Languages and Computation (COMP2049/AE2LAC)

Regular Expressions, Nondeterminism, and Kleene's Theorem

Dr. Tianxiang Cui

tianxiang.cui@nottingham.edu.cn

Regular Languages

- Many simple languages can be expressed by a formula involving the operations of **union**, **concatenation** and **Kleene star**.
- Example: $\Sigma = \{a,b\}$
  - Strings ending in aa: $\{a, b\}^* \{aa\}$
    - This is a simplification of $(\{a\} \cup \{b\})^*\{a\}\{a\}$
  - Strings containing the substring ab or the substring bba: $\{a, b\}^* \{ab, bba\} \{a, b\}^*$
- These are called regular languages

Regular Languages

- **Definition**:
- The set of regular languages ***R*** over an alphabet $\sum$ is defined recursively as follows:
- *Basis Clause*:
  - The empty language $\varnothing$ is the element of ***R***
  - For any symbol $s \in \sum$, the language $\{s\}$ is the element of ***R***
- *Inductive Clause*:
  - For every two languages L_1 and L_2 in ***R***, the three languages $L_1 \cup L_2$, $L_1 L_2$, and $L_1{}^*$ are elements of ***R***

Regular Languages: Example

- The empty language $\emptyset$ is the element of $\boldsymbol{R}$, so $\emptyset^*$ is also the element of $\boldsymbol{R}$

- $\sum = (a,b)$
- $\{a\}$, $\{b\}$ are the elements of $\boldsymbol{R}$
  - The language containing only one word of length 1
- $\{a, b\}$ ($= \{a\} \cup \{b\}$) and $\{ab\}$ ($= \{a\}\{b\}$) are the elements of $\boldsymbol{R}$
- Since $\{a\}$ is the element of $\boldsymbol{R}$, $\{a\}^*$ is also the element of $\boldsymbol{R}$
- $\sum^*$,which is the set of strings consisting of a's and b's, is also the element of $\boldsymbol{R}$ because $\{a, b\}$ is the element of $\boldsymbol{R}$

Regular Expressions

- A regular language has an explicit formula
  - A **regular expression** for a language is a slightly more user-friendly formula
- Parentheses **()** replace curly braces **{}**, and are used only when needed, and the union symbol is replaced by **+**

Regular language	***Regular Expression***
$\varnothing$	$\varnothing$
$\{\varepsilon\}$	ε
$\{a,b\}^*$	$(a+b)^*$
$\{aab\}^*\{a,ab\}$	$(aab)^*(a+ab)$

Regular Expressions

- Two regular expressions are equal if the languages they describe are equal
- $(a^*b^*)^* = (a+b)^*$
  - Both represent the language of all strings over the alphabet $\{a, b\}$
- $(a+b)^*ab(a+b)^*+b^*a^* = (a+b)^*$
  - The first half of the left-hand expression describes the strings that contain the substring *ab* and the second half describes those that don't
- A regular expression is not unique for a language
  - In general, a regular language, corresponds to more than one regular expressions

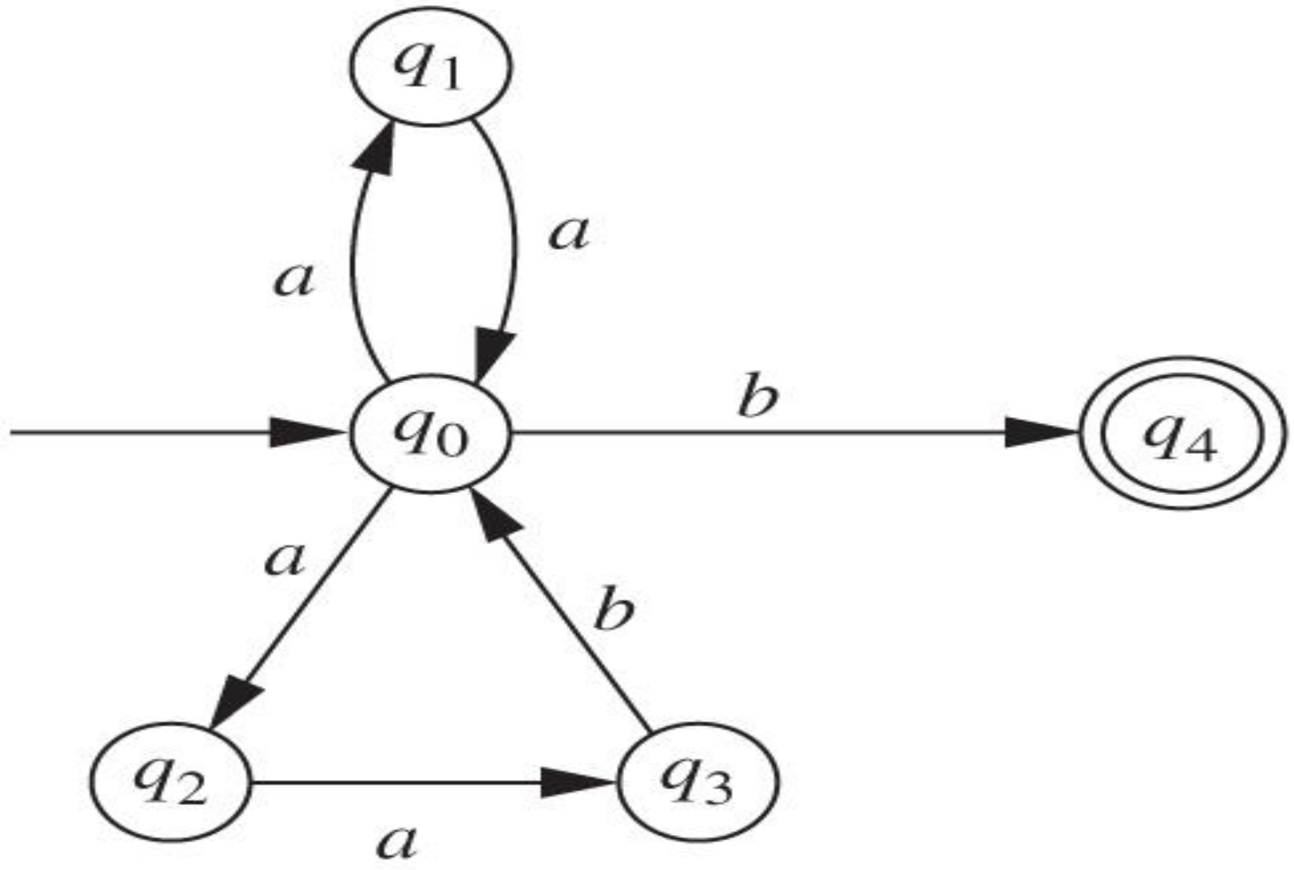
Exercises

- Find the shortest string that is not in the language represented by the regular expression $a^*(ab)^*b^*$
- For the two regular expressions given below
  - Find a string corresponding to r_2 but not to r_1
  - Find a string corresponding to both r_1 and r_2

$$r_1 = a^* + b^* \qquad r_2 = ab^* + ba^* + b^*a + (a^*b)^*$$

- Find a regular expression corresponding to the language of all strings over the alphabet $\{ a, b \}$ that contain exactly two a's
- Find a regular expression corresponding to the language of all strings over the alphabet $\{ a, b \}$ that do not end with ab

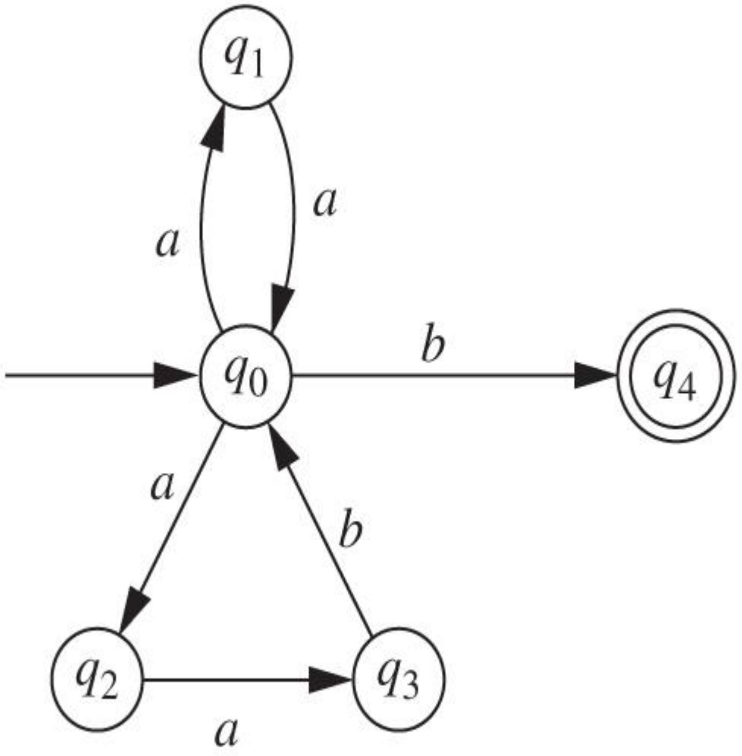
Anything Strange Here?



Nondeterministic Finite Automata

- Kleene's Theorem asserts that regular languages are precisely the languages accepted by **finite automata**
- To prove this, we introduce a more general "device," a **nondeterministic** finite automaton (**NFA**)
- These make it much easier to start with a **regular expression** and draw a **transition diagram** for a device that **accepts the corresponding language** and has an obvious connection to the regular expression
- We will show that allowing nondeterminism doesn't change the languages that can be accepted

Nondeterministic Finite Automata

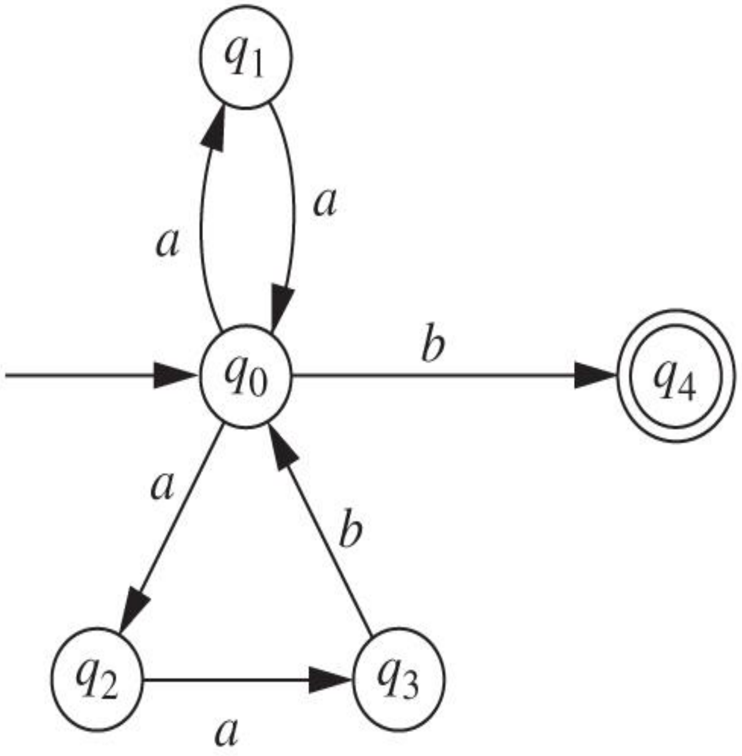


- This NFA closely resembles the regular expression $(aa + aab)^*b$
  - The top loop is aa
  - The bottom loop is aab
  - By following the links we can generate any string in the language
- This is **not** the transition diagram for a DFA; some nodes have more than one arc leaving them

Nondeterministic Finite Automata

- For NFA, the next state may be **nothing** or **two** or **more possible next** states for some **current** state and input symbol. In this case, we should not think of an NFA as describing an algorithm for recognizing a language
- Instead, consider it as describing a number of **different sequences of steps** that might be followed

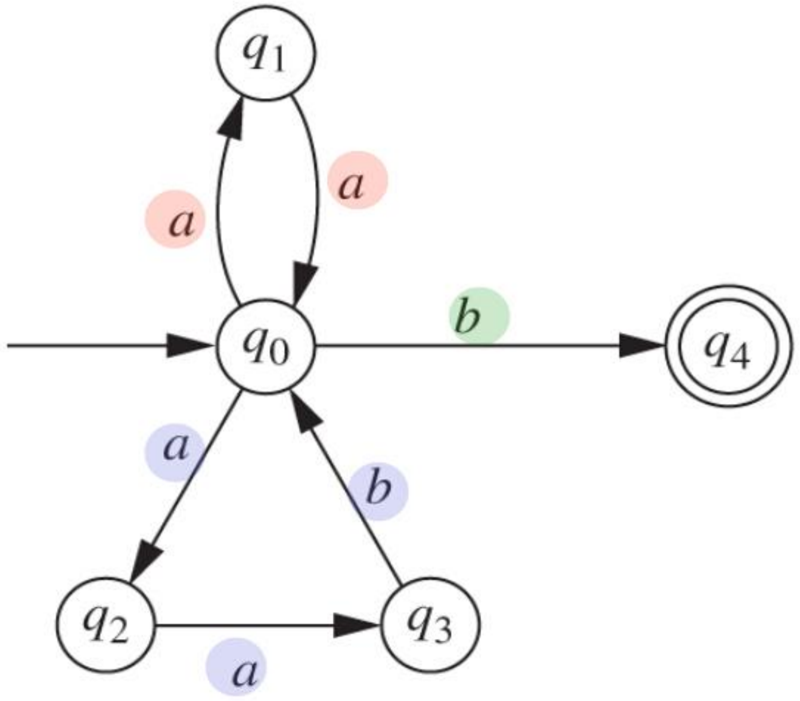
Nondeterministic Finite Automata



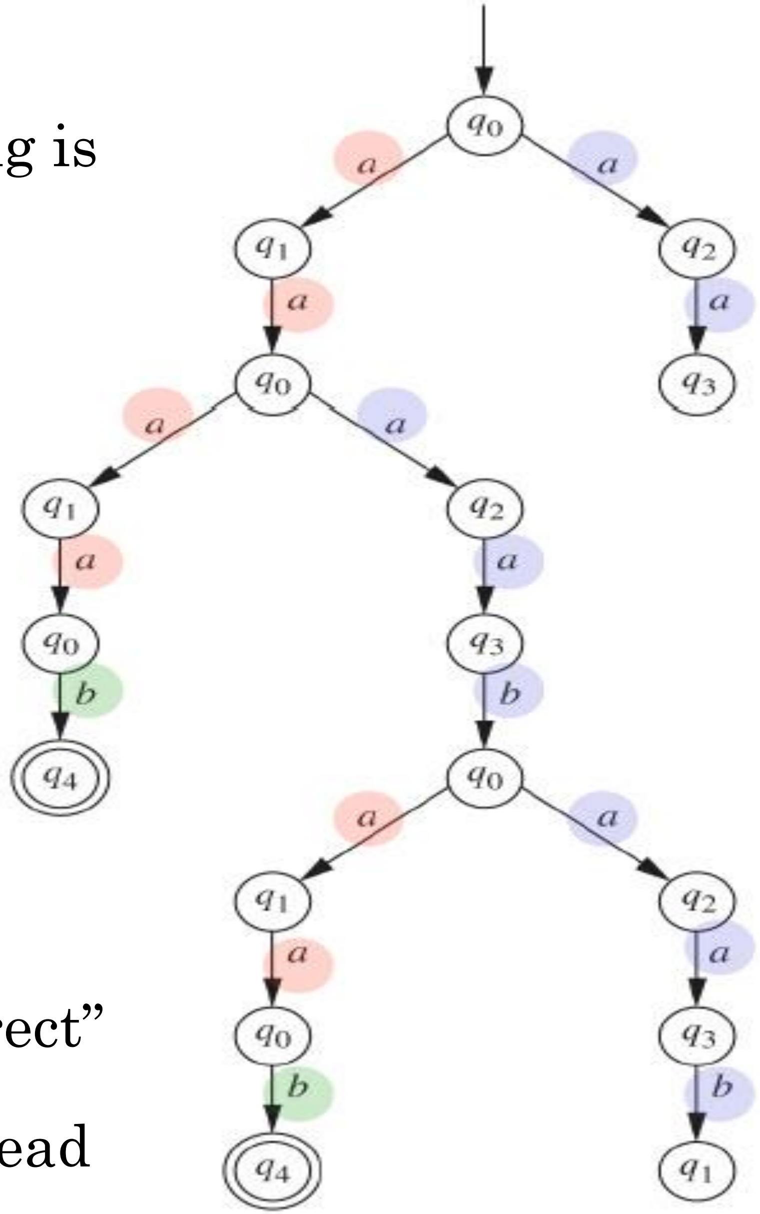
- Suppose we have an input string *aaaabaab* to process using the left NFA
- We can draw a computation tree to visualize the sequences
  - Each **level** corresponds to a prefix of the input string
  - Each **state** on a level is one the machine **could be** in after processing that prefix
  - There is an accepting path for the input string (as well as other paths that are not accepting)

Computation Tree

- One “correct” way to interpret the input string is $aaaabaab$



- The path in which the device makes the “correct” choice at each step ends up at the accepting state when all the input symbols have been read



NFA: Formal Definition

- An NFA is a 5-tuple $A = (Q,\Sigma,\delta,q_0,F)$ given by

1. A finite set of states Q
2. A finite set of input symbols (alphabets) Σ
3. A transition function δ: $Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^Q$
   - The values of δ are not single states, but sets of states, this is different with DFA transition function δ: $Q \times \Sigma \rightarrow Q$
4. An initial state $q_0 \in Q$
5. A set of accepting (or final) states $F \subseteq Q$

- For every element q of Q and every element s of $(\Sigma \cup \{\varepsilon\})$, we interpret $\delta(q, s)$ as the **set of states** to which the NFA can move from state q on input symbol s

Nondeterministic Finite Automata

- The transition function maps a state and an input symbol to **zero or more** successor states. Thus an NFA has "choice"; hence "nondeterministic"
- However, nothing ambiguous about the language defined by an NFA. **Not** the case that some word $w \in L(A)$ sometimes, and $w \notin L(A)$ other times for some NFA A.
  - How? By considering **all possible** states simultaneously
- Allowing nondeterminism doesn't change the languages that can be accepted

NFA: Extended Transition Function

- Defining δ^* is a little harder than for a DFA, as for the receiving input string x, $\delta^*(q, x)$ is a set of states, and for each state p in this set, $\delta(p, s)$ is also a set. Thus, in order to define $\delta^*(q, x.s)$, we need to include all the possibilities:

$$\cup \ \{\delta(p, s) \mid p \in \delta^*(q, x))$$

- Also, we need to consider ε-transitions, which allows the device to change state with **no input**
  - It could potentially occur at any stage

NFA: ε-Closure

- **Definition**: Suppose $A = (Q,\Sigma,\delta,q_0,F)$ is an NFA, and $S \subseteq Q$ is a set of states. The ε-closure of S is the set ε(S), which can be reachable from any state in S following ε-transitions. ε(S) can be defined recursively as follows:

$$S \subseteq \varepsilon(S)$$

$$\text{For every } q \in \varepsilon(S)\text{ , } \delta(q, \varepsilon) \subseteq \varepsilon(S)$$

- Algorithm to calculate ε(S)
  - Initialize T to be S, as in the basis part of the definition
  - Make a sequence of passes, in each pass considering every $q \in T$ and adding every state in $\delta(q, \varepsilon)$ not already there
  - Stop after the first pass in which T does not change
  - The final value of T is ε(S)
- A state is in ε(S), if it is the element of S, or can be reached from an element of S using one or more ε-transitions

NFA: Extended Transition Function

- **Definition**: Let $A = (Q,\Sigma,\delta,q_0,F)$ be an NFA
- Define the extended transition function $\delta^*: Q\times \sum^* \to 2^Q$ as follows:
  - For every $q \in Q$, $\delta^*(q, \varepsilon) = \varepsilon(\{q\})$
  - For every $q \in Q$, every $y \in \sum^*$, and every $s \in \sum$
    - $\delta^*(q, ys) = \varepsilon(\cup \{\delta(p, s) \mid p \in \delta^* (q, y)\})$
  - A string $x \in \sum^*$ is accepted by A if $\delta^*(q_0, x) \cap F \neq \varnothing$
    - i.e., some sequence of transitions involving the symbols of x and ε's leads from q_0 to an accepting state
- The language $L(A)$ accepted by A is the set of all strings accepted by A

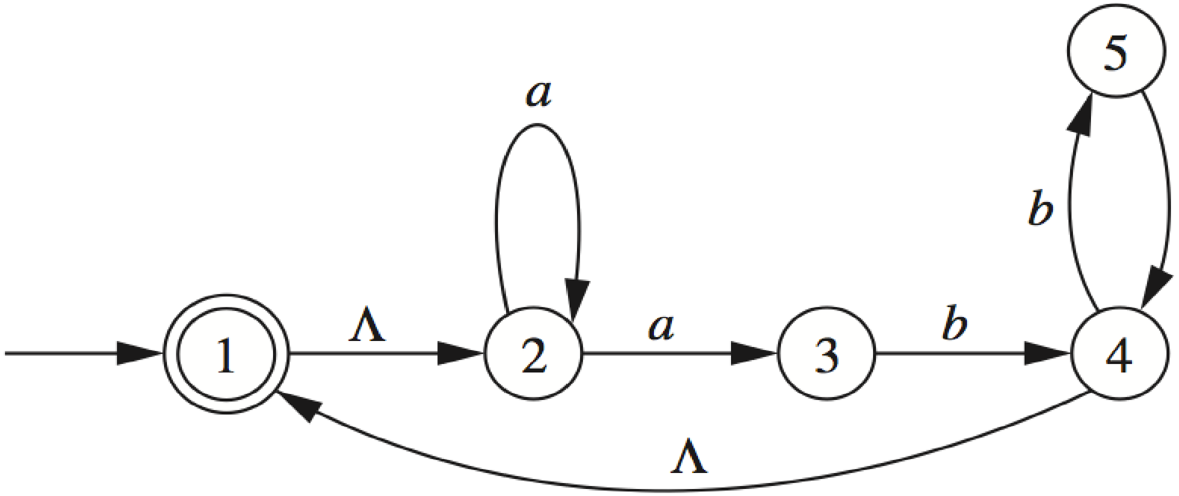
NFA: Eliminate ε-Transitions

- Two types of nondeterminism have arisen:
  - Different arcs for the same input symbol
  - ε-transitions
  - **Both** can be eliminated
- For the second type, introduce new transitions so that we no longer need the ε-transitions
- When there is no s-transition from p to q but the NFA can go from p to q by using one or more ε-transitions as well as s, we introduce the s-transition
- The resulting NFA may have more nondeterminism of the first type, but it will have no ε-transitions

NFA: Eliminate ε-Transitions

- **Theorem**: For every language $L \subseteq \sum^*$ accepted by an NFA $A = (Q,\Sigma,\delta,q_0,F)$, there is an NFA A_1 with no ε-transitions that also accepts L
- Define $A_1 = (Q,\Sigma,\delta_1,q_0,F_1)$, where
  - For every $q \in Q$, $\delta_1(q, \varepsilon) = \varnothing$
  - For every $q \in Q$ and every $s \in \sum$, $\delta_1(q, s) = \delta^*(q, s)$
- Define $F_1 = F \cup \{q_0\}$ if $\varepsilon \in L$, and $F_1 = F$ otherwise
- We can prove that, by structural induction on x, that for every q and every x with $|x| \geq 1$, $\delta_1{}^*(q, x) = \delta^* (q, x)$
  - pp 104-106

Eliminate ε-Transitions: Example

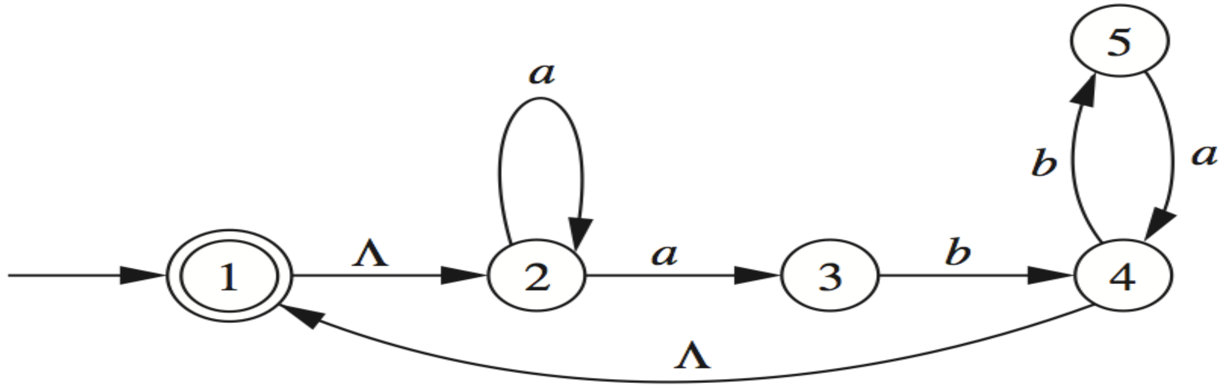


(*a*)

- Accepts the language corresponding to the regular expression

$$(a^*ab(ba)^*)^*$$

Eliminate ε-Transitions: Example



(a)

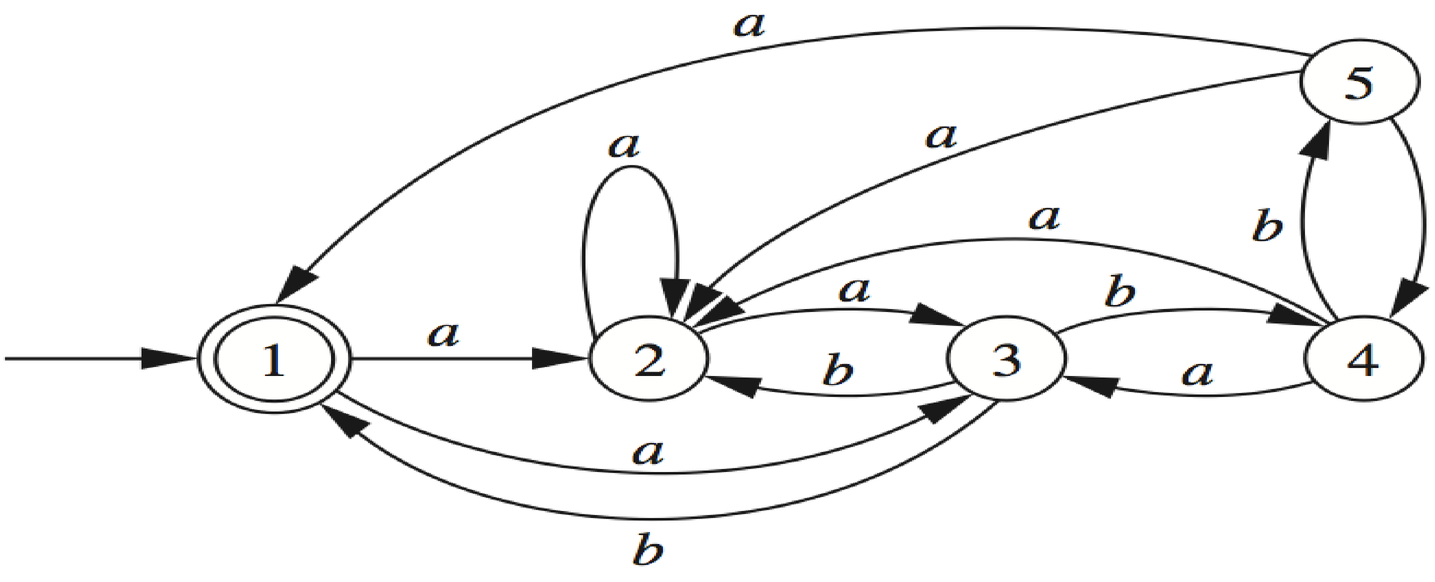
q	$\delta(q,a)$	$\delta(q,b)$	$\delta(q,\Lambda)$	$\delta^*(q,a)$	$\delta^*(q,b)$
1	∅	∅	{2}	{2, 3}	∅
2	{2, 3}	∅	∅	{2, 3}	∅
3	∅	{4}	∅	∅	{1, 2, 4}
4	∅	{5}	{1}	{2, 3}	{5}
5	{4}	∅	∅	{1, 2, 4}	∅

- Where $\delta^*(q,s)$ shows all the states that can be reached from q using either **one** s-transition or the **combination** of **one** s-transition and (**possibly more than one**) ε-transitions

Eliminate ε-Transitions: Example

q	$\delta(q,a)$	$\delta(q,b)$	$\delta(q,\Lambda)$	$\delta^*(q,a)$	$\delta^*(q,b)$
1	∅	∅	{2}	{2, 3}	∅
2	{2, 3}	∅	∅	{2, 3}	∅
3	∅	{4}	∅	∅	{1, 2, 4}
4	∅	{5}	{1}	{2, 3}	{5}
5	{4}	∅	∅	{1, 2, 4}	∅

- Draw the new transitions using the information in $\delta^*(q,s)$



(b)

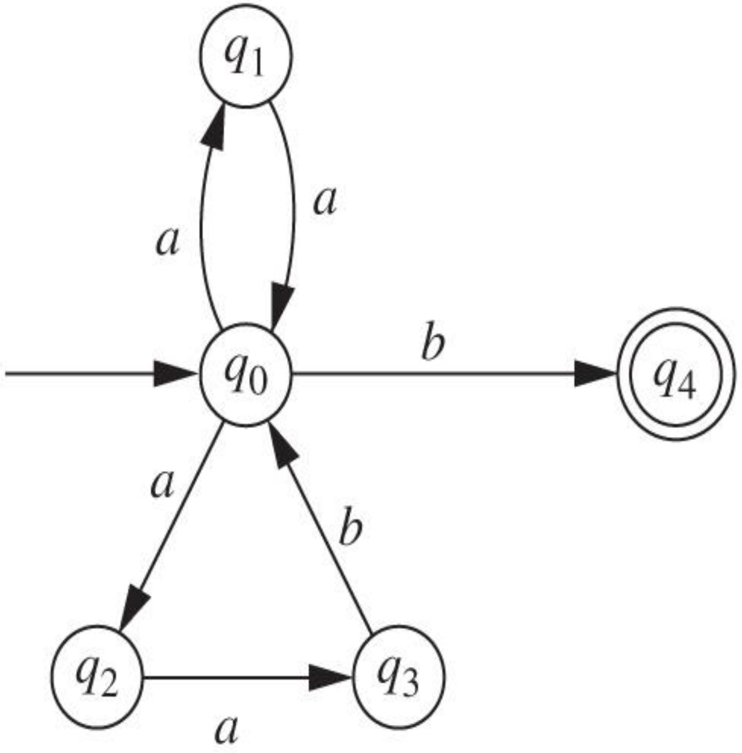
NFA: Observations

- An NFA can be in one of a **set** of states
- When reading an input symbol, the machine enters one of a **new set** of states
- Each set is a subset of Q, so the set of possible states is (**at most**) $P(Q)$
  - Q is **finite**. Thus $P(Q)$ is **finite** too
- There may be lots of states as $|P(Q)| = 2^{|Q|}$, but the number of states is finite
- We can thus convert an NFA into a DFA by considering each possible set of NFA states as a **single** DFA state
  - Known as subset construction

NFA: Subset Construction

- **Theorem**: For every language $L \subseteq \Sigma^*$ accepted by an NFA $A = (Q,\Sigma,\delta,q_0,F)$, there is a DFA $A_1 = (Q_1,\Sigma,\delta_1,q_1,F_1)$ that also accepts L
- **Subset construction**: we can assume A has no ε-transitions. Let $Q_1 = 2^Q$ (i.e. the set of all possible states of A), $q_1 = \{q_0\}$
  - For every $q \in Q_1$ and every $s \in \Sigma$, $\delta_1(q, s) = \cup\{\delta(p, s) \mid p \in q\}$
  - $F_1 = \{q \in Q_1 \mid q \cap F \neq \emptyset\}$
- A_1 is clearly a DFA
  - It accepts the same language as A because for every $x \in \Sigma^*$, $\delta_1^*(q_1, x) = \delta^*(q_0, x)$

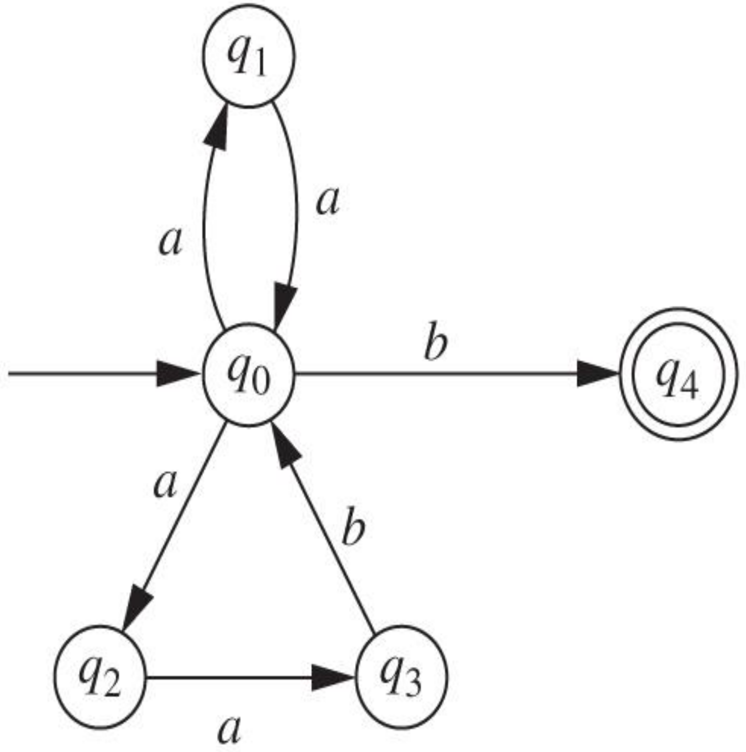
Subset Construction: Example



- Accepts the language corresponding to the regular expression

$$(aa + aab)^*b$$

Subset Construction: Example

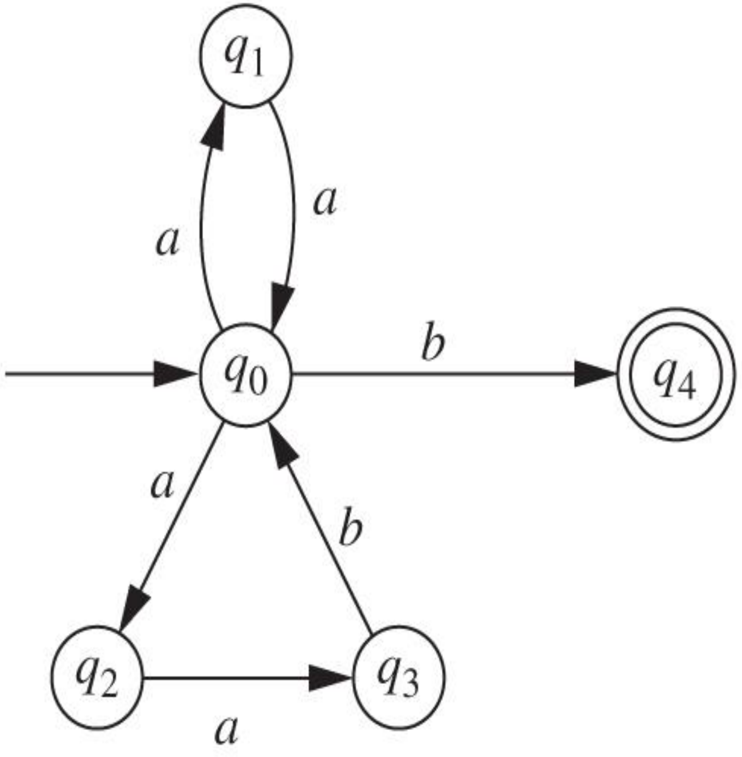


q	$\delta(q, a)$	$\delta(q, b)$
0	{1, 2}	{4}
1	{0}	∅
2	{3}	∅
3	∅	{0}
4	∅	∅

- If an NFA has n states, the equivalent DFA may have at most 2^n states
  - However, many are typically unreachable
  - Save work by only considering reachable states

Subset Construction: Example

- We may only consider the reachable states

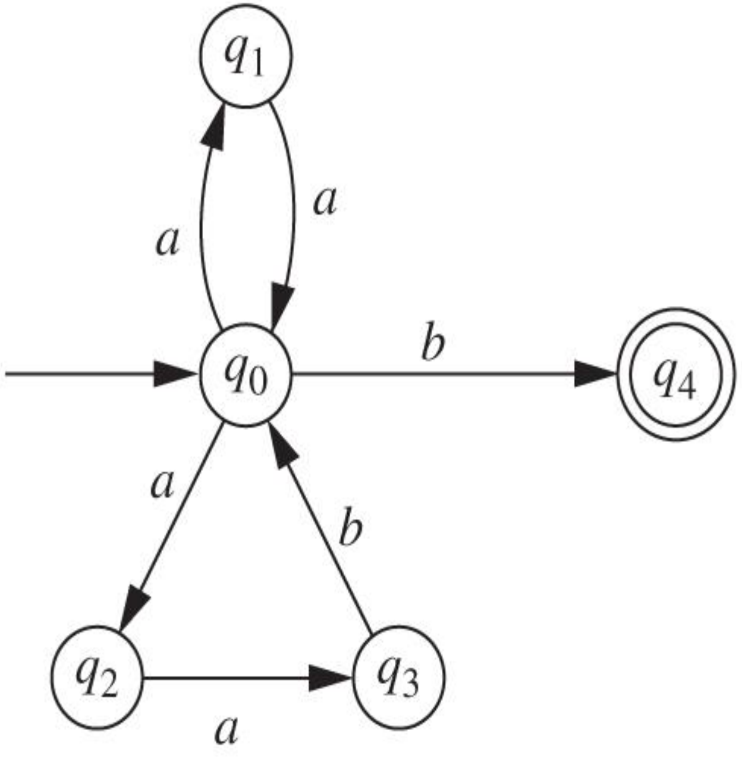


q	$\delta(q,a)$	$\delta(q,b)$
0	{1, 2}	{4}
1	{0}	Ø
2	{3}	Ø
3	Ø	{0}
4	Ø	Ø

q	$\delta(q,a)$	$\delta(q,b)$
0	{1,2}	{4}
{1,2}	{0,3}	$\varnothing$
{4}	$\varnothing$	$\varnothing$
{0,3}	{1,2}	{0,4}
$\varnothing$	$\varnothing$	$\varnothing$
{0,4}	{1,2}	{4}

Subset Construction: Example

- We may only consider the reachable states

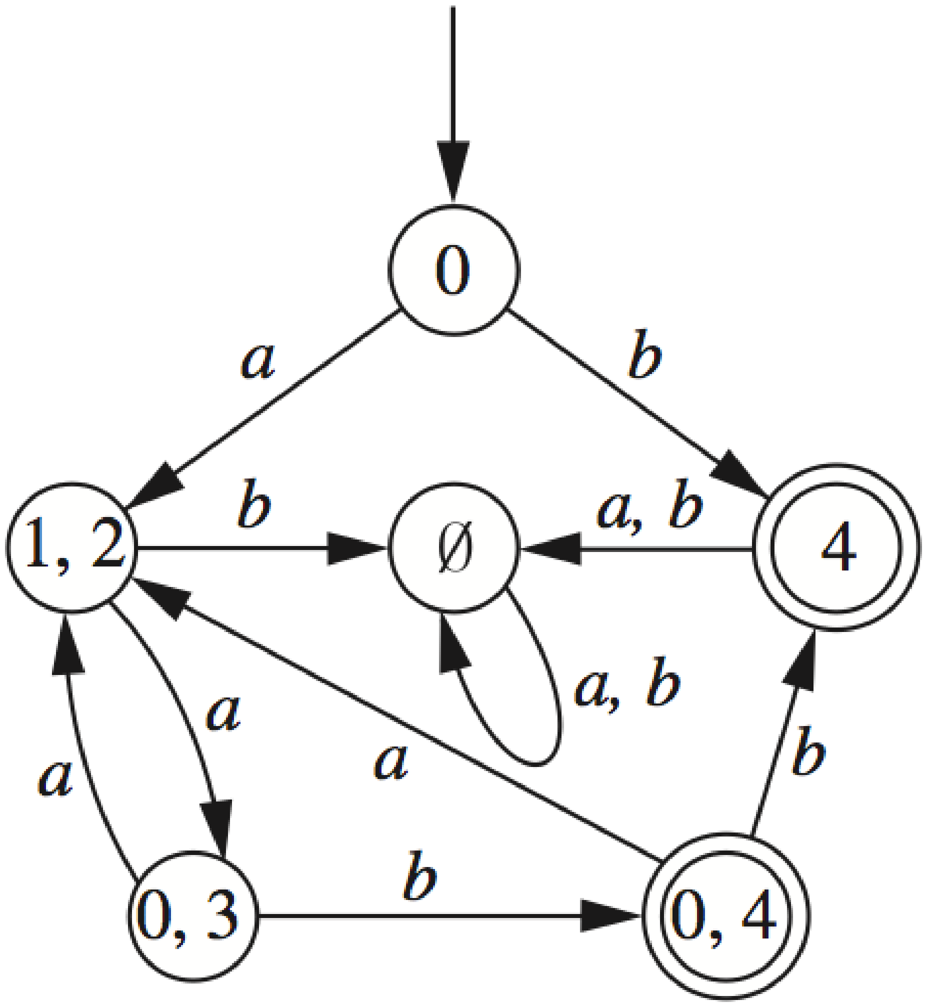


q	$\delta(q, a)$	$\delta(q, b)$
0	{1, 2}	{4}
1	{0}	Ø
2	{3}	Ø
3	Ø	{0}
4	Ø	Ø

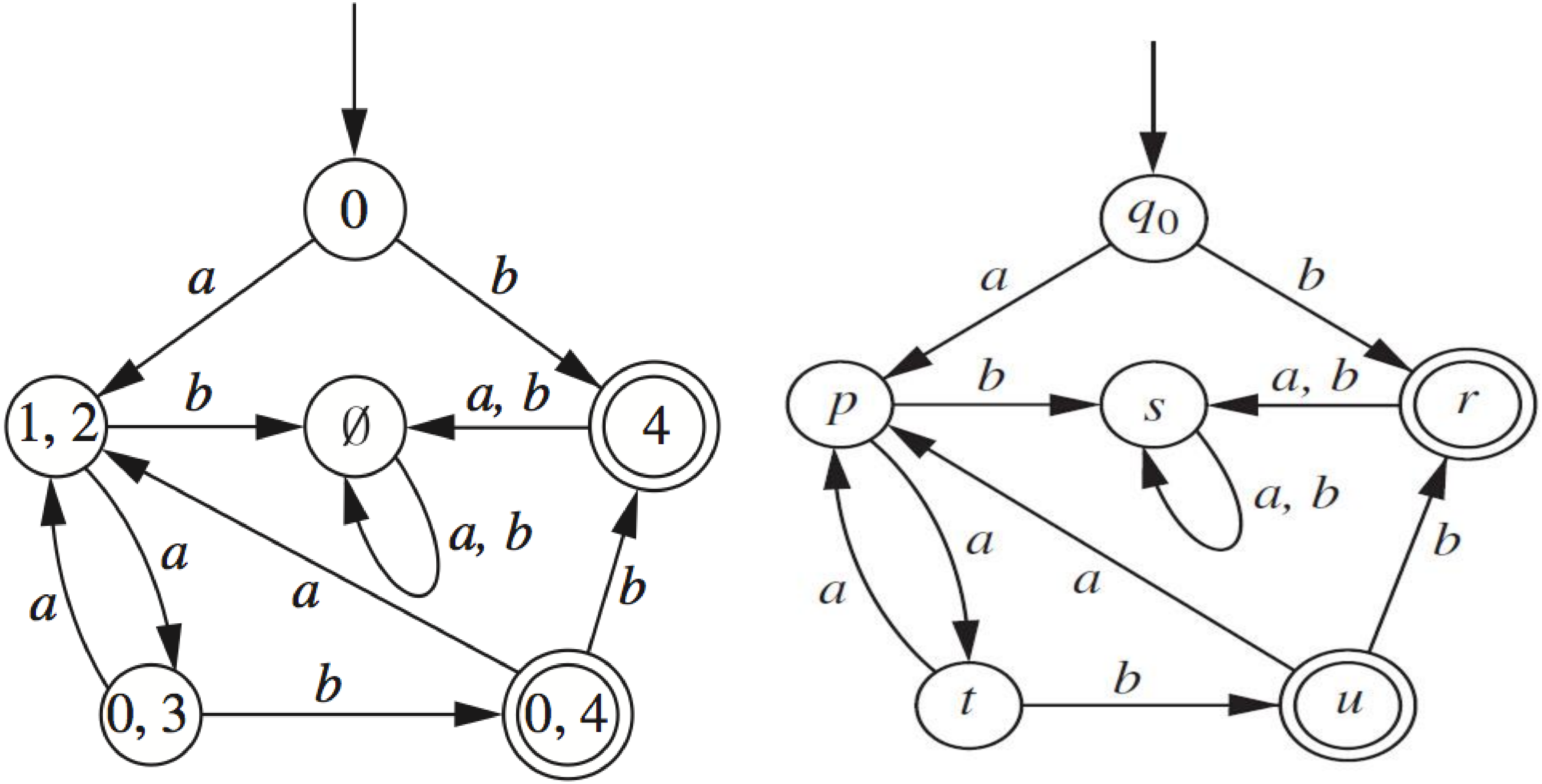
q	$\delta(q,a)$	$\delta(q,b)$
→0	{1,2}	{4}
{1,2}	{0,3}	$\varnothing$
*{4}	$\varnothing$	$\varnothing$
{0,3}	{1,2}	{0,4}
$\varnothing$	$\varnothing$	$\varnothing$
*{0,4}	{1,2}	{4}

Subset Construction: Example

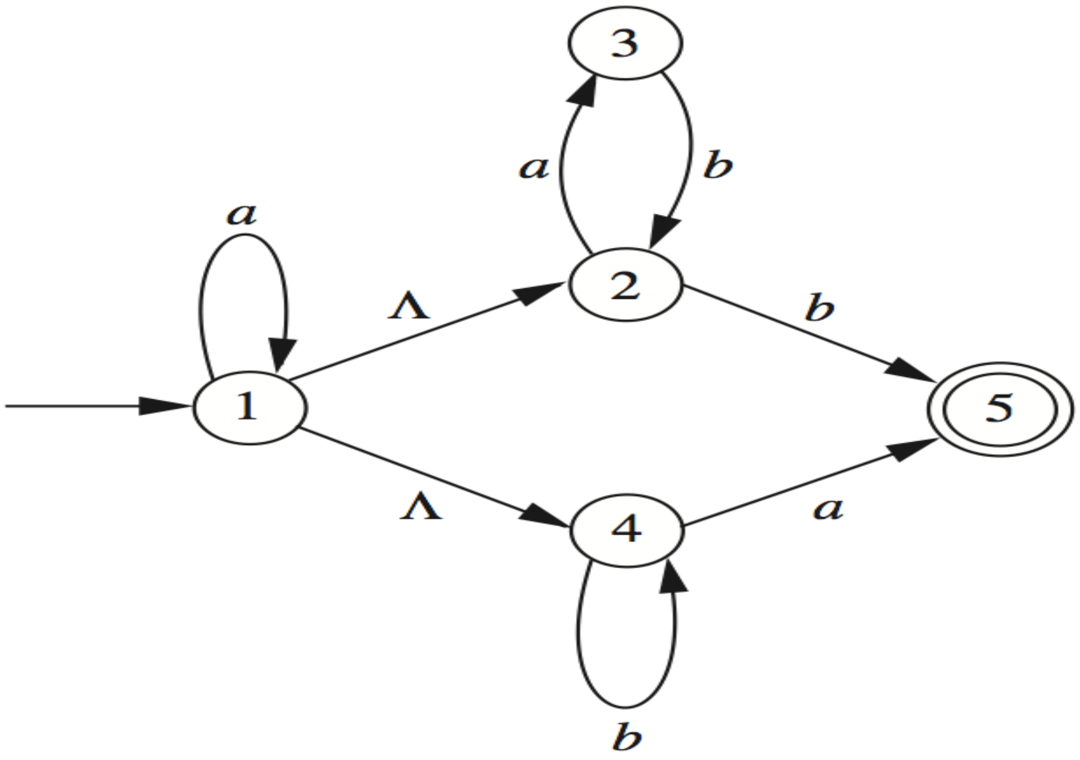
q	$\delta(q,a)$	$\delta(q,b)$
$\to 0$	$\{1,2\}$	$\{4\}$
$\{1,2\}$	$\{0,3\}$	$\varnothing$
$*\{4\}$	$\varnothing$	$\varnothing$
$\{0,3\}$	$\{1,2\}$	$\{0,4\}$
$\varnothing$	$\varnothing$	$\varnothing$
$*\{0,4\}$	$\{1,2\}$	$\{4\}$



Subset Construction: Example



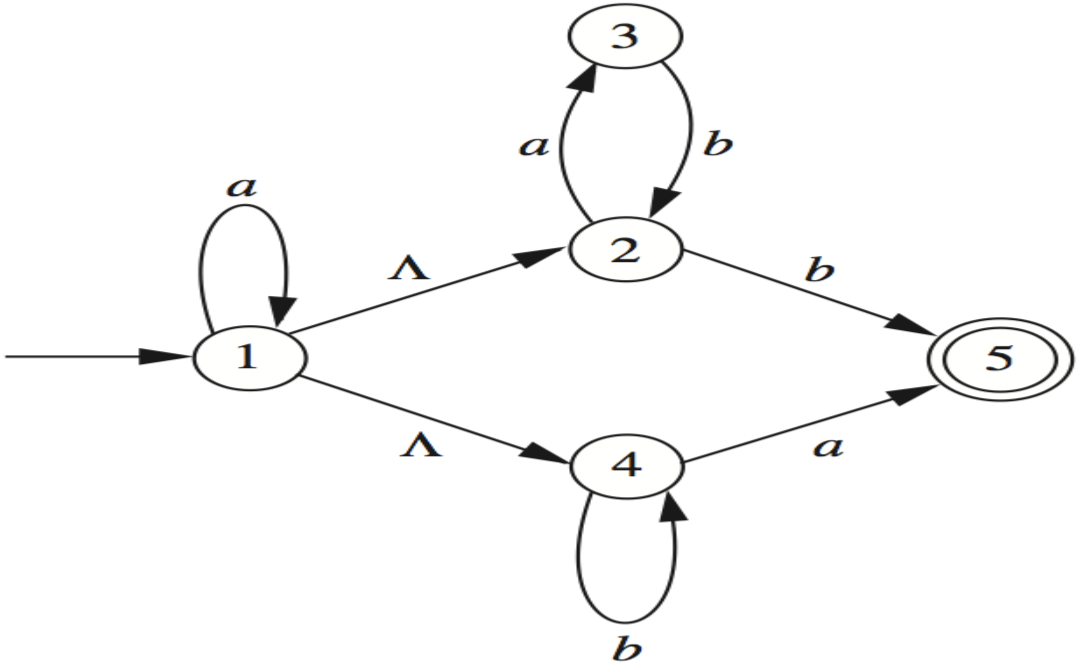
Putting All Together



- Accepts the language corresponding to the regular expression

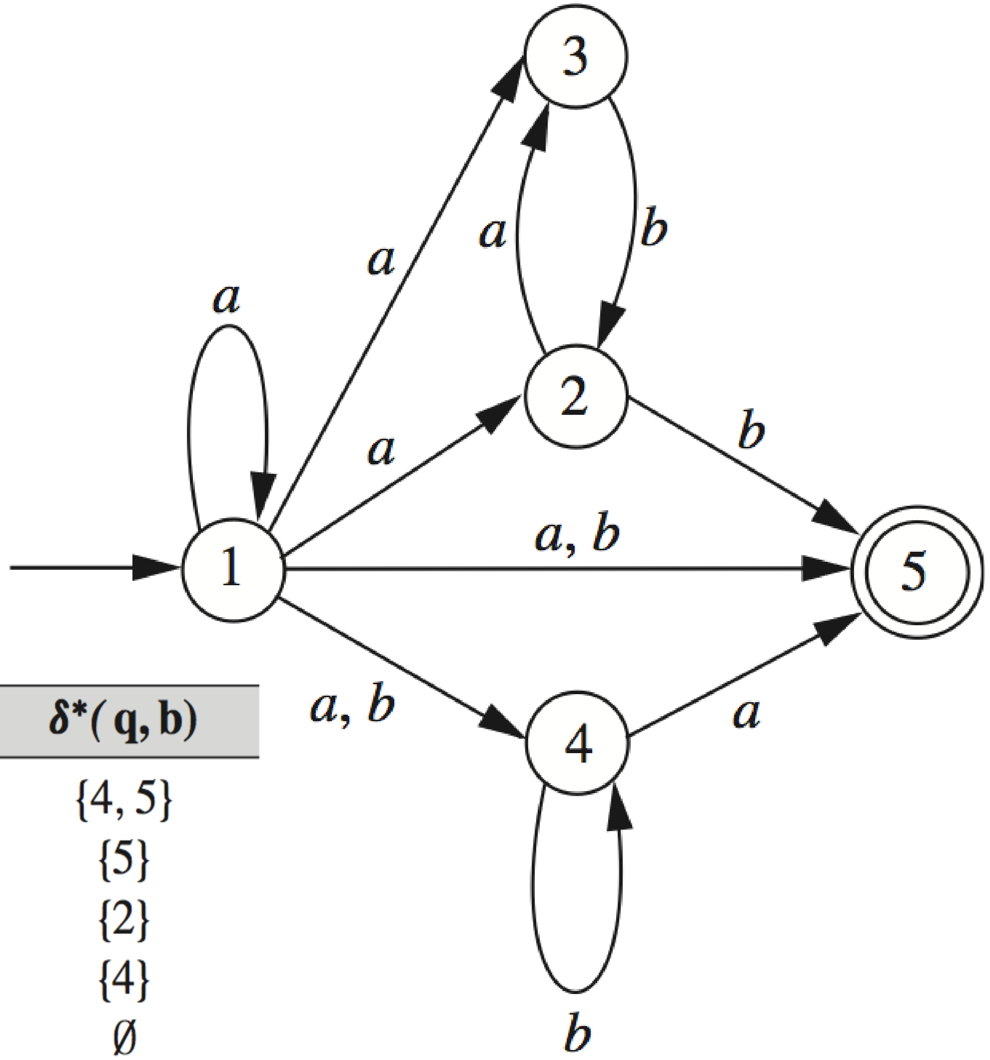
$$a^*((ab)^*b + b^*a)$$

Draw Transition Table



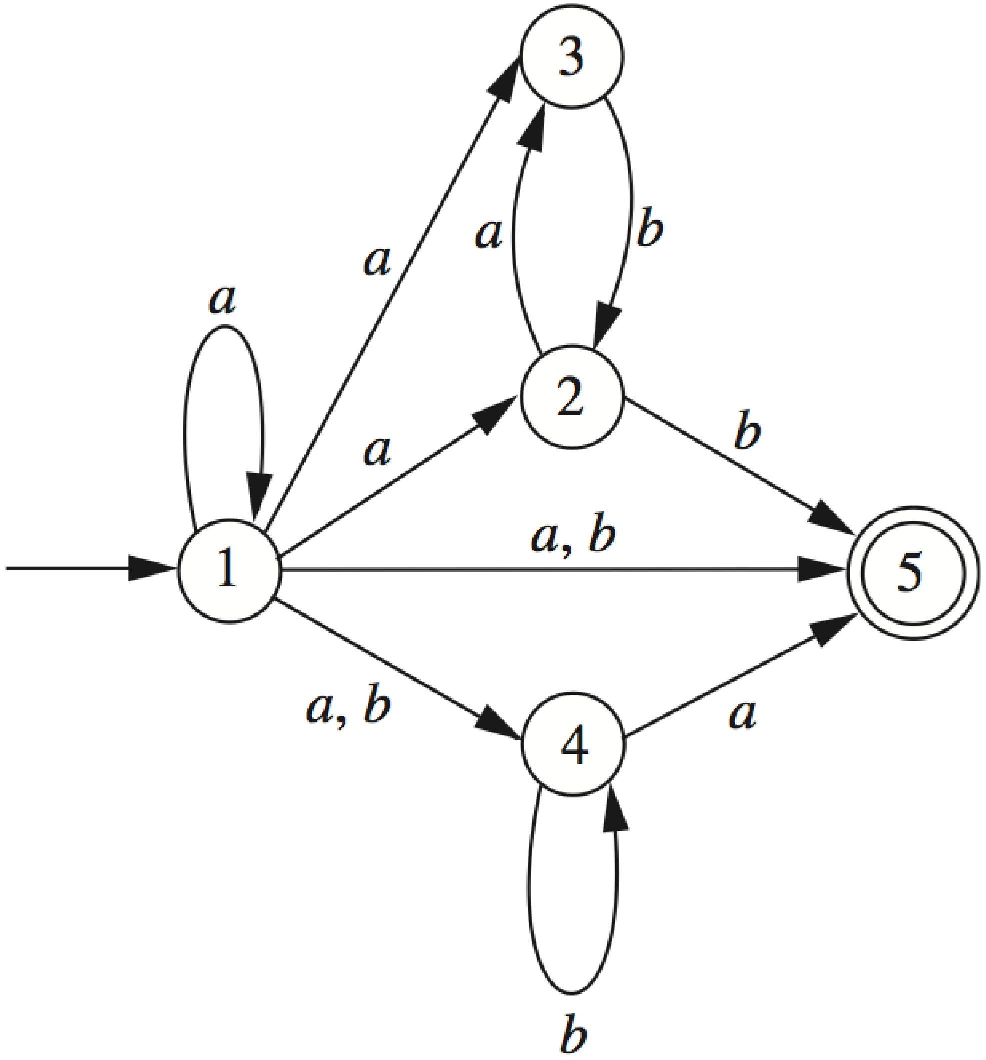
q	$\delta(q, a)$	$\delta(q, b)$	$\delta(q, \Lambda)$	$\delta^*(q, a)$	$\delta^*(q, b)$
1	{1}	∅	{2, 4}	{1, 2, 3, 4, 5}	{4, 5}
2	{3}	{5}	∅	{3}	{5}
3	∅	{2}	∅	∅	{2}
4	{5}	{4}	∅	{5}	{4}
5	∅	∅	∅	∅	∅

Eliminate Null-Transitions



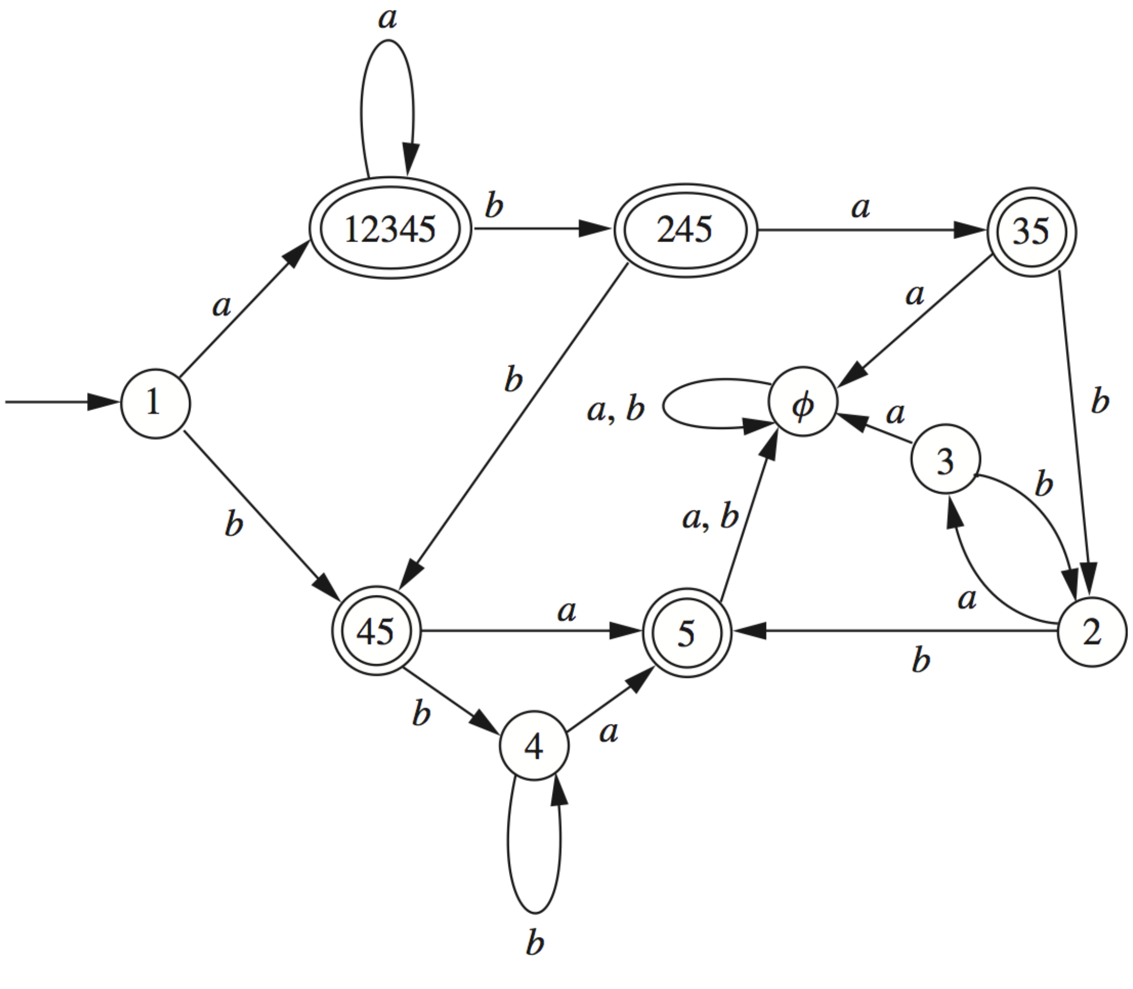
q	$\delta(q, a)$	$\delta(q, b)$	$\delta(q, \Lambda)$	$\delta^*(q, a)$	$\delta^*(q, b)$
1	{1}	∅	{2, 4}	{1, 2, 3, 4, 5}	{4, 5}
2	{3}	{5}	∅	{3}	{5}
3	∅	{2}	∅	∅	{2}
4	{5}	{4}	∅	{5}	{4}
5	∅	∅	∅	∅	∅

Subset Construction



q	$\delta(q,a)$	$\delta(q,b)$
→{1}	{1,2,3,4,5}	{4,5}
*{1,2,3,4,5}	{1,2,3,4,5}	{2,4,5}
*{4,5}	{5}	{4}
*{2,4,5}	{3,5}	{4,5}
{4}	{5}	{4}
*{5}	∅	∅
*{3,5}	∅	{2}
{2}	{3}	{5}
{3}	∅	{2}
∅	∅	∅

Resulting Equivalent DFA



(c)

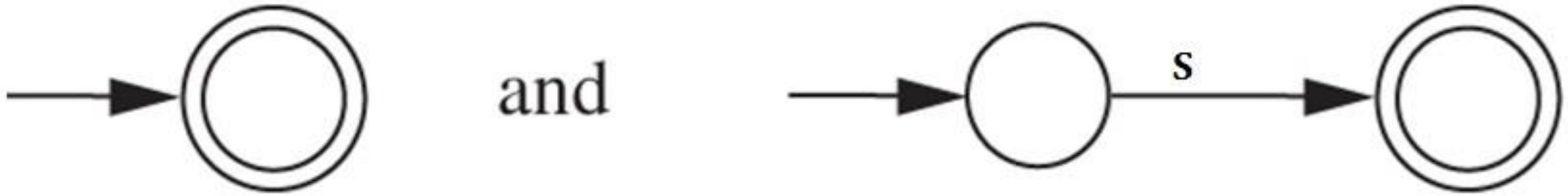
q	$\delta(q,a)$	$\delta(q,b)$
$\rightarrow\{1\}$	$\{1,2,3,4,5\}$	$\{4,5\}$
$*\{1,2,3,4,5\}$	$\{1,2,3,4,5\}$	$\{2,4,5\}$
$*\{4,5\}$	$\{5\}$	$\{4\}$
$*\{2,4,5\}$	$\{3,5\}$	$\{4,5\}$
$\{4\}$	$\{5\}$	$\{4\}$
$*\{5\}$	$\varnothing$	$\varnothing$
$*\{3,5\}$	$\varnothing$	$\{2\}$
$\{2\}$	$\{3\}$	$\{5\}$
$\{3\}$	$\varnothing$	$\{2\}$
$\varnothing$	$\varnothing$	$\varnothing$

Kleene's Theorem, Part 1

- **Theorem**: For every alphabet ∑, every regular language over ∑ can be accepted by a finite automaton
- Because of what we have just shown, it is enough to show that every regular language over ∑ can be accepted by an NFA
- The proof is by structural induction, based on the recursive definition of the set of regular languages over ∑

Kleene's Theorem, Part 1

- The basis cases are easy
- The machines pictured below accept the languages ∅ and {s}, respectively



Kleene's Theorem, Part 1

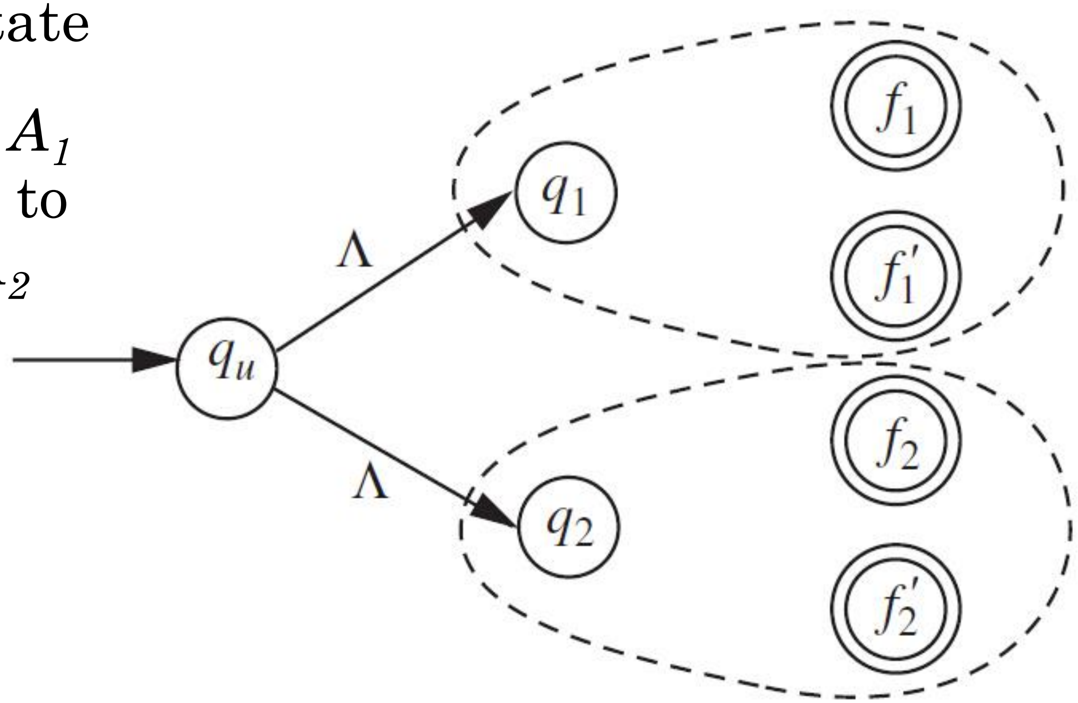
- The induction hypothesis is that, suppose L_1 and L_2 are both regular languages over $\sum$ for both i =1 and i = 2, L_i can be accepted by an NFA $A_i = (Q_i,\Sigma,\delta_i,q_i,F_i)$
- For the induction step, we need to show there are NFAs accepting the three languages

1. $L(A_1) \cup L(A_2)$
2. $L(A_1)\, L(A_2)$
3. $L(A_1)^*$

- For simplicity, we assume there are two NFAs, A_1 and A_2, accepting $L(A_1)$ and $L(A_1)$, respectively. Each of A_1 and A_2 may have two accepting states, both distinct from the initial state

Kleene's Theorem, Part 1

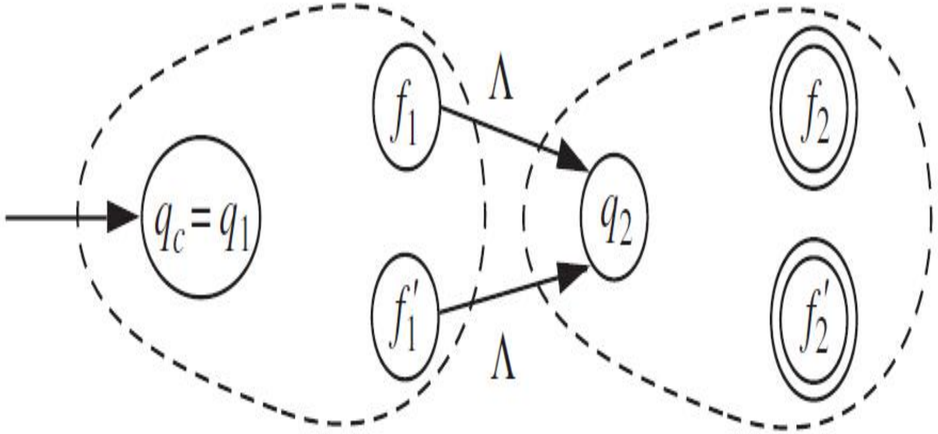
- This NFA, A_u, accepts the language $L(A_1) \cup L(A_2)$
- Its states are those of A_1 and A_2 and one additional state q_u that is the initial state
- The transitions include all the ones in A_1 and A_2 as well as ε-transitions from q_u to q_1 and q_2, the initial states of A_1 and A_2
- The accepting states are simply the states in $F_1 \cup F_2$



(a)

Kleene's Theorem, Part 1

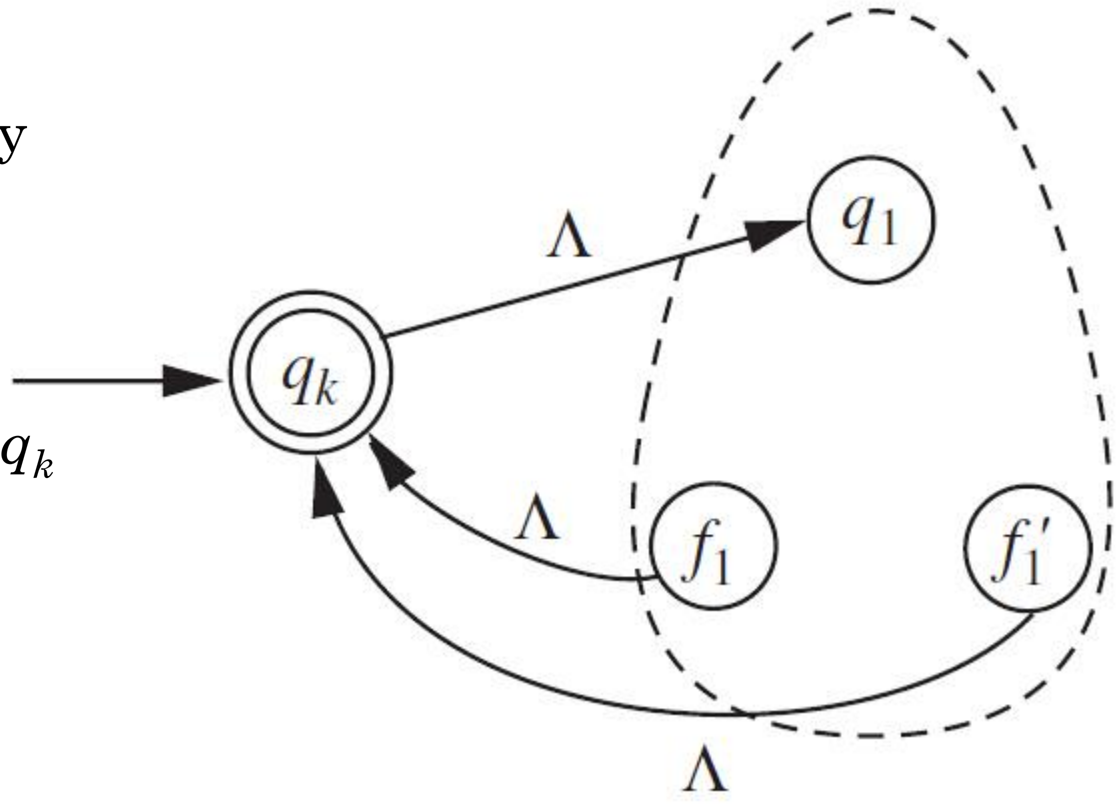
- This NFA, A_c, accepts the language $L(A_1)L(A_2)$
- The initial state is q_1, and the accepting states are the elements of F_2
- The transitions include all those of A_1 and A_2 and a new ε-transition from every element of A_1 to q_2



(b)

Kleene's Theorem, Part 1

- This NFA, A_k, accepts the language $L(A_1)^*$
- Its states are the elements of Q_1 and a new initial state q_k that is also the only accepting state
- The transitions are those of A_1, a ε-transition from q_k to q_1, and a ε-transition from every element of F_1 to q_k



(c)

Kleene’s Theorem, Part 1

- By using these constructions, we can create for **every** regular expression an NFA that accepts the corresponding language

Kleene's Theorem, Part 2

- **Theorem**: For every finite automaton $A = (Q,\Sigma,\delta,q_0,F)$, the language $L(A)$ is regular
- ***Proof***:
- First, for two states p and q, we define notation for the language

$$L(p, q) = \{x \in \sum^* \mid \delta^*(p, x) = q\}$$

- If we can show that for every p and q in Q, $L(p, q)$ is regular, then it will follow that $L(A)$ is, because
  - $L(A) = \cup \{L(q_0, q) \mid q \in F\}$
  - The union of a finite collection of regular languages is regular
- We will show that $L(p, q)$ is regular by expressing it in terms of simpler languages that are regular

Kleene's Theorem, Part 2

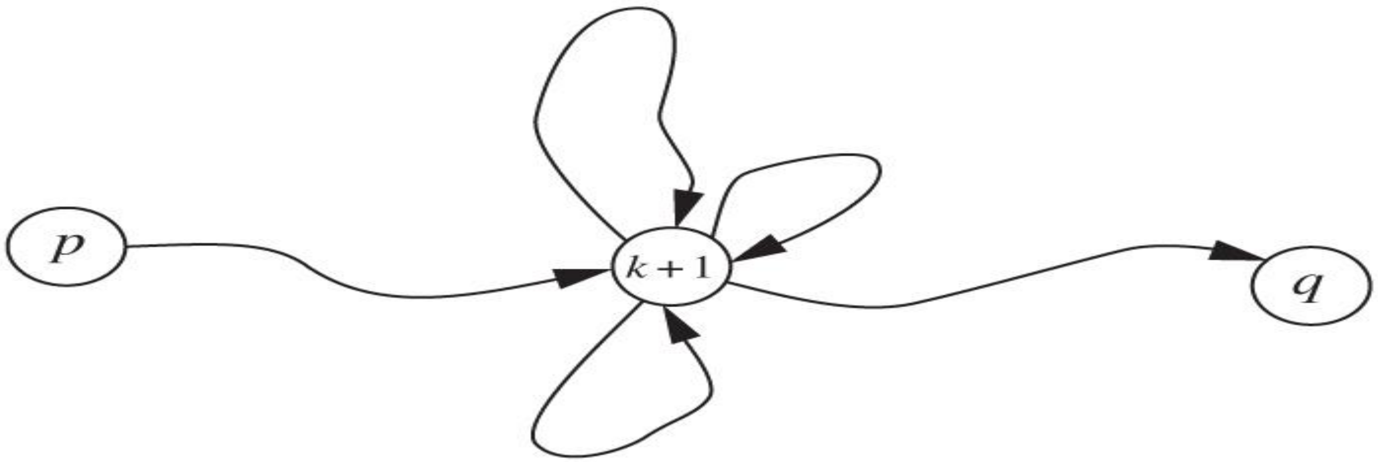
- We will consider the distinct states through which A passes as it moves from p to q
- If $x \in L(p, q)$, we say x causes A to go from p to q through a state r if there are non-null strings x_1 and x_2 such that $x = x_1x_2$, $\delta^*(p, x_1) = r$, and $\delta^*(r, x_2) = q$
  - In using a string of length 1 to go from p to q, A does not go through any state
  - In using a string of length $n \geq 2$, it goes through a state $n - 1$ times (but if $n > 2$, these states may not be distinct)

Kleene's Theorem, Part 2

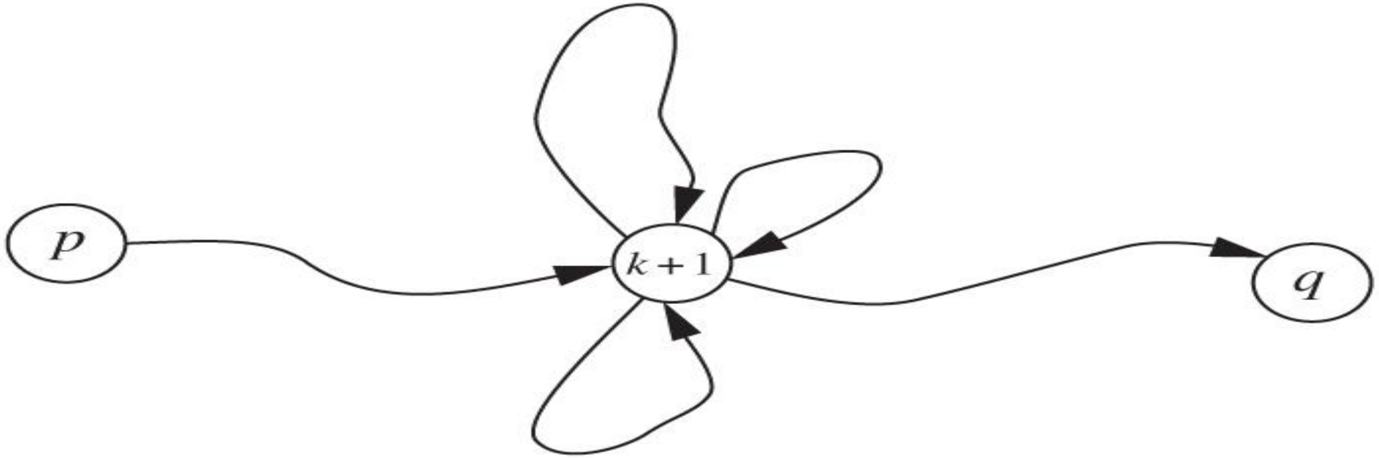
- Assume Q has n elements numbered 1 to n
- For p, $q \in Q$ and $j \geq 0$, we let $L(p, q, j)$ be the set of strings in $L(p, q)$ that cause A to go from p to q without going through any state numbered higher than j
- The set $L(p, q, 0)$ is the set of strings that allow A to go from p to q without going through any state at all
  - Includes the set of alphabet symbols s for which $\delta(p,s) = q$
  - And in the case when $p = q$
  - In any case, $L(p, q, 0)$ is a finite set of strings and therefore regular
- Suppose that for some number $k \geq 0$, $L(p, q, k)$ is regular for every p, $q \in Q$ and consider how a string can be in $L(p, q, k+1)$

Kleene's Theorem, Part 2

- Suppose that for some number $k \geq 0$, $L(p, q, k)$ is regular for every $p, q \in Q$ and consider how a string can be in $L(p, q, k{+}1)$
- The easiest way is for it to be in $L(p, q, k)$
- If not, it causes A to go to k+1 one or more times, but A goes through nothing higher
  - It can go from p to $k + 1$; it may return to $k + 1$ one or more times; and it finishes by going from $k + 1$ to q



Kleene's Theorem, Part 2



- On each of these individual portions, the path starts or stops at state $k + 1$ but doesn't go through any state numbered higher than k
- Every string in $L(p, q, k+1)$ can be described in one of those two ways and every string that has one of these two forms is in $L(p, q, k+1)$. This leads to the formula

$$L(p, q, k+1) = L(p, q, k) \cup L(p, k+1, k)\, L(k+1, k+1, k)^*\, L(k+1, q, k)$$

Concluding Remarks

- Regular languages and regular expressions
- Nondeterministic finite automata
  - Eliminate ε-transitions
  - Convert NFA into DFA
- Kleene's theorem