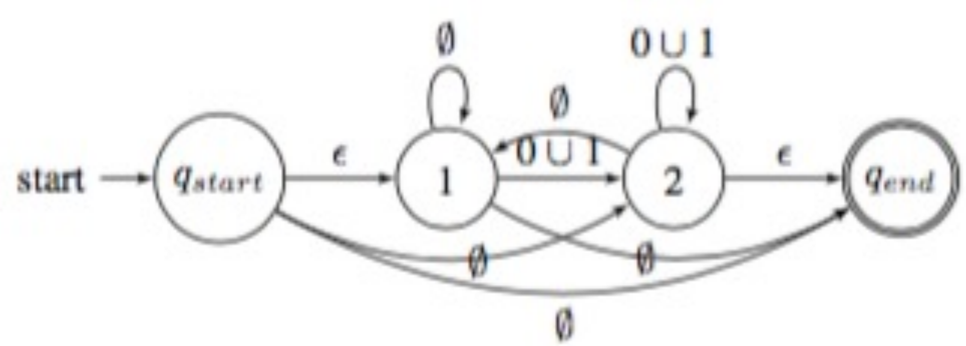
Regular Language

- 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

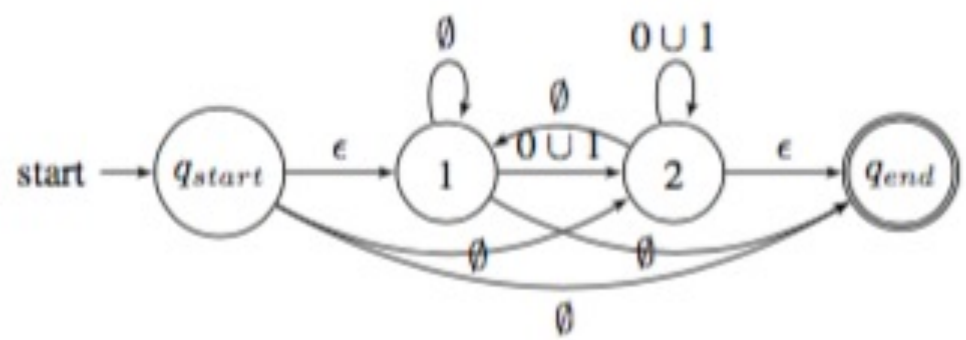
- Lemma 1.60: ("has DFA" ⇒ "has regular expression")
  - Step 2: Reduce the NFA down to two states – Example
    - Recursive Case: $k > 2$ states:



    - 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

$R_4 = \delta(q_i, q_j)$

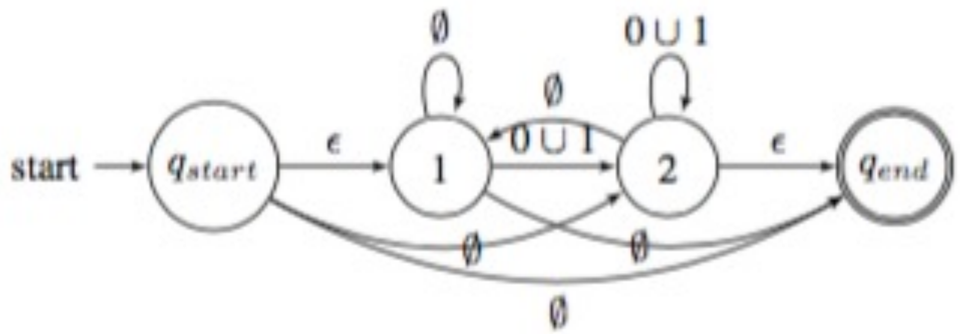
q_i goes to q_j

- Ex: Let q_{rip} = state 1:
  - $(q_i, q_j) = (q_{start}$, state 2)
    - $(R1 \circ R2^* \circ R3) \cup R4$
    - $= (\varepsilon \circ \emptyset^* \circ (0 \cup 1)) \cup \emptyset = 0 \cup 1$

VCU School of Engineering | Computer Science

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

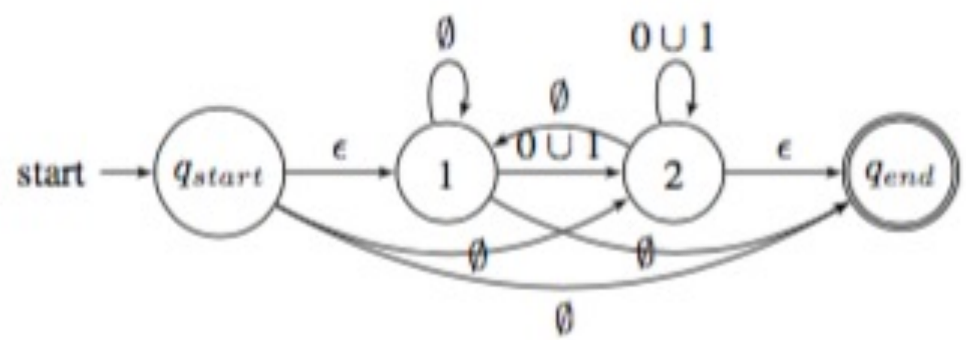
$R_4 = \delta(q_i, q_j)$

q_i goes to q_j

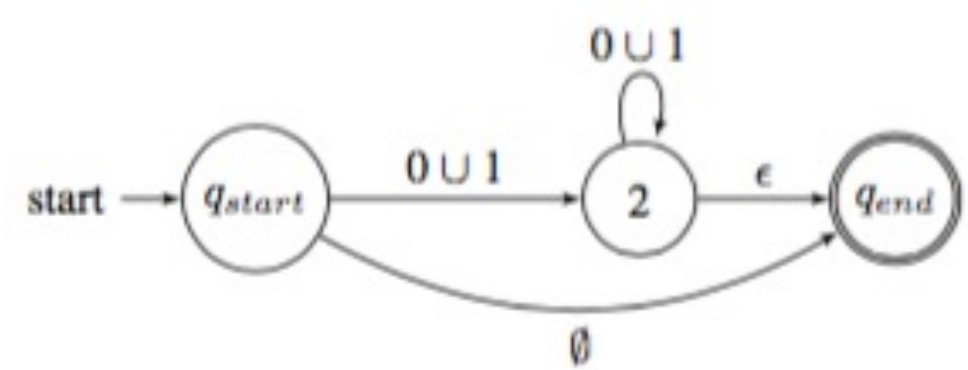
- Ex: Let q_{rip} = state 1:
  - (q_i, q_j) = (state 2, state 2)
    - $(R1 \circ R2^* \circ R3) \cup R4$
    - $= (\emptyset \circ \emptyset^* \circ (0 \cup 1)) \cup (0 \cup 1) = 0 \cup 1$

Regular Language

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

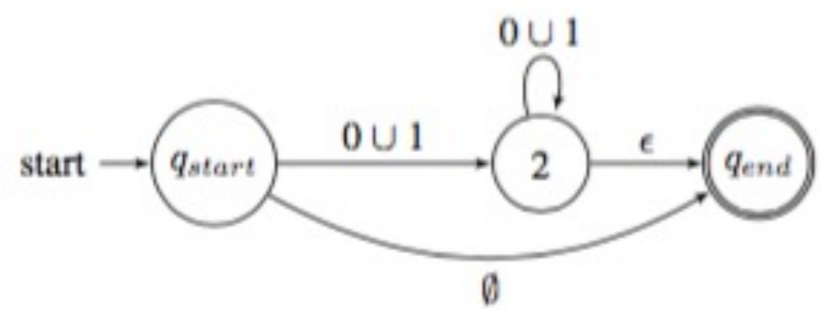


    - <u>Ex</u>: Let q_{rip} = state 1:
      - (q_i, q_j) = (q_{start}, state 2) = $0 \cup 1$
      - (q_i, q_j) = (state 2, state 2) = $0 \cup 1$

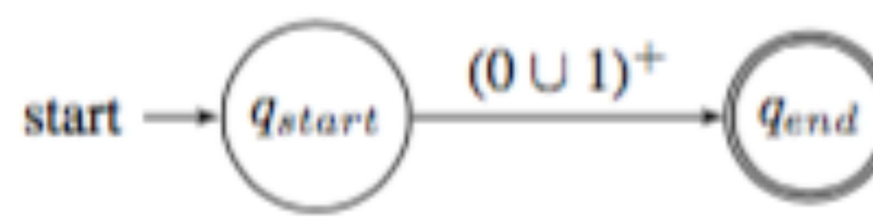


Regular Language

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

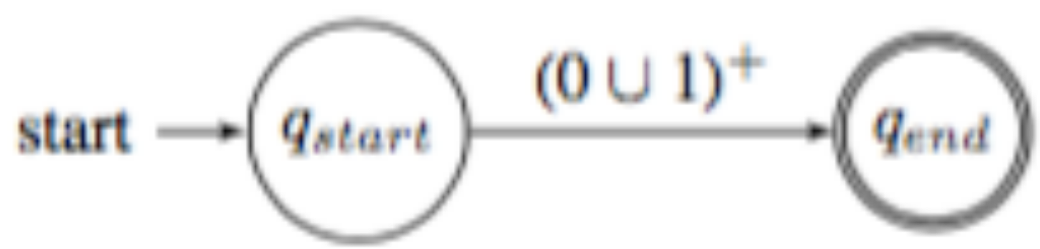


- Ex: Let q_{rip} = state 2, must consider one pair (q_i, q_j):
  - (q_i, q_j) = (q_{start}, q_{end})
    - = (R1 ∘ R2* ∘ R3) ∪ R4
    - = $((0 \cup 1) \circ (0 \cup 1)^* \circ \varepsilon) \cup \emptyset$
    - = $(0 \cup 1)(0 \cup 1)^* = (0 \cup 1)^+$



Regular Language

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



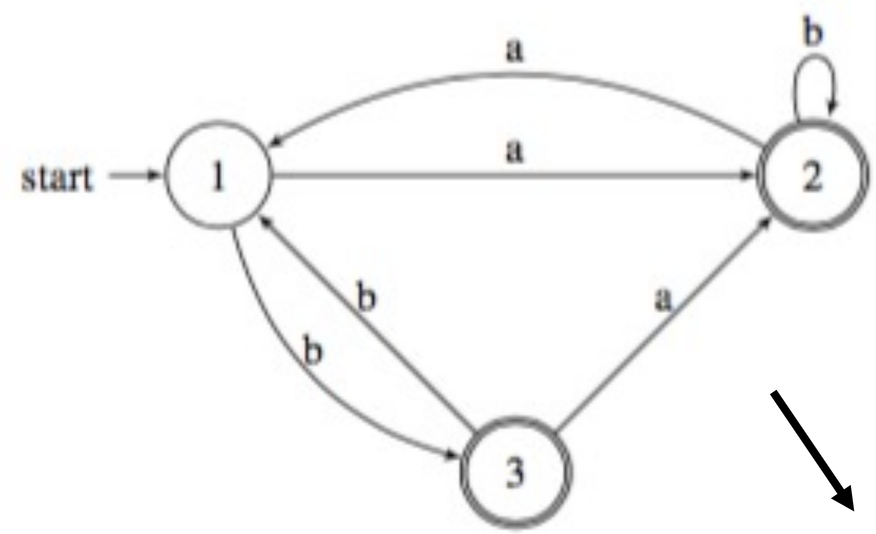
    - No other reductions needed, so $(0 \cup 1)^+$ is the regular expression

Regular Language

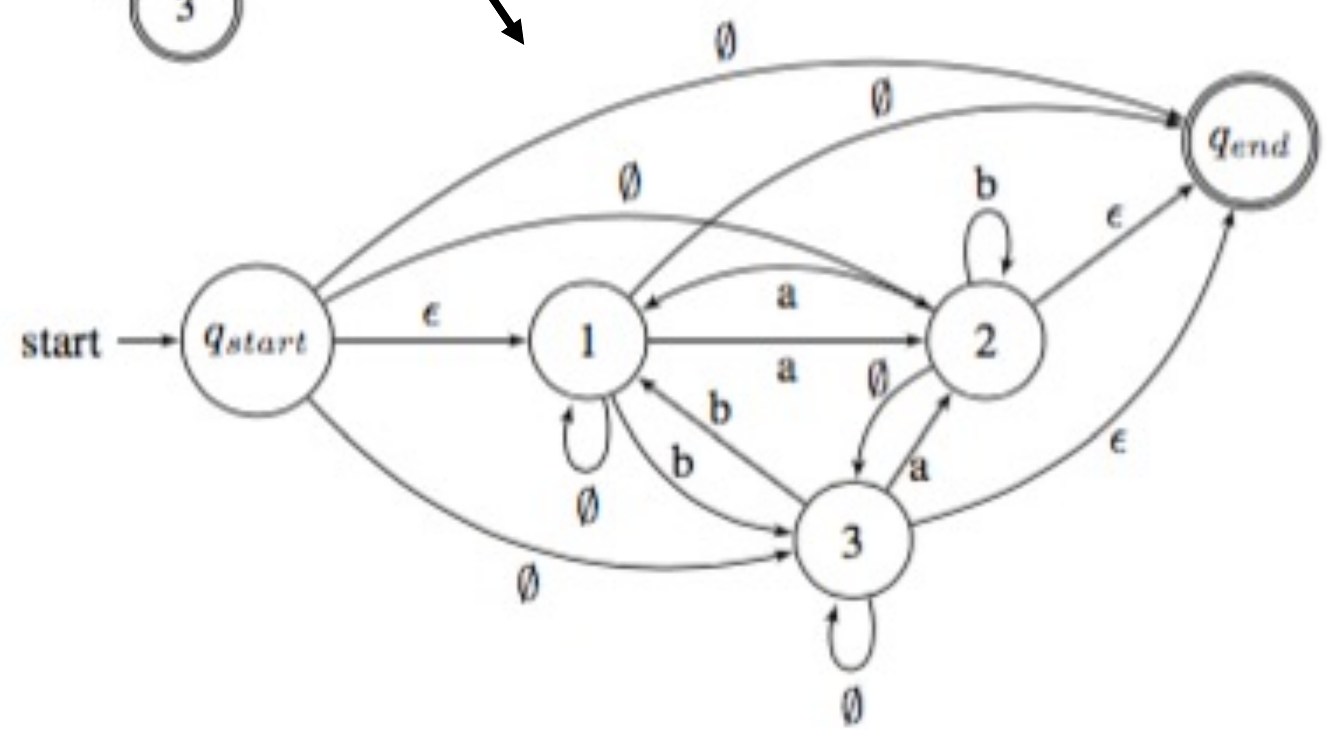
- Theorem 1.54: A language is regular if and only if some regular expression describes it.
  - Forward: Lemma 1.55
    - Claim: If a language L is described by a regular expression, then the language L is regular.
  - Backwards: Lemma 1.60
    - Claim: If a language is regular, then it can be described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

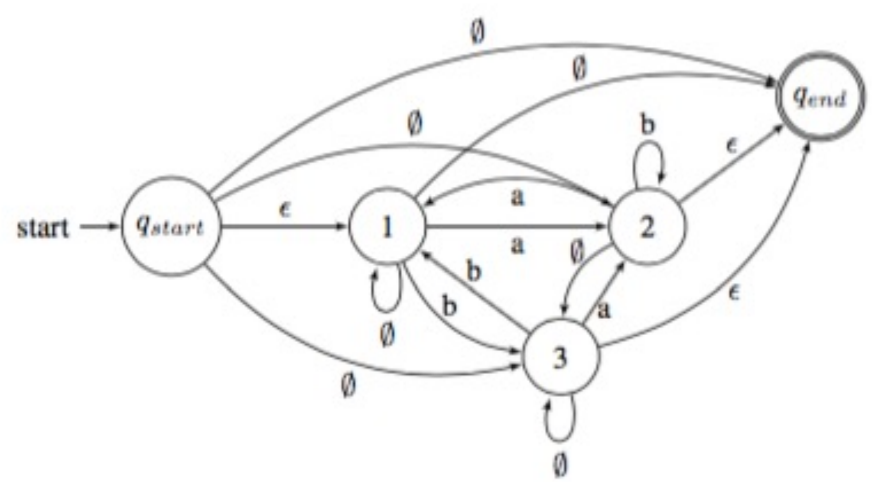


    - NFA



Regular Language

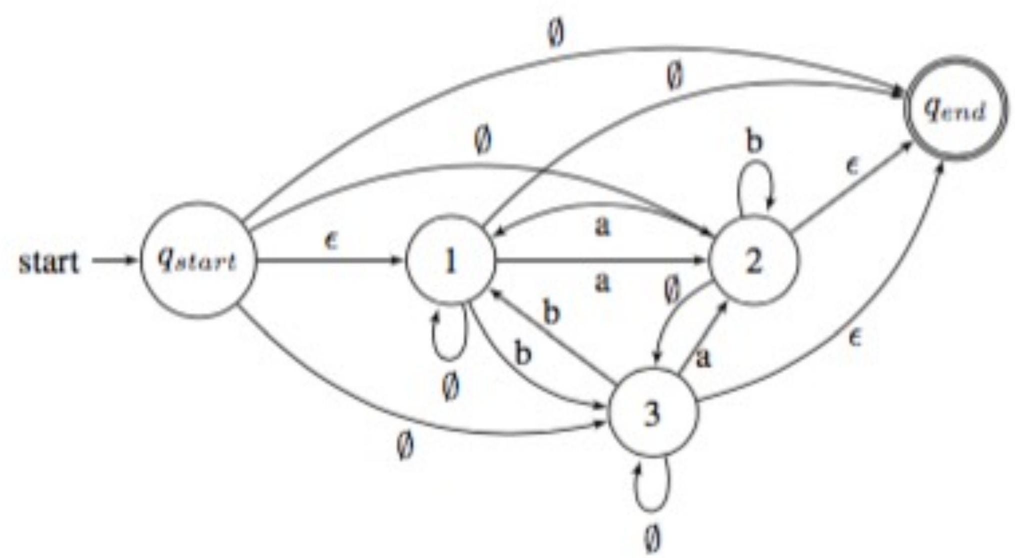
- Lemma 1.60: Example 2
  - Step 2: Reduce the NFA down to two states
    - NFA



    - 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

$R_4 = \delta(q_i, q_j)$

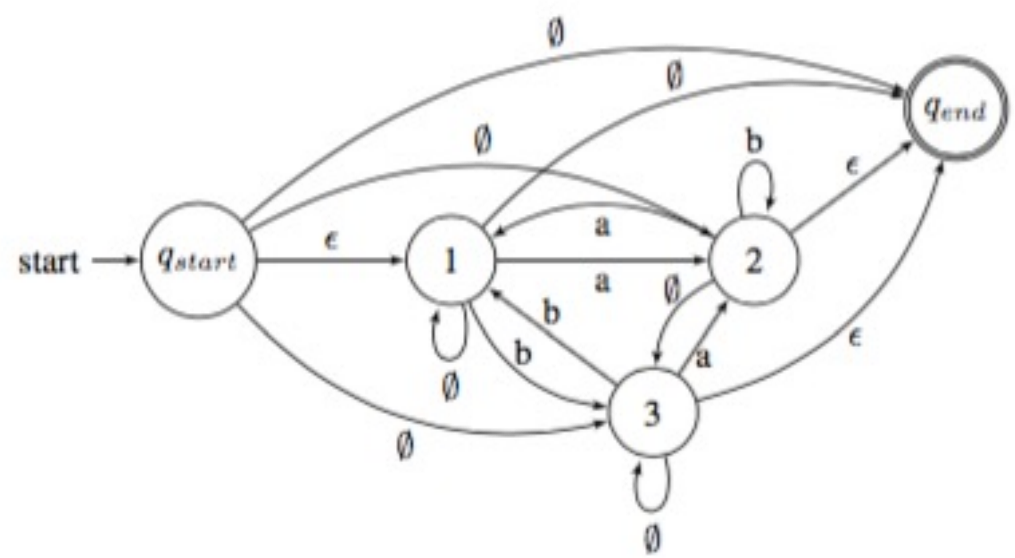
q_i goes to q_j

- Let q_{rip} = state 1, (q_{start}, state 2),
  - $(R1 \circ R2^* \circ R3) \cup R4$
  - $= (\varepsilon \circ \emptyset^* \circ a) \cup \emptyset = a$

School of Engineering | Computer Science

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

$R_4 = \delta(q_i, q_j)$

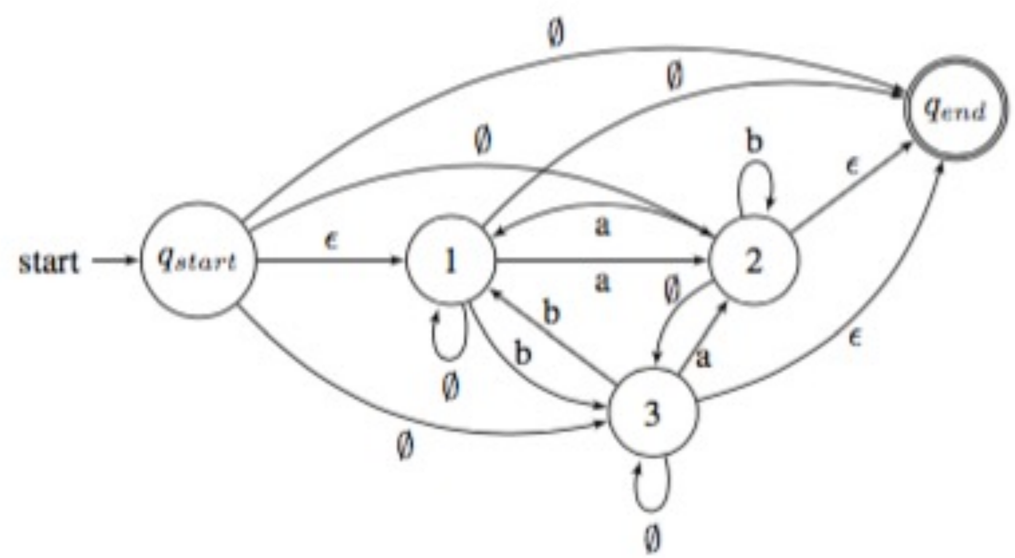
q_i goes to q_j

- Let q_{rip} = state 1, (state 2, state 2)
  - $(R1 \circ R2^* \circ R3) \cup R4$
  - $= (a \circ \emptyset^* \circ a) \cup b = aa \cup b$

School of Engineering | Computer Science

Regular Language

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

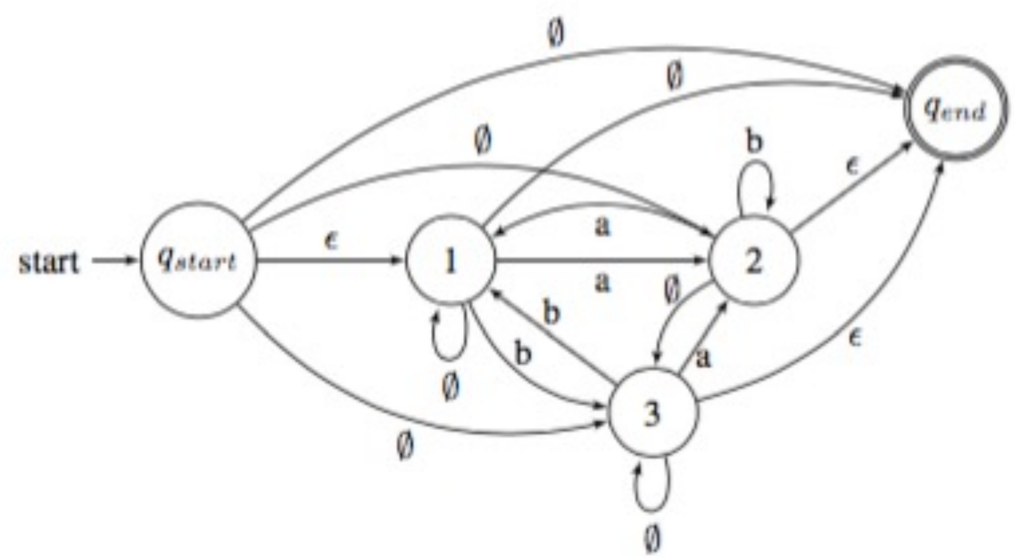


$R_1 = \delta(q_i, q_{rip})$
q_i goes to q_{rip}
$R_2 = \delta(q_{rip}, q_{rip})$
q_{rip} goes to itself
$R_3 = \delta(q_{rip}, q_j)$
q_{rip} goes to q_j
$R_4 = \delta(q_i, q_j)$
q_i goes to q_j

- Let q_{rip} = state 1, so (state 2, state 3)
  - $(R1 \circ R2^* \circ R3) \cup R4$
  - $= (a \circ \emptyset^* \circ b) \cup \emptyset = ab$

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

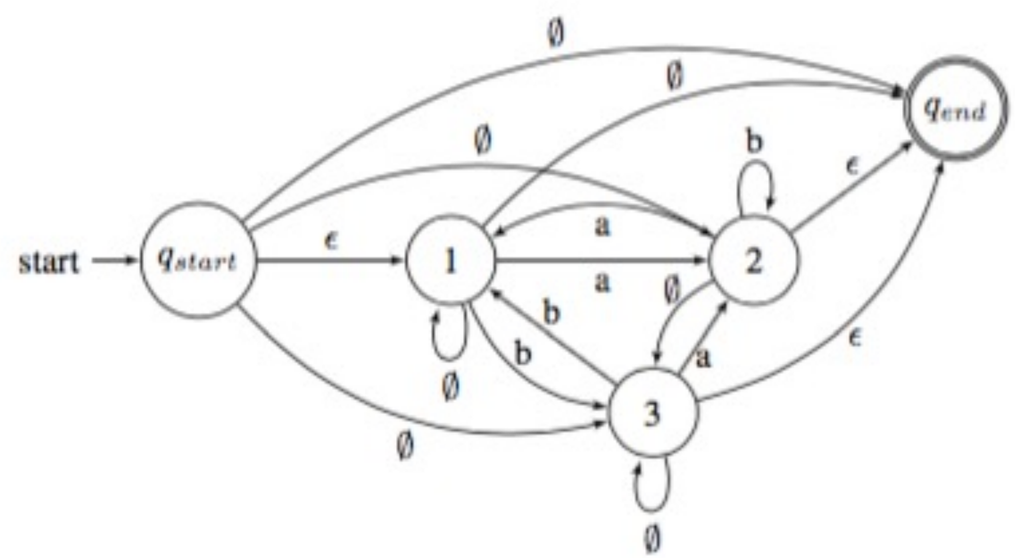
$R_4 = \delta(q_i, q_j)$

q_i goes to q_j

- Let q_{rip} = state 1, (q_{start}, state 3)
  - $(R1 \circ R2^* \circ R3) \cup R4$
  - $= (\varepsilon \circ \emptyset^* \circ b) \cup \emptyset = b$

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

$R_4 = \delta(q_i, q_j)$

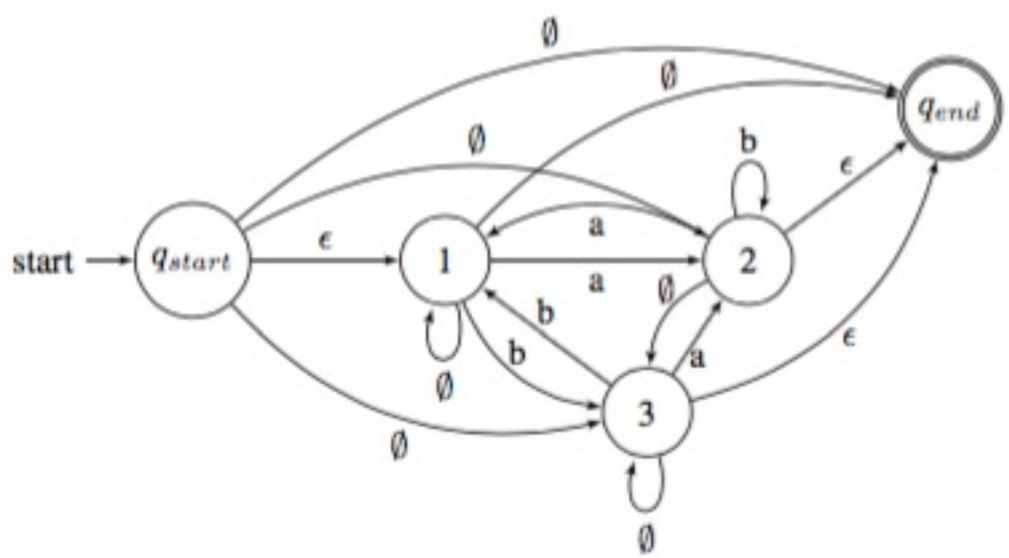
q_i goes to q_j

- Let q_{rip} = state 1, (state 3, state 3)
  - $(R1 \circ R2^* \circ R3) \cup R4$
  - $= (b \circ \emptyset^* \circ b) \cup \emptyset = bb$

School of Engineering | Computer Science

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

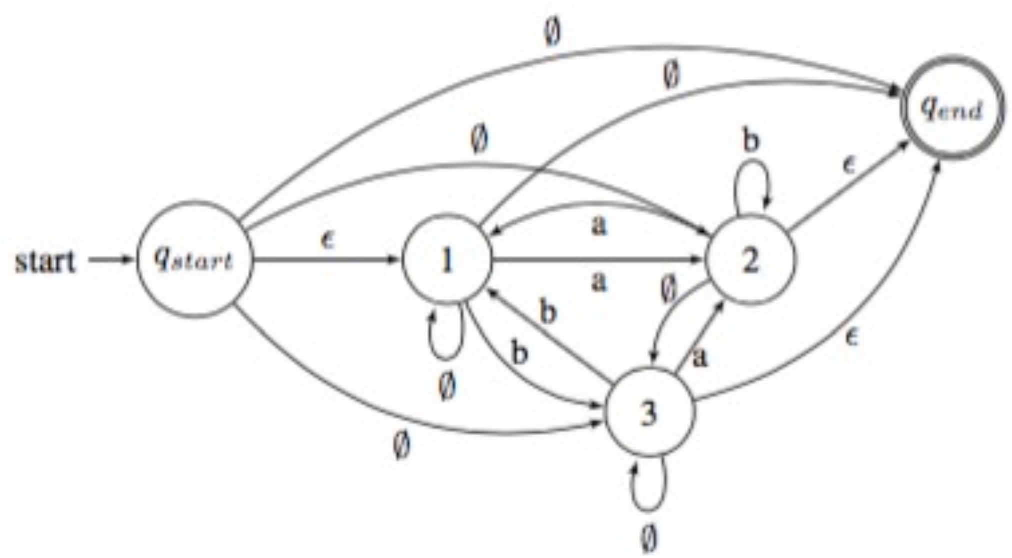
$R_4 = \delta(q_i, q_j)$

q_i goes to q_j

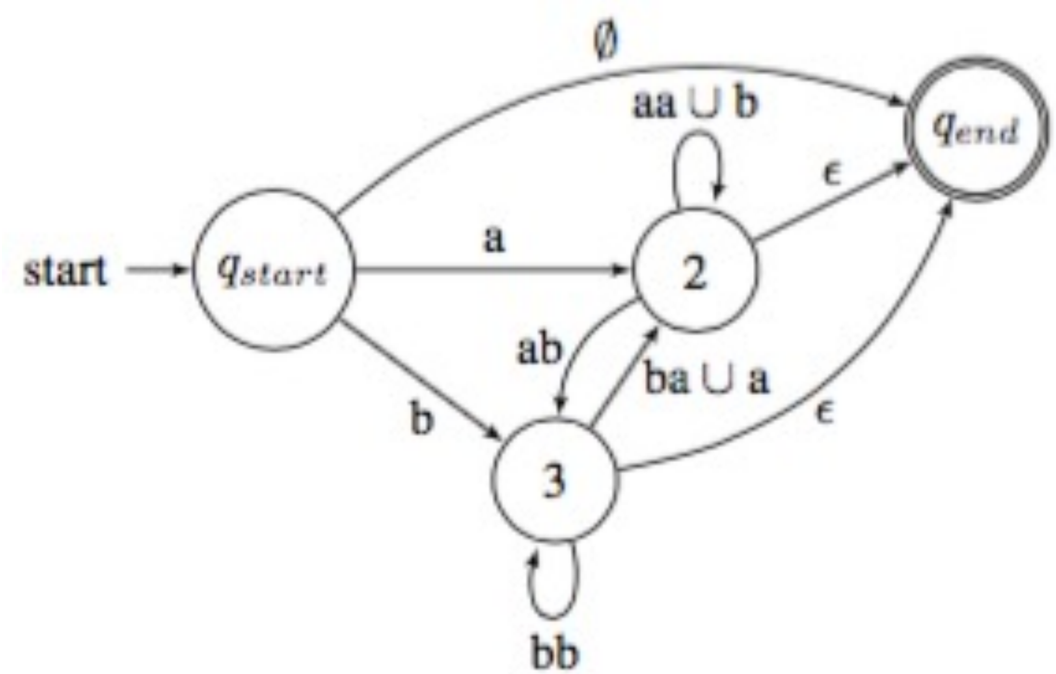
- Let q_{rip} = state 1, (state 3, state 2)
  - (R1 ∘ R2* ∘ R3) ∪ R4
  - = (b ∘ ∅* ∘ a) ∪ a = ba ∪ a

Regular Language

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

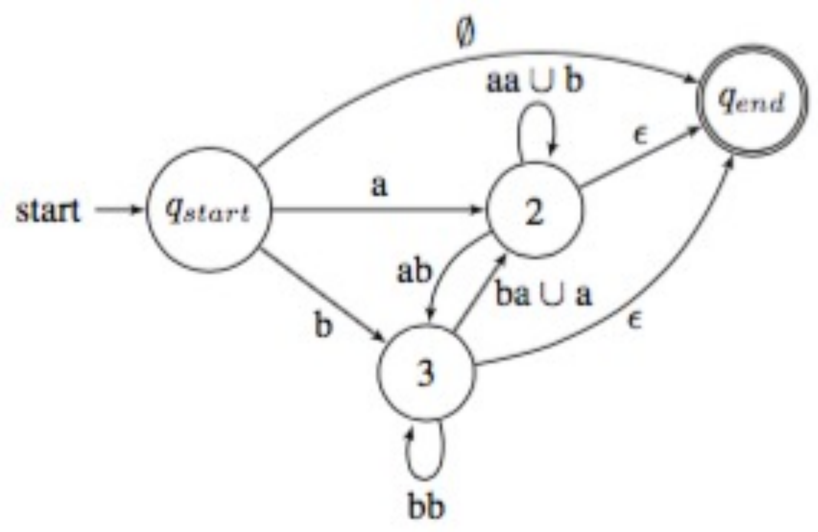


- Let q_{rip} = state 1, so



Regular Language

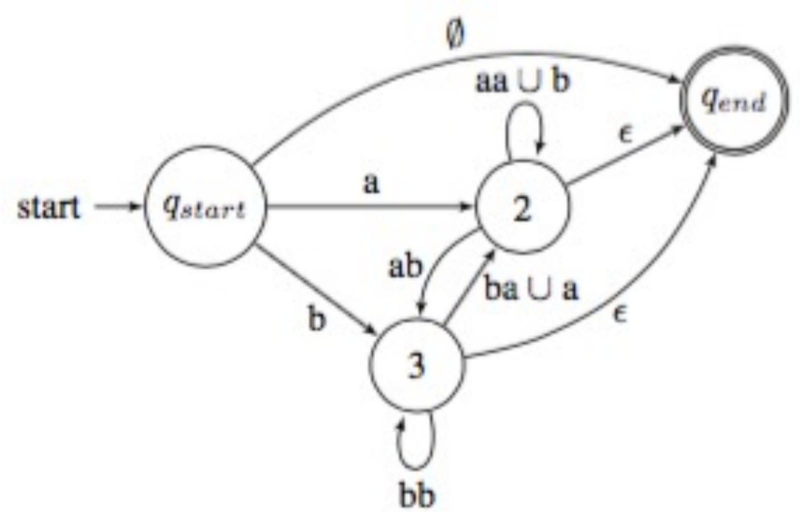
- Lemma 1.60: Example 2
  - Step 2: Reduce the NFA down to two states
  - NFA



- 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

- <u>Lemma 1.60</u>: Example 2
  - <u>Step 2</u>: 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

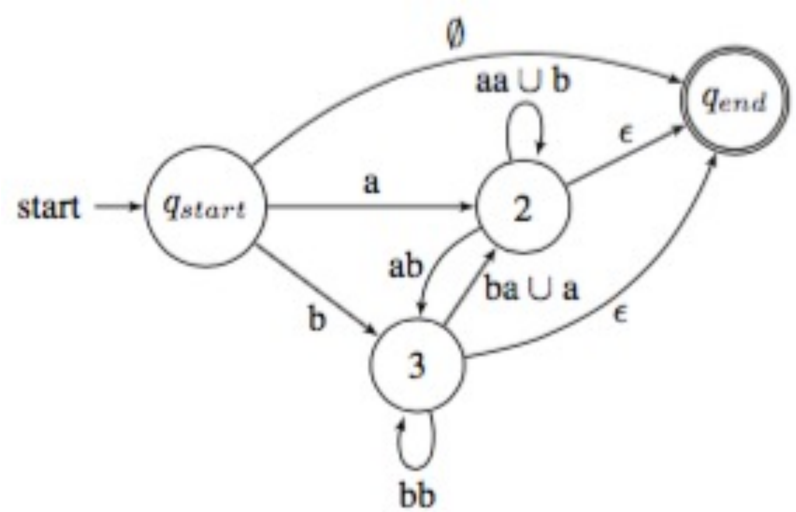
$R_4 = \delta(q_i, q_j)$

q_i goes to q_j

School of Engineering | Computer Science

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

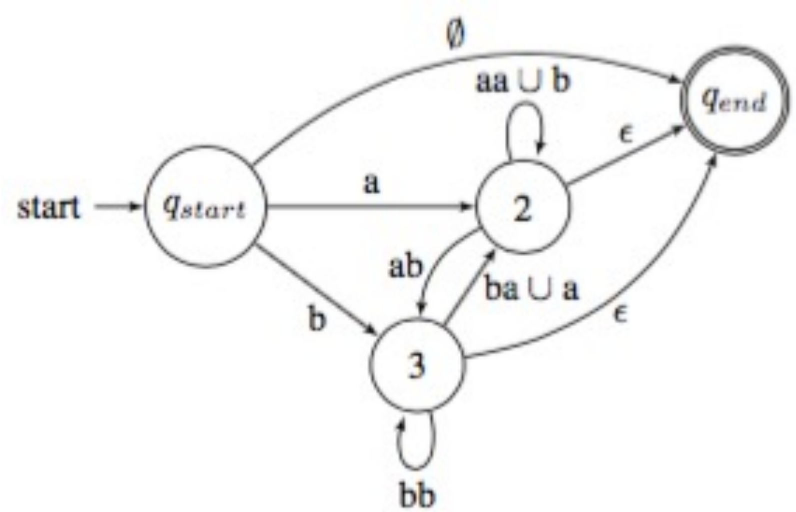
$R_4 = \delta(q_i, q_j)$

q_i goes to q_j

School of Engineering | Computer Science

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

$R_4 = \delta(q_i, q_j)$

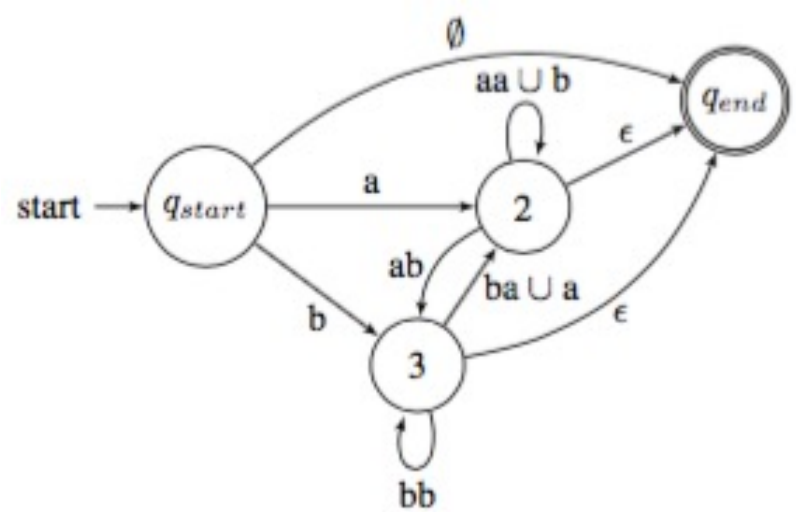
q_i goes to q_j

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

School of Engineering | Computer Science

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})
  - $(R1 \circ R2^* \circ R3) \cup R4$
  - $= ((ba \cup a) \circ (aa \cup b)^* \circ \varepsilon) \cup \varepsilon$
  - $= (ba \cup a)(aa \cup b)^* \cup \varepsilon$

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

q_i goes to q_{rip}

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

q_{rip} goes to itself

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

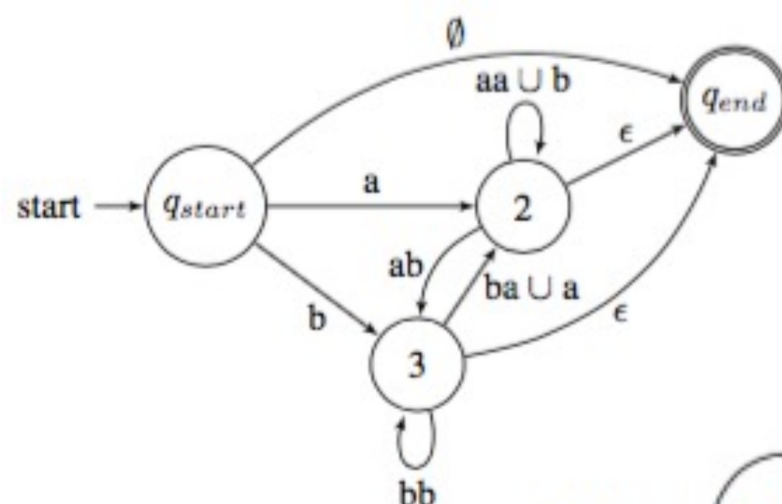
q_{rip} goes to q_j

$R_4 = \delta(q_i, q_j)$

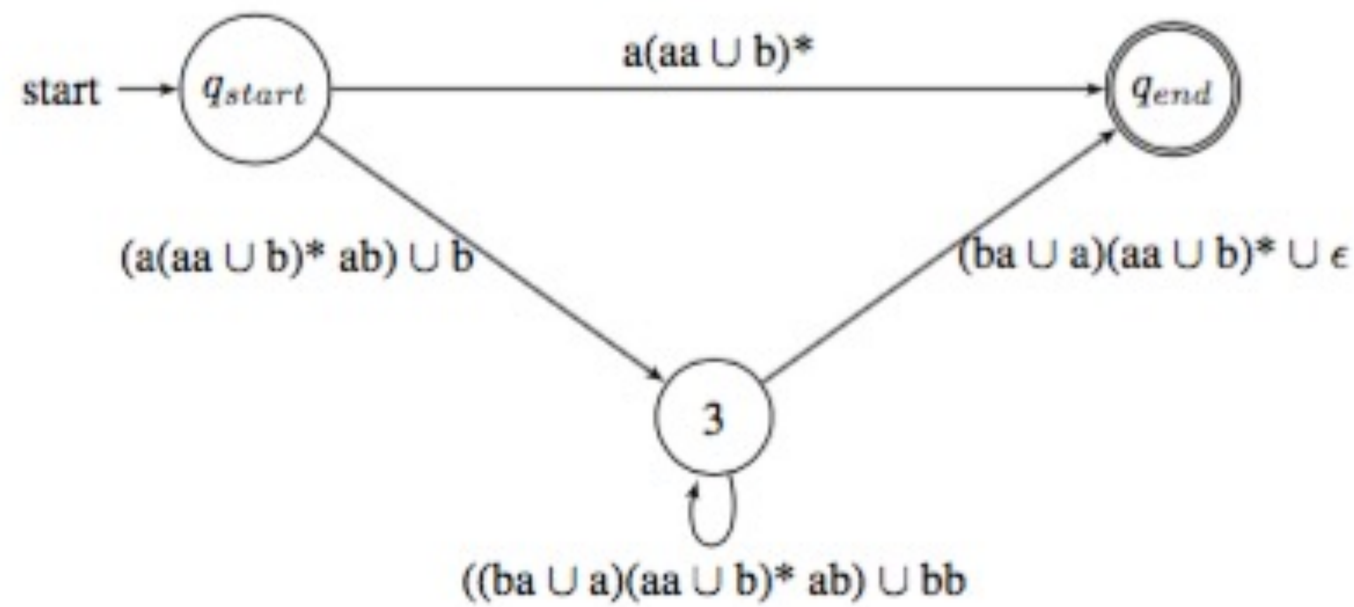
q_i goes to q_j

Regular Language

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

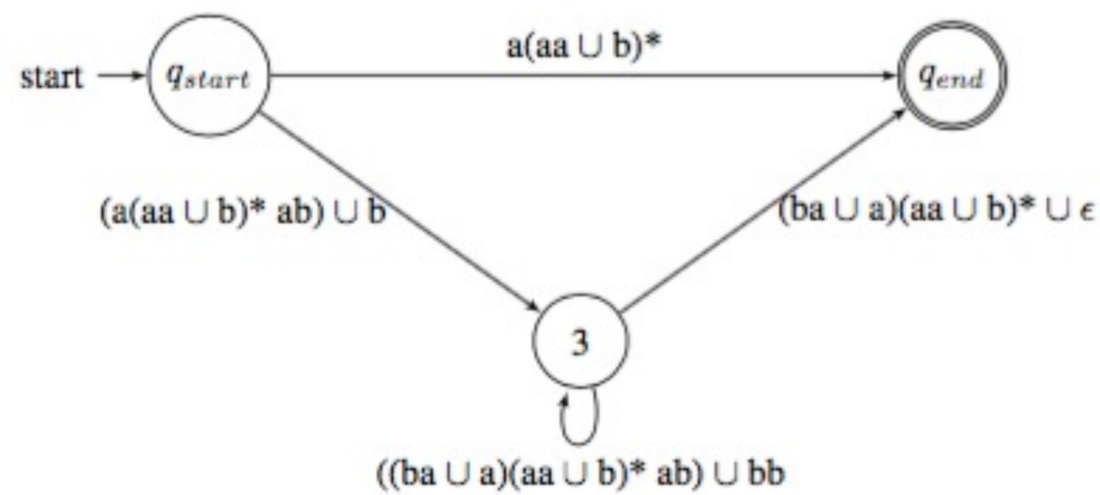


- Let q_{rip} = state 2, so



Regular Language

- <u>Lemma 1.60</u>: Example 2
  - <u>Step 2</u>: Reduce the NFA down to two states
    - NFA

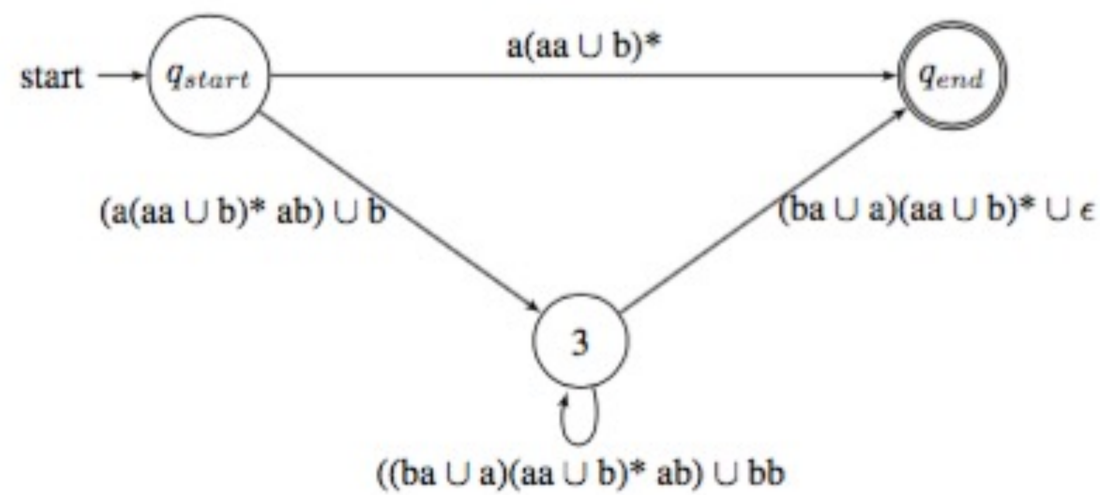


- Let q_{rip} = state 3, so only 1 transition: (q_{start}, q_{end})
  - $(R1 \circ R2^* \circ R3) \cup R4$
  - $= ((a(aa \cup b)^* ab) \cup b) \circ (((ba \cup a)(aa \cup b)^* ab) \cup bb)^* \circ ((ba \cup a)(aa \cup b)^* \cup \varepsilon) \cup (a(aa \cup b)^*)$

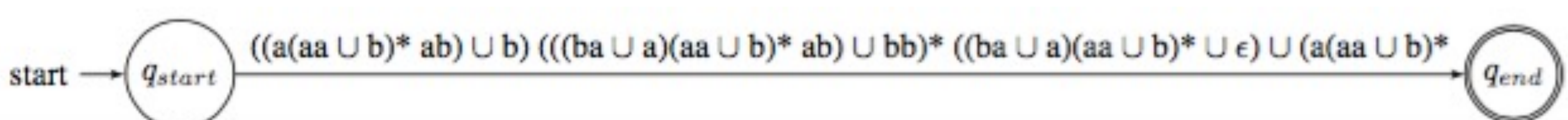
School of Engineering | Computer Science

Regular Language

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



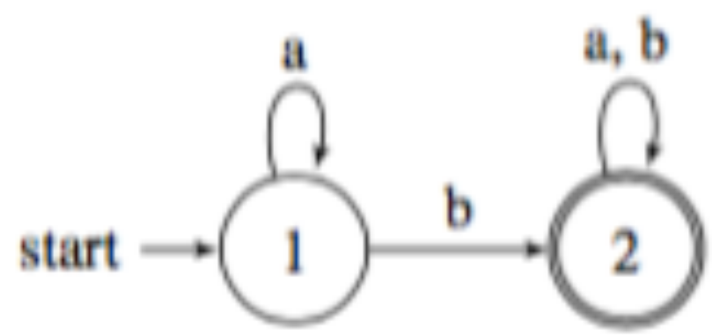
    - $k = 2$, so $R = ((a(aa \cup b)^*\ ab) \cup b) \circ (((ba \cup a)(aa \cup b)^*\ ab) \cup bb)^* \circ ((ba \cup a)(aa \cup b)^* \cup \varepsilon) \cup (a(aa \cup b)^*)$



Try It

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

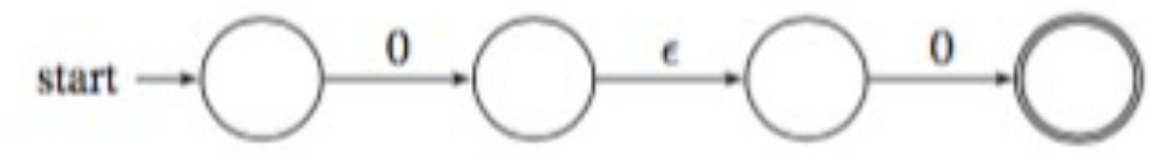
2. Convert the DFA below into a Regular Expression.



Try It

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

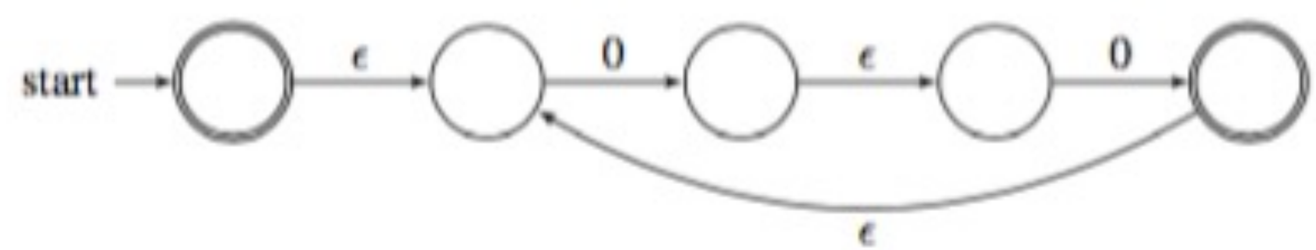
- 00



- 11

- 01

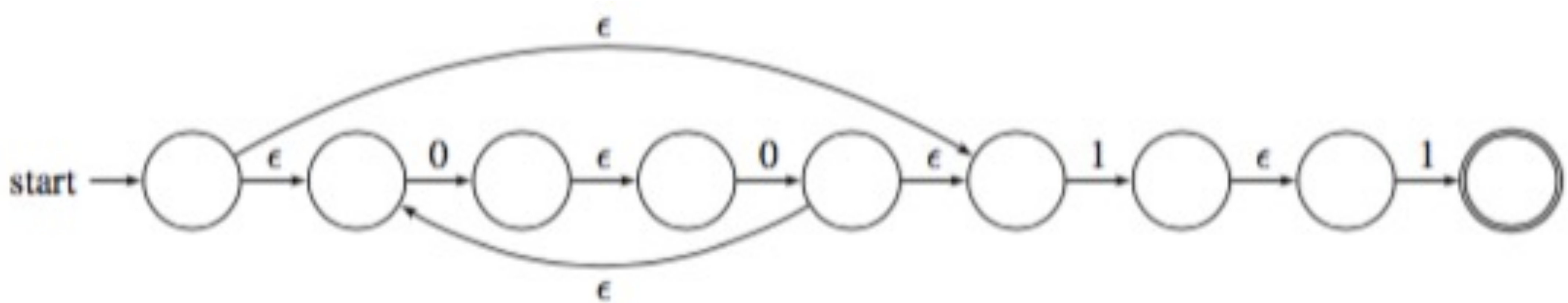
- (00)*



Try It

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

- (00)*11

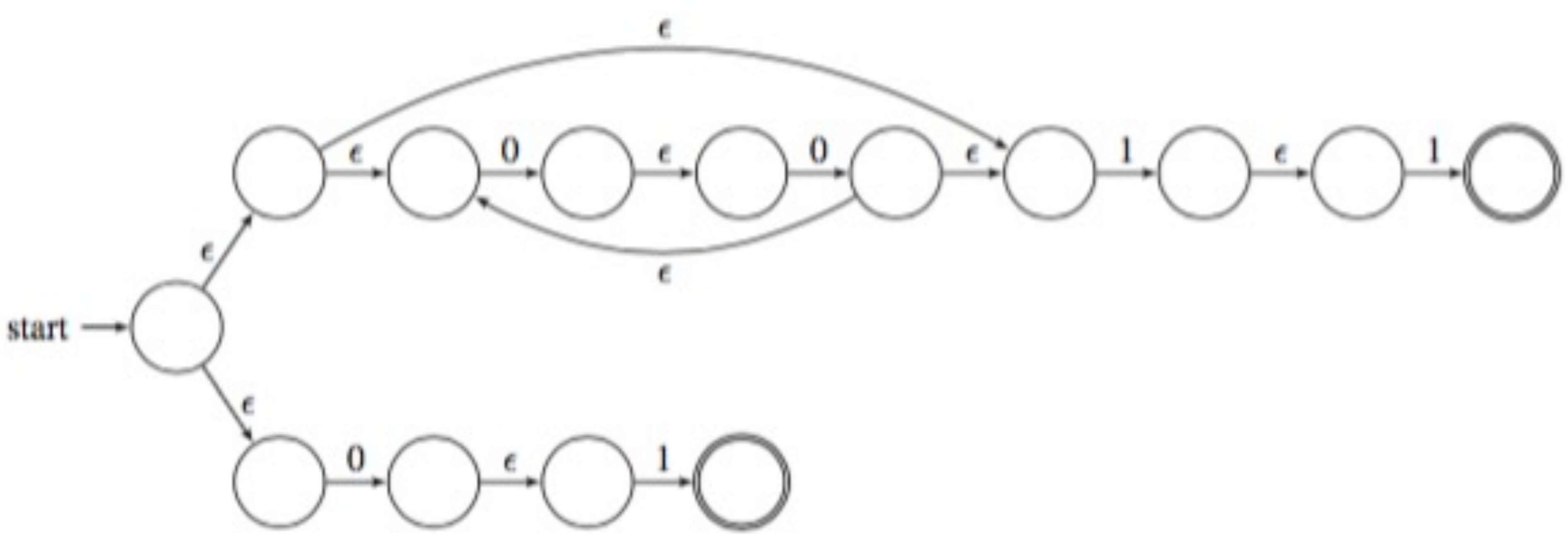


School of Engineering | Computer Science

Try It

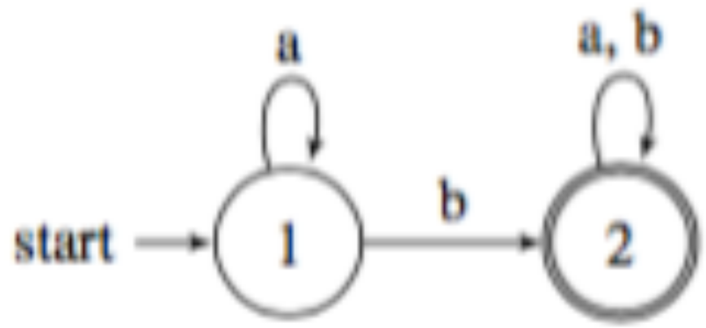
1. Construct an NFA from the Regular Expression ((00)*11) ∪ 01, cont.

- ((00)*11) ∪ 01

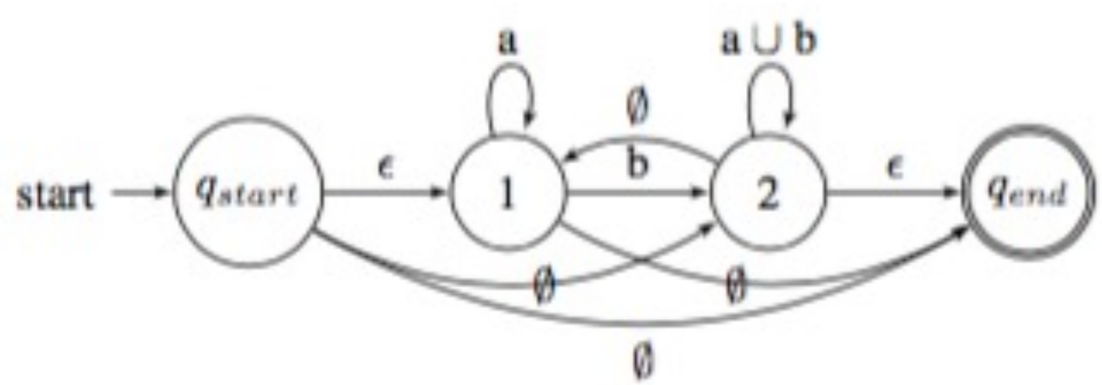


Try It

2. Convert the DFA below into a Regular Expression.



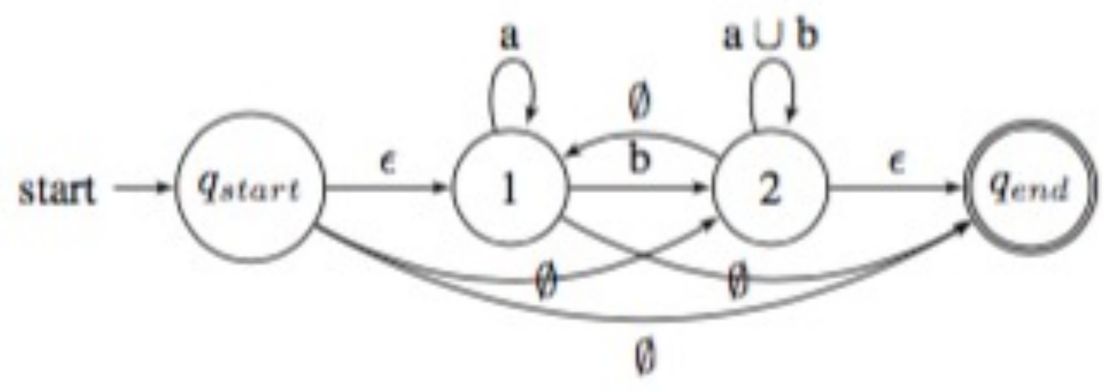
- Set up the generalized NFA



School of Engineering | Computer Science

Try It

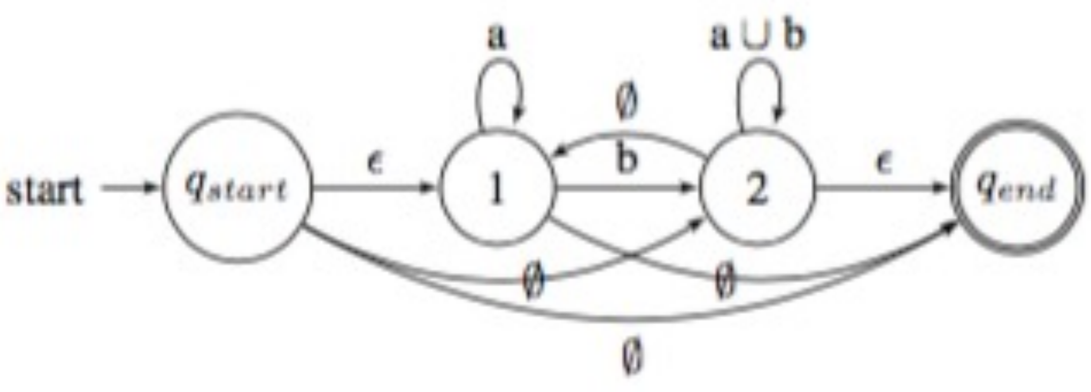
2. Convert the NFA below into a Regular Expression.



- 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.

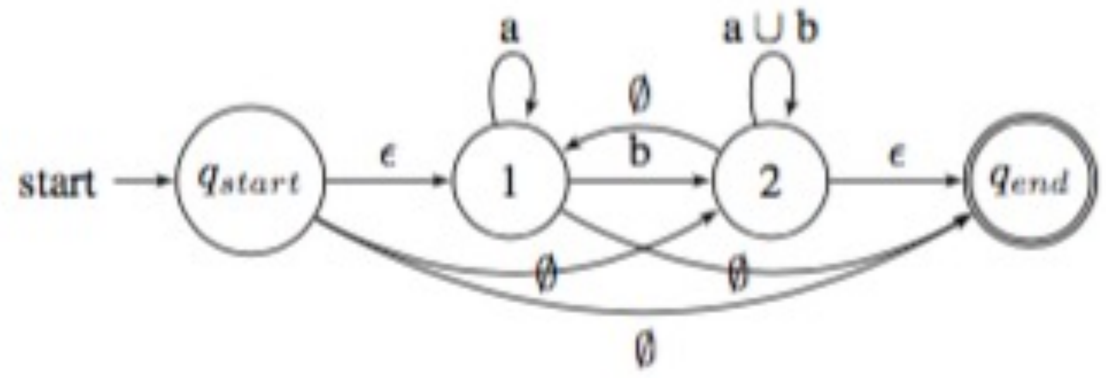


- Let q_{rip} = state 1, (q_{start}, state 2)
  - (R1 ∘ R2* ∘ R3) ∪ R4
  - = (ε ∘ a* ∘ b) ∪ ∅ = a*b
- Let q_{rip} = state 1, (state 2, state 2)
  - (R1 ∘ R2* ∘ R3) ∪ R4
  - = (∅ ∘ a* ∘ b) ∪ (a ∪ b) = a ∪ b

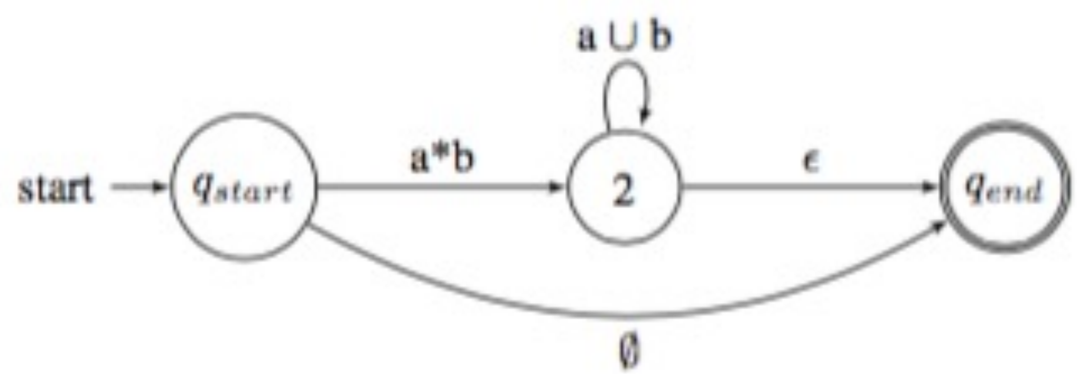
School of Engineering | Computer Science

Try It

2. Convert the NFA below into a Regular Expression.



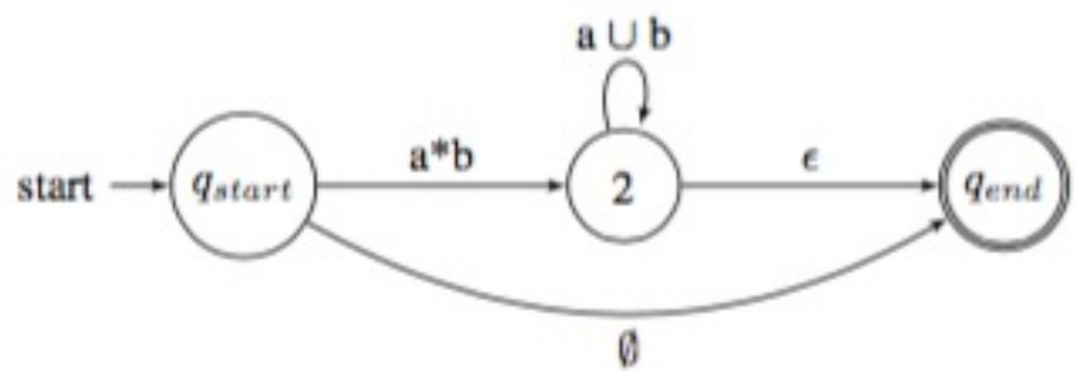
- Let q_{rip} = state 1, so get:



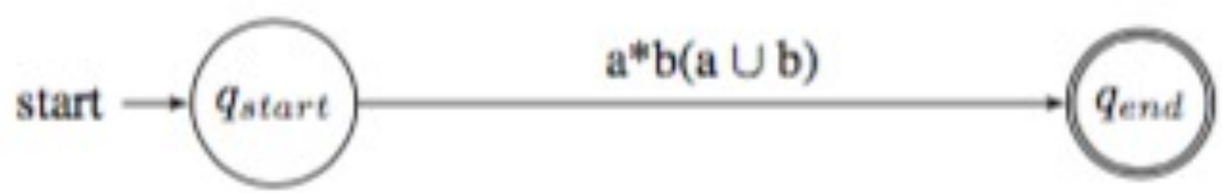
VCU School of Engineering | Computer Science

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 ∘ R2* ∘ R3) ∪ R4
  - = (a*b ∘ (a ∪ b)* ∘ ε) ∪ ∅ = **a*b(a ∪ b)*** = R



VCU School of Engineering | Computer Science