4. NONDETERMINISM AND KLEENE'S THEOREM

4.1 Nondeterministic Finite Automata

- Finding a finite automaton (FA) corresponding to a given regular expression can be difficult.

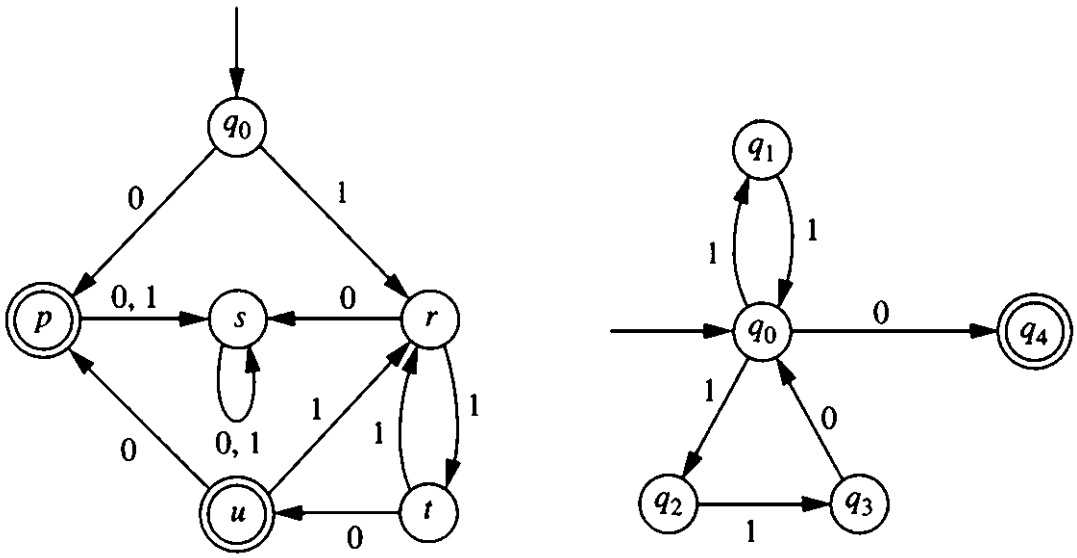
- It is often much easier to construct a nondeterministic finite automaton (NFA).

- NFAs accept exactly the same languages as FAs, and there is a straightforward procedure for converting an NFA to an equivalent FA.

An Example of a Nondeterministic Finite Automaton

- **Example 4.1:** A Simpler Approach to Accepting {11, 110}*{0}

  The FA on the left recognizes the language corresponding to the regular expression (11 + 110)*0. The diagram on the right does not satisfy normal FA rules, but reflects much more clearly the structure of the regular expression.



  Differences between the second diagram and a normal FA:

  Some transitions are missing.
  One state (q_0) has two transitions on the same input symbol.

  In state q_0 with input symbol 1, the machine "guesses" which move to make. This is called *nondeterminism*.

  A nondeterministic machine accepts an input string if *some* choice of moves corresponding to that string leads to an accepting state.

The Formal Definition of a Nondeterministic Finite Automaton

- The formal definition of a nondeterministic finite automaton is almost the same as the definition of an ordinary FA. The only change is that values of the transition function are sets of states rather than single states.

- **Definition 4.1:** A nondeterministic finite automaton, abbreviated NFA, is a 5-tuple $M = (Q, \Sigma, q_0, A, \delta)$, where Q and Σ are nonempty finite sets, $q_0 \in Q$, $A \subseteq Q$, and

  $\delta : Q \times \Sigma \rightarrow 2^Q$

  Q is the set of states, Σ is the alphabet, q_0 is the initial state, and A is the set of accepting states.

Extending the Transition Function

- Just as in the case of FAs, it is useful to extend the transition function δ from $Q \times \Sigma$ to the larger set $Q \times \Sigma^*$.

- For an NFA M, δ^* should be defined so that $\delta^*(p, x)$ is the set of states M can legally be in as a result of starting in state p and processing the symbols in the string x.

- **Definition 4.2a:** For an NFA $M = (Q, \Sigma, q_0, A, \delta)$, and any $p \in Q$, $\delta^*(p, \Lambda) = \{p\}$. For any $p \in Q$ and any $x = a_1a_2\ldots a_n \in \Sigma^*$ (with $n \geq 1$), $\delta^*(p, x)$ is the set of all states q for which there is a sequence of states $p = p_0, p_1, \ldots, p_{n-1}, p_n = q$ satisfying

  $p_i \in \delta(p_{i-1}, a_i)$ for each i with $1 \leq i \leq n$

- It is also useful to have a recursive definition of δ^*.

- **Definition 4.2b**: Let $M = (Q, \Sigma, q_0, A, \delta)$ be an NFA. The function $\delta^* : Q \times \Sigma \rightarrow 2^Q$ is defined as follows.

  1. For any $q \in Q$, $\delta^*(q, \Lambda) = \{q\}$.
  2. For any $q \in Q$, $y \in \Sigma^*$ and $a \in \Sigma$, $\delta^*(q, ya) = \bigcup_{r \in \delta^*(q, y)} \delta(r, a)$

- A string x is accepted by an NFA M if there is a sequence of moves M can make, starting in its initial state and processing the symbols of x, that will lead to an accepting state.

- **Definition 4.3:** Let $M = (Q, \Sigma, q_0, A, \delta)$ be an NFA. The string $x \in \Sigma^*$ is accepted by M if $\delta^*(q_0, x) \cap A \neq \varnothing$. The language recognized, or accepted, by M is the set $L(M)$ of all strings accepted by M. For any language $L \subseteq \Sigma^*$, L is recognized by M if $L = L(M)$.

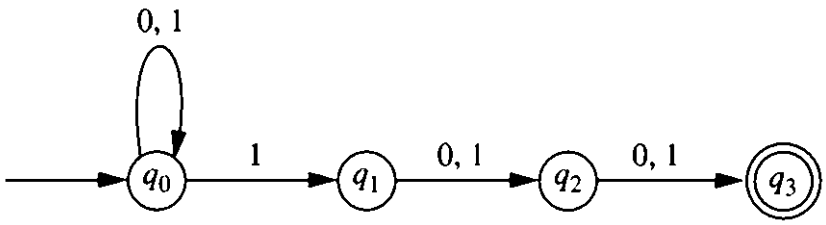
Extending the Transition Function (Continued)

- **Example 4.2:** Using the Recursive Definition of δ^* in an NFA

  Let $M = (Q, \Sigma, q_0, A, \delta)$, where $Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{0, 1\}$, A = $\{q_3\}$, and δ is given by the following table.

q	$\delta(q, 0)$	$\delta(q, 1)$
q_0	$\{q_0\}$	$\{q_0, q_1\}$
q_1	$\{q_2\}$	$\{q_2\}$
q_2	$\{q_3\}$	$\{q_3\}$
q_3	$\varnothing$	$\varnothing$

  The transition diagram for M:



  The definition of δ^* can be used to calculate $\delta^*(q_0, x)$ for various strings x of increasing length:

  $\delta^*(q_0, 0) = \{q_0\}$
  $\delta^*(q_0, 1) = \{q_0, q_1\}$
  $\delta^*(q_0, 11) = \{q_0, q_1, q_2\}$
  $\delta^*(q_0, 01) = \{q_0, q_1\}$
  $\delta^*(q_0, 111) = \{q_0, q_1, q_2, q_3\}$
  $\delta^*(q_0, 011) = \{q_0, q_1, q_2\}$

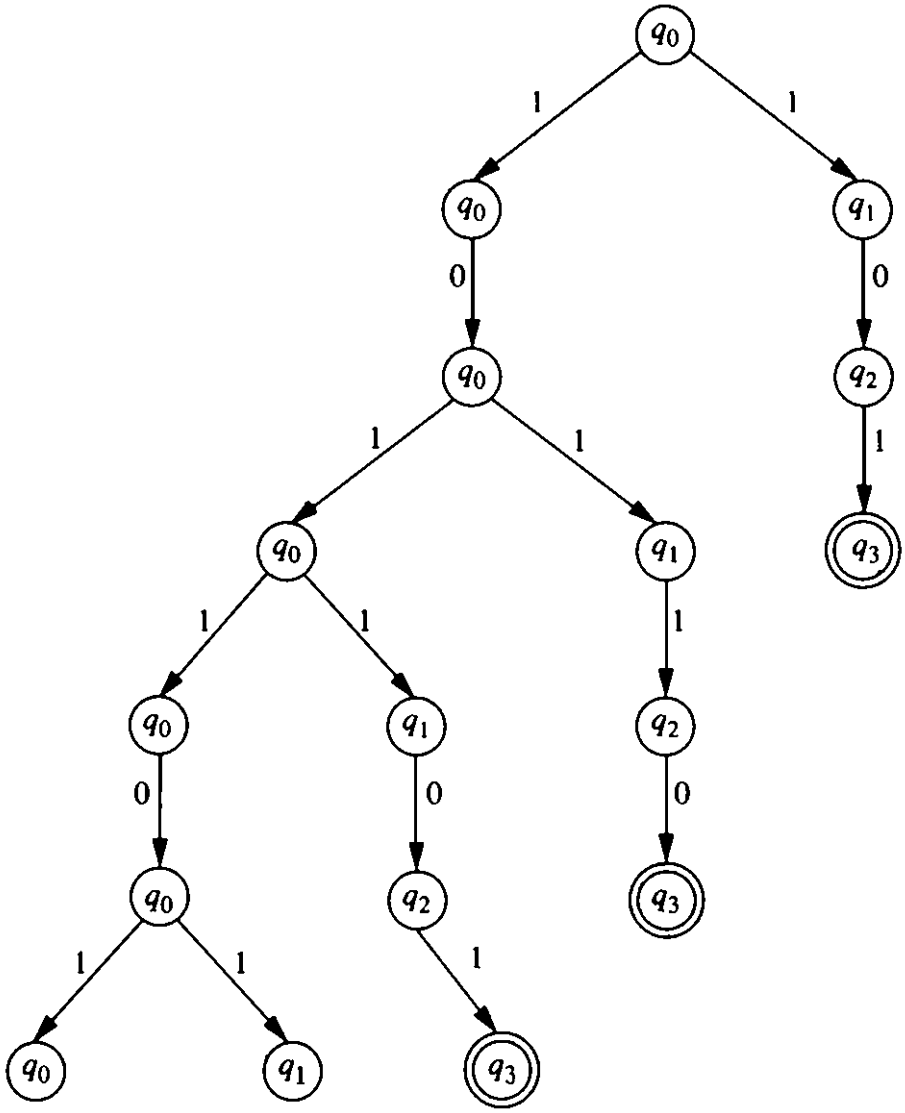
  The language recognized by M is $\{0, 1\}^*\{1\}\{0, 1\}^2$, which was called L_3 in Example 3.17. The technique used to build M can be used to construct for any $n \geq 1$ an NFA with $n + 1$ states that recognizes L_n.

  Since any ordinary FA accepting L_n needs at least 2^n states (Example 3.17), it is clear that an NFA recognizing a language may have considerably fewer states than any FA recognizing the language.

Extending the Transition Function (Continued)

Another way to visualize the behavior of the NFA as it processes a string is to draw a *computation tree* that traces the choices the machine has at each step.

A computation tree for the input string x = 101101:



It is easy to read off from the tree the sets $\delta^*(q_0, y)$ for prefixes y of x.

Deciding whether x is accepted is simply a matter of checking whether any accepting states appear in the tree at level $|x|$.

The Subset Construction for Converting an NFA to an FA

- Although an NFA may be easier to construct an an FA, nondeterministic finite automata as a group are no more powerful than FAs.

- Any NFA can be transformed into an FA by letting the states of the FA be subsets of the NFA's set of states. This technique is known as the *subset construction*.

- **Theorem 4.1.** For any NFA $M = (Q, \Sigma, q_0, A, \delta)$ accepting a language $L \subseteq \Sigma^*$, there is an FA $M_1 = (Q_1, \Sigma, q_1, A_1, \delta_1)$ that also accepts L.

  Proof: M_1 is defined as follows:

  $Q_1 = 2^Q \quad q_1 = \{q_0\} \quad$ for $q \in Q_1$ and $a \in \Sigma$, $\delta_1(q, a) = \bigcup_{r \in q} \delta(r, a)$
  $A_1 = \{q \in Q_1 \mid q \cap A \neq \varnothing\}$

  The fact that M_1 accepts the same language as M follows from the fact that for any $x \in \Sigma^*$, $\delta_1^*(q_1, x) = \delta^*(q_0, x)$, which can be proved using structural induction on x. If $x = \Lambda$,

  $$\begin{aligned} \delta_1^*(q_1, x) &= \delta_1^*(q_1, \Lambda) \\ &= q_1 && \text{(by definition of } \delta_1^*) \\ &= \{q_0\} && \text{(by definition of } q_1) \\ &= \delta^*(q_0, \Lambda) && \text{(by definition of } \delta^*) \\ &= \delta^*(q_0, x) \end{aligned}$$

  The induction hypothesis is that x is a string satisfying $\delta_1^*(q_1, x) = \delta(q_0, x)$, and the goal is to prove that for any $a \in \Sigma$, $\delta_1^*(q_1, xa) = \delta^*(q_0, xa)$:

  $$\begin{aligned} \delta_1^*(q_1, xa) &= \delta_1(\delta_1^*(q_1, x), a) && \text{(by definition of } \delta_1^*) \\ &= \delta_1(\delta^*(q_0, x), a) && \text{(by the induction hypothesis)} \\ &= \bigcup_{r \in \delta^*(q_0, x)} \delta(r, a) && \text{(by definition of } \delta_1) \\ &= \delta^*(q_0, xa) && \text{(by definition of } \delta^*) \end{aligned}$$

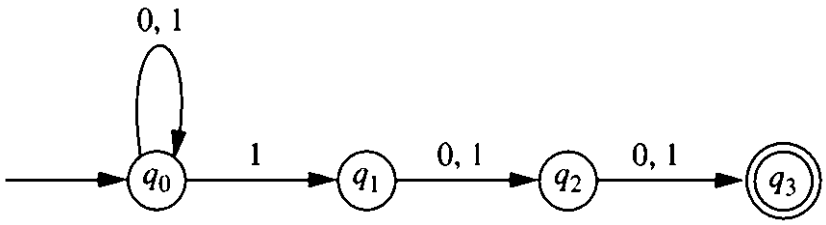
  A string x is accepted by M_1 if $\delta_1^*(q_1, x) \in A_1$; this is true if and only if $\delta^*(q_0, x) \in A_1$; using the definition of A_1, this is true if and only if $\delta^*(q_0, x) \cap A \neq \varnothing$. In other words, x is accepted by M_1 if and only if x is accepted by M.

Examples of the Subset Construction

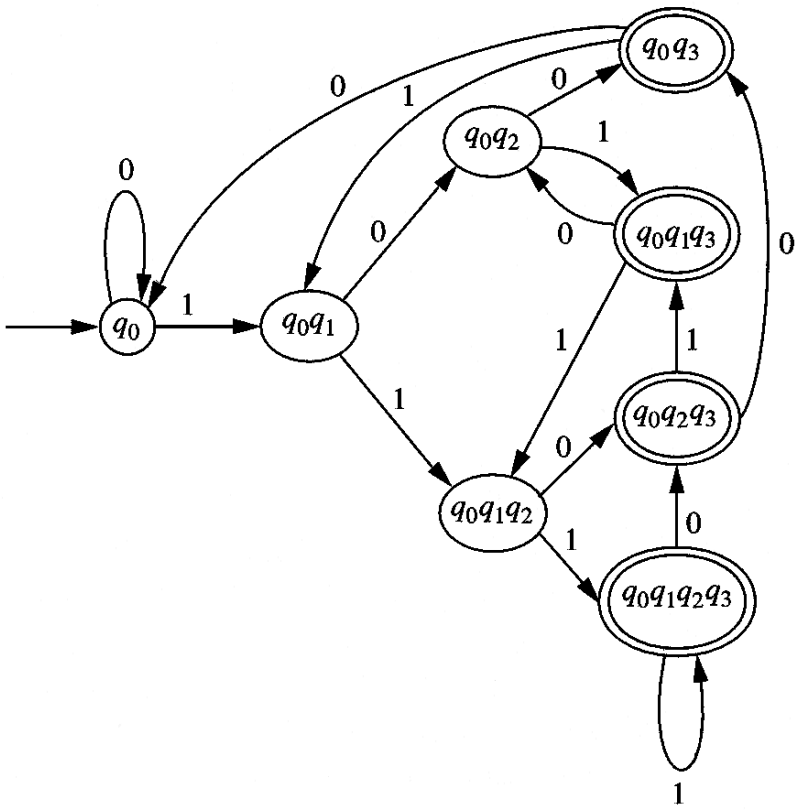
- The proof of Theorem 4.1 provides an algorithm (the subset construction) for removing the nondeterminism from an NFA.

- **Example 4.3:** Applying the Subset Construction to Convert an NFA to an FA

  Consider the NFA of Example 4.2:



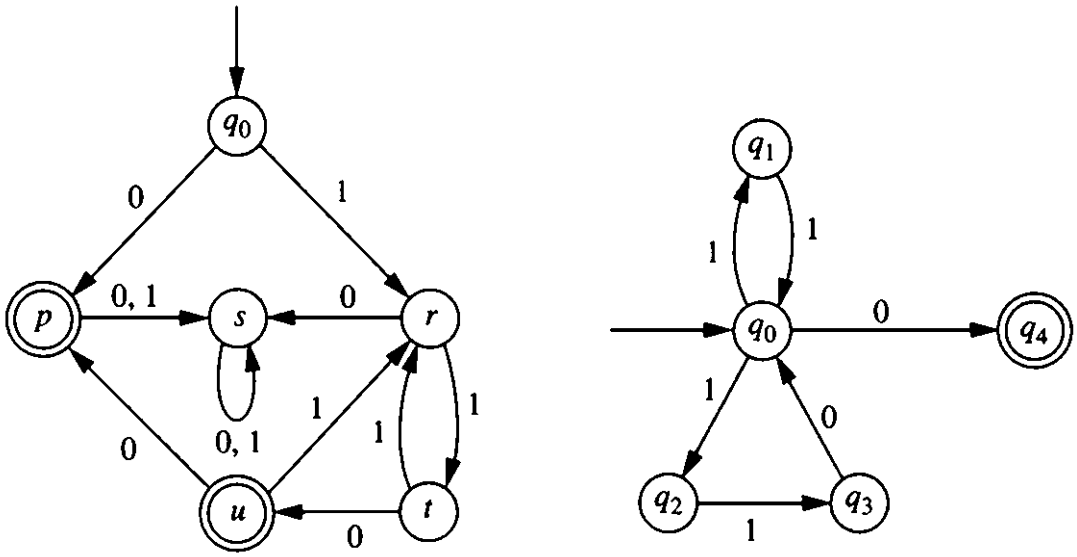
  The subset construction could produce an FA with as many as 16 states. However, it is often possible to get by with fewer by using only states that are reachable from the initial state. As it turns out, only eight states are reachable, which happens to be the minimum required for an FA that recognizes this language:



Examples of the Subset Construction (Continued)

- **Example 4.4:** Another Example Illustrating the Subset Construction

  The FA on the left (used in previous examples) is the result when the NFA on the right is converted to an FA using the subset construction:



  Converting the NFA to an FA produces the following transition table:

q	$\delta_1(q, 0)$	$\delta_1(q, 1)$
$\{q_0\}$	$\{q_4\}$	$\{q_1, q_2\}$
$\{q_4\}$	$\varnothing$	$\varnothing$
$\{q_1, q_2\}$	$\varnothing$	$\{q_0, q_3\}$
$\varnothing$	$\varnothing$	$\varnothing$
$\{q_0, q_3\}$	$\{q_0, q_4\}$	$\{q_1, q_2\}$
$\{q_0, q_4\}$	$\{q_4\}$	$\{q_1, q_2\}$

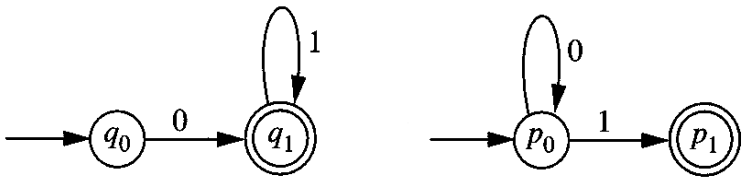
  The transition diagram on the left is the result when the states are renamed in the following way: $q_0 = \{q_0\}$, $p = \{q_4\}$, $r = \{q_1, q_2\}$, $s = \varnothing$, $t = \{q_0, q_3\}$, and $u = \{q_0, q_4\}$.

4.2 Nondeterministic Finite Automata with Λ-Transitions

- One further modification of finite automata will be helpful in proving that regular languages are the same as those accepted by FAs.

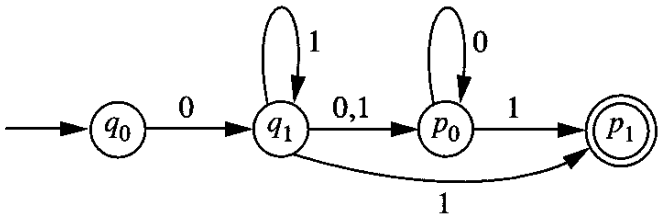
- **Example 4.5:** How a Device More General Than an NFA Can Be Useful

  The following NFAs M_1 and M_2 accept the two languages $L_1 = \{0\}\{1\}^*$and $L_2 = \{0\}^*\{1\}$, respectively:

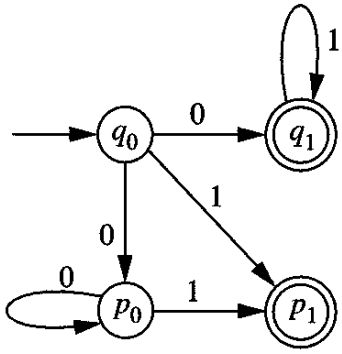


  Consider the problem of combining these NFAs in order to create new NFAs that recognize the concatenation and union of L_1 and L_2.

  The obvious way to build an NFA that accepts L_1L_2 is to connect the accepting state of M_1 with the initial state of M_2. In fact, adding transitions on 0 and 1 from q_1 to p_0 will work, provided that an extra transition is added on 1 from q_1 to p_1:



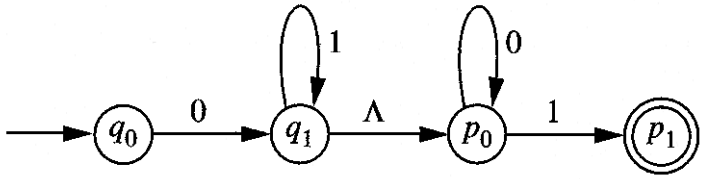
  The algorithm in Chapter 3 for constructing an FA that accepts the union of two languages does not always work for NFAs. The following NFA, constructed in an ad hoc fashion from M_1 and M_2, will accept $L_1 \cup L_2$:



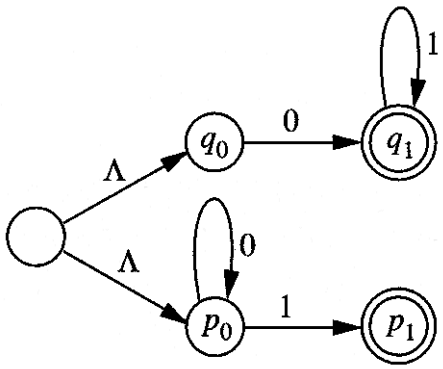
Nondeterministic Finite Automata with Λ-Transitions (Continued)

As these examples show, combining two NFAs is not always straightforward and may obscure the structure of the original NFAs.

One way to solve this problem is to allow the resulting NFA to make transitions without reading input (or, equivalently, make transitions while reading Λ). An NFA that accepts L_1L_2 would now have the following appearance:



A machine that accepts $L_1 \cup L_2$ would begin in a new initial state and then use a Λ-transition to enter either M_1 or M_2:

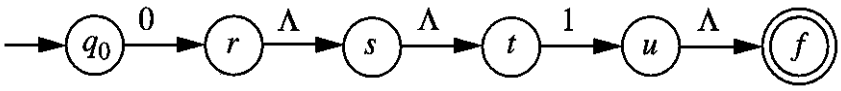


- **Definition 4.4:** A nondeterministic finite automaton with Λ-transitions (abbreviated NFA-Λ) is a 5-tuple $(Q, \Sigma, q_0, A, \delta)$, where Q and Σ are finite sets, $q_0 \in Q$, $A \subseteq Q$, and

  $\delta : Q \times (\Sigma \cup \{\Lambda\}) \rightarrow 2^Q$

Extending the Transition Function

- Defining the extended transition function δ^* for an NFA-Λ is tricky because of the possibility of Λ-transitions interspersed among ordinary transitions. For example, the following NFA-Λ accepts the string 01, but requires five transitions to do so:



- The first definition of δ^* will be nonrecursive.

- **Definition 4.5a:** For an NFA-Λ $M = (Q, \Sigma, q_0, A, \delta)$, states $p, q \in Q$, and a string $x = a_1a_2\ldots a_n \in \Sigma^*$, M is said to move from p to q by a sequence of transitions corresponding to x if there exist an integer $m \geq n$, a sequence $b_1, b_2, \ldots, b_m \in \Sigma \cup \{\Lambda\}$ satisfying $b_1b_2\ldots b_m = x$, and a sequence of states $p = p_0, p_1, \ldots, p_m = q$ so that for each i with $1 \leq i \leq m$, $p_i \in \delta(p_{i-1}, b_i)$.

  For $x \in \Sigma^*$ and $p \in Q$, $\delta^*(p, x)$ is the set of all states $q \in Q$ such that there is a sequence of transitions corresponding to x by which M moves from p to q.

Extending the Transition Function (Continued)

- Coming up with a recursive definition of δ^* is even harder. Even the basis part is difficult, because $\delta^*(q, \Lambda)$ contains not just q, but all states reachable from q by Λ-transitions. Similarly, a transition made on an input symbol a may be followed by Λ-transitions.

- The recursive definition of δ^* can be simplified by first defining another concept: the *Λ-closure* of a set S of states. The Λ-closure of S is the set of all states that can be reached from elements of S by using Λ-transitions.

- **Definition 4.6:** Let $M = (Q, \Sigma, q_0, A, \delta)$ be an NFA-Λ, and let S be any subset of Q. The Λ-closure of S is the set $\Lambda(S)$ defined as follows.

  1. Every element of S is an element of $\Lambda(S)$;
  2. For any $q \in \Lambda(S)$, every element of $\delta(q, \Lambda)$ is in $\Lambda(S)$;
  3. No other elements of Q are in $\Lambda(S)$.

- Because $\Lambda(S)$ is finite, its recursive definition can be translated into an algorithm.

- **Algorithm to Calculate $\Lambda(S)$:** Start with $T = S$. Make a sequence of passes, in each pass considering every $q \in T$ and adding to T all elements of $\delta(q, \Lambda)$ that are not already elements of T. Stop after any pass in which T does not change. The set $\Lambda(S)$ is the final value of T.

- With the help of the Λ-closure concept, the δ^* function can now be defined recursively.

- **Definition 4.5b:** Let $M = (Q, \Sigma, q_0, A, \delta)$ be an NFA-Λ. The extended transition function $\delta^* : Q \times \Sigma^* \rightarrow 2^Q$ is defined as follows.

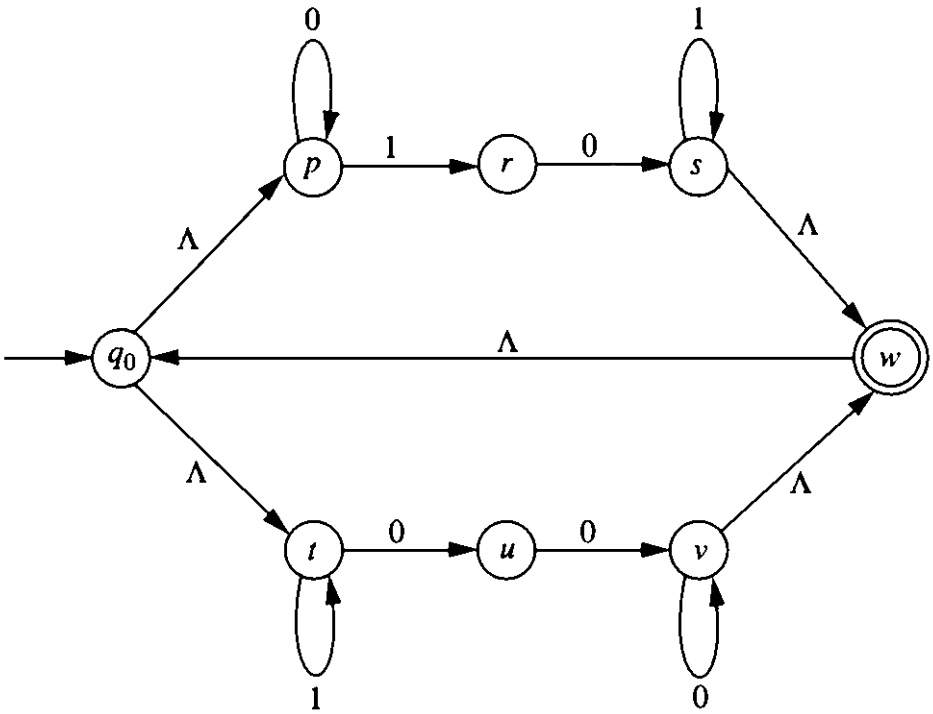
  1. For any $q \in Q$, $\delta^*(q, \Lambda) = \Lambda(\{q\})$.
  2. For any $q \in Q$, $y \in \Sigma^*$, and $a \in \Sigma$, $\delta^*(q, ya) = \Lambda\left(\bigcup_{r \in \delta^*(q, y)} \delta(r, a) \right)$

  A string x is accepted by M if $\delta^*(q_0, x) \cap A \neq \varnothing$. The language recognized by M is the set $L(M)$ of all strings accepted by M.

Extending the Transition Function (Continued)

- **Example 4.6:** Applying the Definitions of $\Lambda(S)$ and δ^*

  Consider the following NFA-Λ:



  Using Definition 4.5b, the value of $\delta^*(q, x)$ can be calculated for any state q and string x. Example of computing $\delta^*(q_0, 010)$:

$$\delta^*(q_0, \Lambda) = \Lambda(\{q_0\}) = \{q_0, p, t\}$$
$$\delta^*(q_0, 0) = \Lambda(\delta(q_0, 0) \cup \delta(p, 0) \cup \delta(t, 0)) = \Lambda(\varnothing \cup \{p\} \cup \{u\})$$
$$= \Lambda(\{p, u\}) = \{p, u\}$$
$$\delta^*(q_0, 01) = \Lambda(\delta(p, 1) \cup \delta(u, 1)) = \Lambda(\{r\}) = \{r\}$$
$$\delta^*(q_0, 010) = \Lambda(\delta(r, 0)) = \Lambda(\{s\}) = \{s, w, q_0, p, t\}$$

  The last step involves calculating the value of $\Lambda(S)$, where $S = \{s\}$:

  Initially: $T = S = \{s\}$
  After one iteration: $T = \{s, w\}$
  After two iterations: $T = \{s, w, q_0\}$
  After three iterations: $T = \{s, w, q_0, p, t\}$ (unchanged in the fourth iteration)

Converting an NFA-Λ to an NFA

- Theorem 4.1 stated that NFAs are no more powerful than FAs with regard to the languages they can accept. To establish the same result for NFA-Λs, it is sufficient to show that any NFA-Λ can be replaced by an equivalent NFA.

- **Theorem 4.2.** If $L \subseteq \Sigma^*$ is a language that is accepted by the NFA-Λ $M = (Q, \Sigma, q_0, A, \delta)$, then there is an NFA $M_1 = (Q_1, \Sigma, q_1, A_1, \delta_1)$ that also accepts L.

  Proof: M_1 can use the same state set Q and the same initial state as M. The transition function δ_1 will be defined so that if M can move from p to q using certain symbols together with Λ-transitions, then M_1 can move from p to q using those symbols without the Λ-transitions.

  Let M_1 be the NFA $(Q, \Sigma, q_0, A_1, \delta_1)$, where for any $q \in Q$ and $a \in \Sigma$,

  $\delta_1(q, a) = \delta^*(q, a)$

  and

  $$A_1 = \begin{cases} A \cup \{q_0\} & \text{if } \Lambda(\{q_0\}) \cap A \neq \varnothing \text{ in } M \\ A & \text{otherwise} \end{cases}$$

  The next step is to show that $\delta_1^*(q, x) = \delta^*(q, x)$ if x is any nonnull string. (The equality may not hold when x is null, because $\delta_1^*(q, \Lambda) = \{q\}$, whereas $\delta^*(q, \Lambda)$ may contain additional states.)

  The proof is by structural induction on x.

  The basis step is $x = a \in \Sigma$. In this case, $\delta^*(q, a) = \delta_1(q, a)$ by definition of δ_1, and $\delta_1(q, a) = \delta_1^*(q, a)$ because M_1 is an NFA.

Converting an NFA-Λ to an NFA (Continued)

In the induction step, assume that $|y| \geq 1$ and that $\delta_1^*(q, y) = \delta^*(q, y)$ for any $q \in Q$. The goal is to show that $\delta_1^*(q, ya) = \delta^*(q, ya)$ for any $a \in \Sigma$.

$$\delta_1^*(q, ya) = \bigcup_{r \in \delta_1^*(q, y)} \delta_1(r, a) \quad \text{(by Definition 4.2b)}$$

$$= \bigcup_{r \in \delta^*(q, y)} \delta_1(r, a) \quad \text{(by the induction hypothesis)}$$

$$= \bigcup_{r \in \delta^*(q, y)} \delta^*(r, a) \quad \text{(by definition of } \delta_1\text{)}$$

$$= \bigcup_{r \in \delta^*(q, y)} \Lambda\left(\bigcup_{p \in \Lambda(\{r\})} \delta^*(p, a) \right) \quad \text{(by Definition 4.5b)}$$

The union of Λ-closures is the Λ-closure of the union, so the last set can be written as

$$\Lambda\left(\bigcup_{r \in \delta^*(q, y)} \left(\bigcup_{p \in \Lambda(\{r\})} \delta(p, a) \right) \right)$$

It follows from the definition of Λ-closure (and the fact that $\delta^*(q, y)$ is a Λ-closure) that for every $r \in \delta^*(q, y)$, $\Lambda(\{r\}) \subseteq \delta^*(q, y)$; therefore, the two separate unions are unnecessary, and the formula reduces to

$$\Lambda\left(\bigcup_{r \in \delta^*(q, y)} \delta(r, a) \right)$$

which is the definition of $\delta^*(q, ya)$. The induction is complete.

Converting an NFA-Λ to an NFA (Continued)

It is not hard to show that M_1 recognizes L, the language recognized by M.

First, consider the case when $\Lambda(\{q_0\}) \cap A = \varnothing$ in M. Then the null string is accepted by neither M nor M_1. For any other string x, $\delta_1^*(q_0, x) = \delta^*(q_0, x)$, and the accepting states of M and M_1 are the same. Therefore, x is accepted by M_1 if and only if it is accepted by M.

In the other case, when $\Lambda(\{q_0\}) \cap A \neq \varnothing$, the construction defines A_1 to be $A \cup \{q_0\}$. This time Λ is accepted by both M and M_1. For any other x, $\delta_1^*(q_0, x) = \delta^*(q_0, x)$. If this set contains a state in A, then both M and M_1 accept x. If not, the only way x could be accepted by one of the two machines but not the other would be for $\delta^*(q_0, x)$ to contain q_0. (It would then intersect A_1 but not A.) However, this cannot happen either. By definition, $\delta^*(q_0, x)$ is the Λ-closure of a set. If $\delta^*(q_0, x)$ contained q_0, it would have to contain every element of $\Lambda(\{q_0\})$, and therefore, since $\Lambda(\{q_0\}) \cap A \neq \varnothing$, it would have to contain an element of A.

Converting an NFA-Λ to an NFA (Continued)

- It is technically not quite correct to say that every FA is an NFA, since the values of the transition function are states in one case and sets of states in the other.

- Similarly, an NFA is not an NFA-Λ, because the domain of the transition function in an NFA is $Q \times \Sigma$, and in an NFA-Λ it is $Q \times (\Sigma \cup \{\Lambda\})$.

- If these technicalities are ignored, it is easy to show the following result.

- **Theorem 4.3.** For any alphabet Σ, and any language $L \subseteq \Sigma^*$, these three statements are equivalent:

  1. L can be recognized by an FA.
  2. L can be recognized by an NFA.
  3. L can be recognized by an NFA-Λ.

  Proof: According to Theorems 4.1 and 4.2, statement 3 implies statement 2, and statement 2 implies statement 1. It is therefore sufficient to show that statement 1 implies statement 3.

  Suppose that L is accepted by the FA $M = (Q, \Sigma, q_0, A, \delta)$. An NFA-$\Lambda$ $M_1 = (Q, \Sigma, q_0, A, \delta_1)$ accepting L can be constructed as follows. $\delta_1 : Q \times (\Sigma \cup \{\Lambda\}) \rightarrow 2^Q$ is defined by the formulas

  $\delta_1(q, \Lambda) = \varnothing \qquad \delta_1(q, a) = \{\delta(q, a)\}$

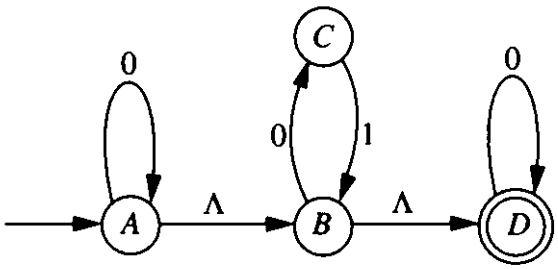
  (for any $q \in Q$ and any $a \in \Sigma$). The fact that M_1 accepts L follows immediately from the observation that δ and δ_1 are identical, except for the technical difference that the value is a state in one case and a set containing only that state in the other.

Examples of Converting an NFA-Λ to an NFA

- The proof of Theorem 4.2 provides an algorithm for eliminating Λ-transitions from an NFA-Λ.

- **Example 4.7:** Converting an NFA-Λ to an NFA

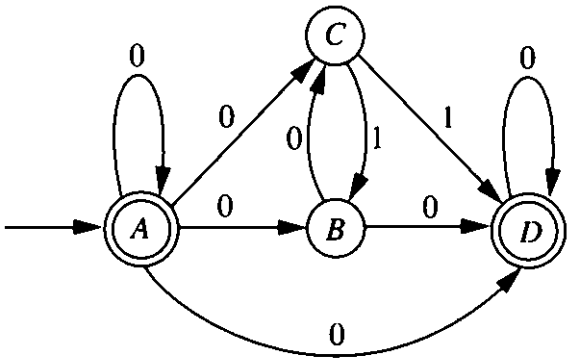
  The following NFA-Λ accepts the language {0}*{01}*{0}*:



  The transition function in tabular form, along with the values δ*(*q*, 0) and δ*(*q*, 1) that give the values of the transition function in the resulting NFA:

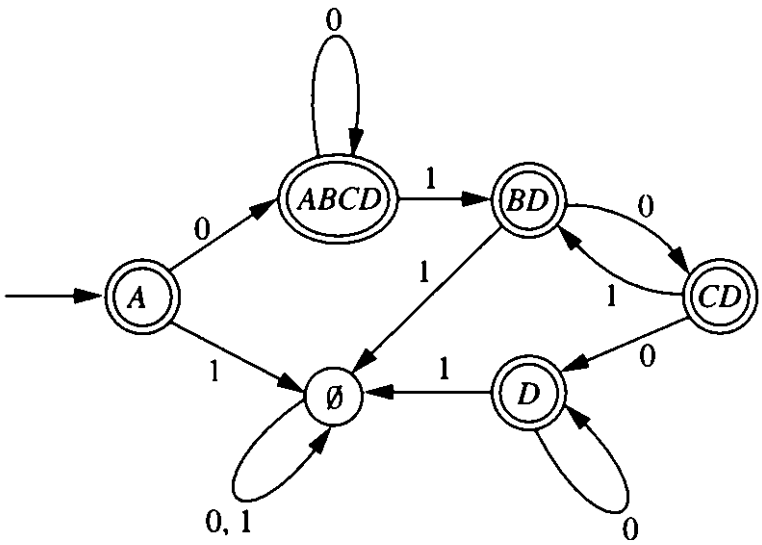
q	**δ(*q*, Λ)**	**δ(*q*, 0)**	**δ(*q*, 1)**	**δ*(*q*, 0)**	**δ*(*q*, 1)**
A	{*B*}	{*A*}	∅	{*A*, *B*, *C*, *D*}	∅
B	{*D*}	{*C*}	∅	{*C*, *D*}	∅
C	∅	∅	{*B*}	∅	{*B*, *D*}
D	∅	{*D*}	∅	{*D*}	∅

  The resulting NFA:



Examples of Converting an NFA-Λ to an NFA (Continued)

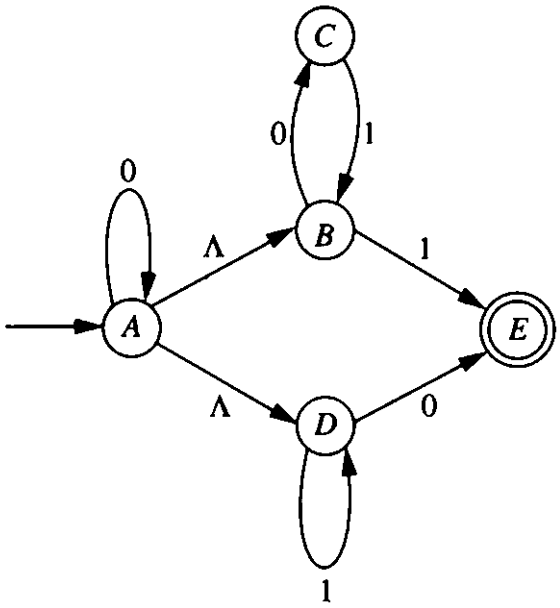
Having the values $\delta^*(q, 0)$ and $\delta^*(q, 1)$ in tabular form is useful in arriving at the corresponding FA:



Examples of Converting an NFA-Λ to an NFA (Continued)

- **Example 4.8:** Another Example of Converting an NFA-Λ to an NFA

  The following NFA-Λ recognizes the language $\{0\}^*(\{01\}^*\{1\} \cup \{1\}^*\{0\})$:

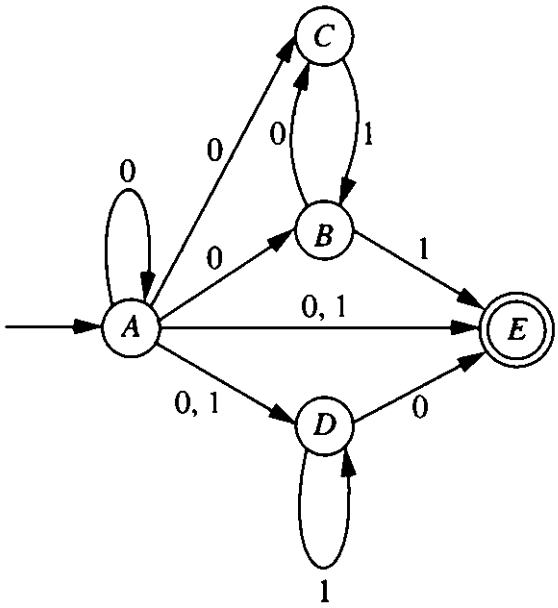


  The transition function in tabular form, along with the transition function for the resulting NFA:

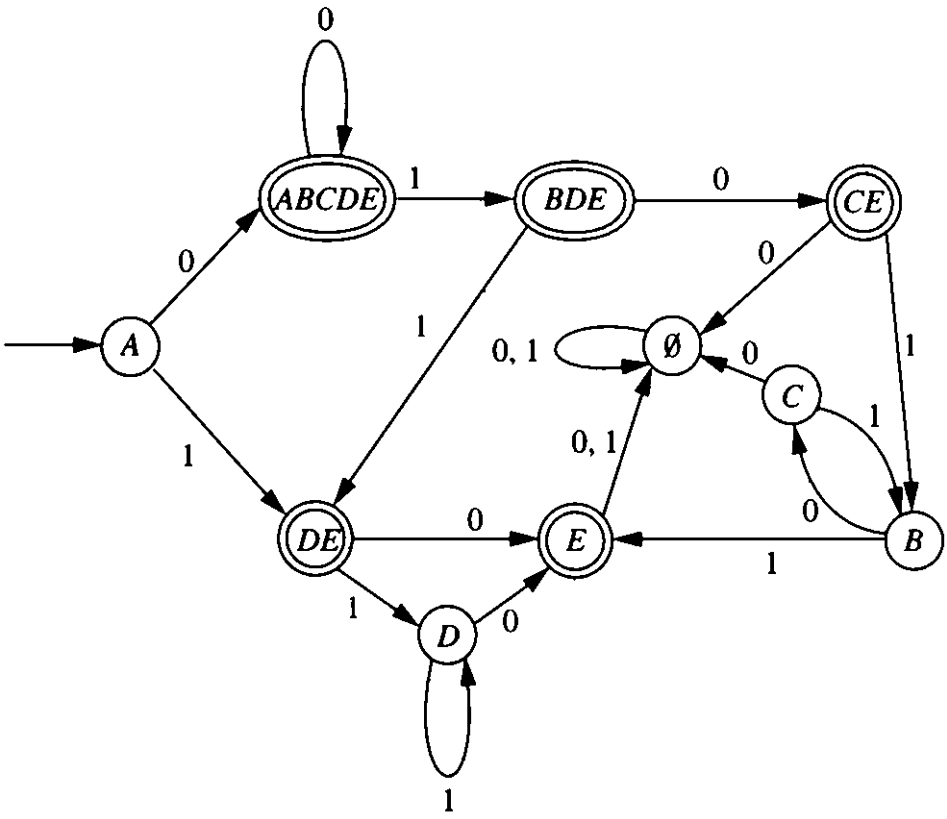
q	$\delta(q, \Lambda)$	$\delta(q, 0)$	$\delta(q, 1)$	$\delta^*(q, 0)$	$\delta^*(q, 1)$
A	$\{B, D\}$	$\{A\}$	$\varnothing$	$\{A, B, C, D, E\}$	$\{D, E\}$
B	$\varnothing$	$\{C\}$	$\{E\}$	$\{C\}$	$\{E\}$
C	$\varnothing$	$\varnothing$	$\{B\}$	$\varnothing$	$\{B\}$
D	$\varnothing$	$\{E\}$	$\{D\}$	$\{E\}$	$\{D\}$
E	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$

Examples of Converting an NFA-Λ to an NFA (Continued)

The resulting NFA:



The result of converting this NFA to an FA:



4.3 Kleene's Theorem

- This section provides a proof of Theorem 3.1, which states that the regular languages are exactly the language accepted by FAs. For convenience, the "if" and "only if" parts are proved separately.

- **Theorem 4.4 (Kleene's Theorem, Part 1).** Any regular language can be accepted by a finite automaton.

  Proof: Because of Theorem 4.3, it is sufficient to show that every regular language can be accepted by an NFA-Λ.

  The set of regular languages over the alphabet Σ is defined in Definition 3.1 to be the smallest set of languages that contains the basic languages ∅, {Λ}, and $\{a\}$ ($a \in \Sigma$) and is closed under the operations of union, concatenation, and Kleene *.

  This definition allows a proof using structural induction that every regular language over Σ can be accepted by an NFA-Λ.

  The basis step of the proof is to show that the three basic languages can be accepted by NFA-Λs, which is easy to do:

  The induction hypothesis is that L_1 and L_2 are languages that can be accepted by NFA-Λs, and the induction step is to show that $L_1 \cup L_2$, L_1L_2, and L_1* can also be.

  Suppose that L_1 and L_2 are recognized by the NFA-Λs M_1 and M_2, respectively, where for both $i = 1$ and $i = 2$,

  $M_i = (Q_i, \Sigma, q_i, A_i, \delta_i)$

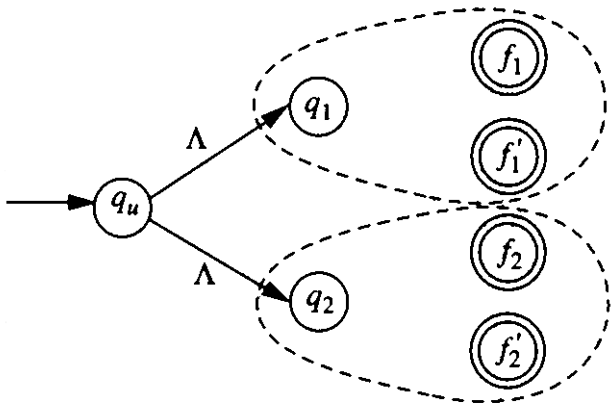
  By renaming states if necessary, it can be assumed that $Q_1 \cap Q_2 = \varnothing$. The plan is to construct NFA-Λs M_u, M_c, and M_k, recognizing the languages $L_1 \cup L_2$, L_1L_2, and L_1*, respectively.

Kleene's Theorem, Part 1 (Continued)

Construction of $M_u = (Q_u, \Sigma, q_u, A_u, \delta_u)$. Let q_u be a new state, not in either Q_1 or Q_2, and let

$Q_u = Q_1 \cup Q_2 \cup \{q_u\}$ $\quad$ $A_u = A_1 \cup A_2$

δ_u will be defined so that M_u can move from its initial state to either q_1 or q_2 by a Λ-transition:



A formal definition of δ_u:

$\delta_u(q_u, \Lambda) = \{q_1, q_2\}$ $\quad$ $\delta_u(q_u, a) = \varnothing$ for every $a \in \Sigma$

and for each $q \in Q_1 \cup Q_2$ and $a \in \Sigma \cup \{\Lambda\}$,

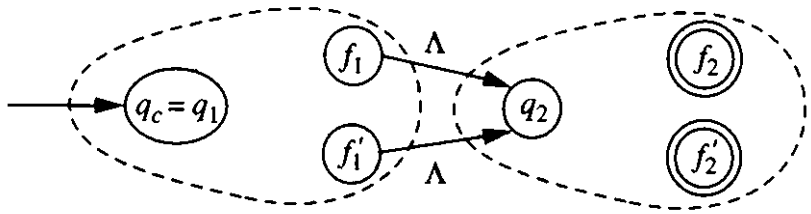
$$\delta_u(q, a) = \begin{cases} \delta_1(q, a) & \text{if } q \in Q_1 \\ \delta_2(q, a) & \text{if } q \in Q_2 \end{cases}$$

For either value of i, if $x \in L_i$, then M_u can process x by moving to q_i on a Λ-transition and then executing the moves that cause M_i to accept x.

On the other hand, if x is accepted by M_u, there is a sequence of transitions corresponding to x, starting at q_u and ending at an element of A_1 or A_2. The first of these transitions must be a Λ-transition from q_u to either q_1 or q_2, since there are no other transitions from q_u. Thereafter, since $Q_1 \cap Q_2 = \varnothing$, either all the transitions are between elements of Q_1 or all are between elements of Q_2. It follows that x must be accepted by either M_1 or M_2.

Kleene's Theorem, Part 1 (Continued)

Construction of $M_c = (Q_c, \Sigma, q_c, A_c, \delta_c)$. Let $Q_c = Q_1 \cup Q_2$, $q_c = q_1$, and $A_c = A_2$. The transitions will include all those of M_1 and M_2, as well as a Λ-transition from each state in A_1 to q_2:



For any q not in A_1, and $a \in \Sigma \cup \{\Lambda\}$, $\delta_c(q, a)$ is defined to be either $\delta_1(q, a)$ or $\delta_2(q, a)$, depending on whether q is in Q_1 or Q_2. For $q \in A_1$,

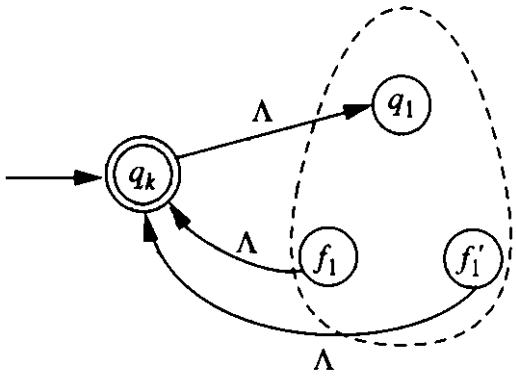
$\delta_c(q, a) = \delta_1(q, a)$ for every $a \in \Sigma$, and $\delta_c(q, \Lambda) = \delta_1(q, \Lambda) \cup \{q_2\}$

On an input string x_1x_2, where $x_i \in L_i$ for both values of i, M_c can process x_i, arriving at a state in A_1; jump from this state to q_2 by a Λ-transition; and then process x_2 the way M_2 would, so that x_1x_2 is accepted.

Conversely, if x is accepted by M_c, there is a sequence of transitions corresponding to x that begins at q_1, and ends at an element of A_2. One of them must therefore be from an element of Q_1 to an element of Q_2, and according to the definition of δ_c, this can only be a Λ-transition from an element of A_1 to q_2. Because $Q_1 \cap Q_2 = \varnothing$, all the previous transitions are between elements of Q_1, and all the subsequent ones are between elements of Q_2. It follows that $x = x_1\Lambda x_2 = x_1x_2$, where x_1 is accepted by M_1 and x_2 is accepted by M_2; in other words, $x \in L_1L_2$.

Kleene's Theorem, Part 1 (Continued)

Construction of $M_k = (Q_k, \Sigma, q_k, A_k, \delta_k)$. Let q_k be a new state not in Q_1 and let $Q_k = Q_1 \cup \{q_k\}$ and $A_k = \{q_k\}$. Once again all the transitions of M_1 will be allowed in M_k, but in addition there is a Λ-transition from q_k to q_1, and there is a Λ-transition from each element of A_1 to q_k:



More precisely,

$\delta_k(q_k, \Lambda) = \{q_1\}$ and $\delta_k(q_k, a) = \varnothing$ for $a \in \Sigma$.
For $q \in Q_1$ and $a \in \Sigma \cup \{\Lambda\}$, $\delta_k(q_k, a) = \delta_1(q, a)$ unless $q \in A_1$ and $a = \Lambda$.
For $q \in A_1$, $\delta_k(q, \Lambda) = \delta_1(q, \Lambda) \cup \{q_k\}$.

Suppose $x \in L_1^*$. If $x = \Lambda$, then clearly x is accepted by M_k. Otherwise, for some $m \geq 1$, $x = x_1x_2\ldots x_m$, where $x_i \in L_1$ for each i. M_k can move from q_k to q_1 by a Λ-transition; for each i, M_k moves from q_1 to an element f_i of A_1 by a sequence of transitions corresponding to x_i; and for each i, M_k then moves from f_i back to q_k by a Λ-transition. It follows that $(\Lambda x_1\Lambda)(\Lambda x_2\Lambda)\ldots(\Lambda x_m\Lambda) = x$ is accepted by M_k.

On the other hand, if x is accepted by M_k, there is a sequence of transitions corresponding to x that begins and ends at q_k. Since the only transition from q_k is a Λ-transition to q_1, and the only transitions to q_k are Λ-transitions from elements of A_1, x can be written as $x = (\Lambda x_1\Lambda)(\Lambda x_2\Lambda)\ldots(\Lambda x_m\Lambda)$ where, for each i, there is a sequence of transitions corresponding to x_i from q_1 to an element of A_1. Therefore, $x \in L_1^*$.

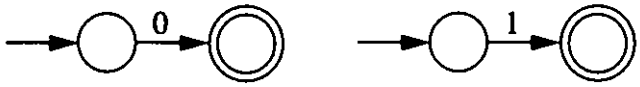
Applying Kleene's Theorem, Part 1

- The constructions in the proof of Theorem 4.4 provide an algorithm for constructing an NFA-Λ corresponding to a given regular expression.

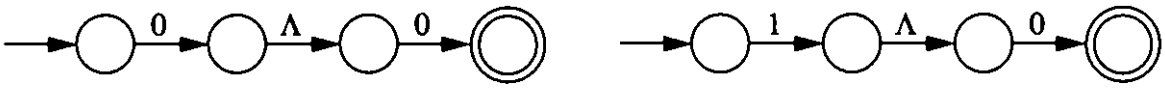
- **Example 4.9:** Applying the Algorithm in the Proof of Theorem 4.4

  Let *r* be the regular expression (00 + 1)*(10)*.

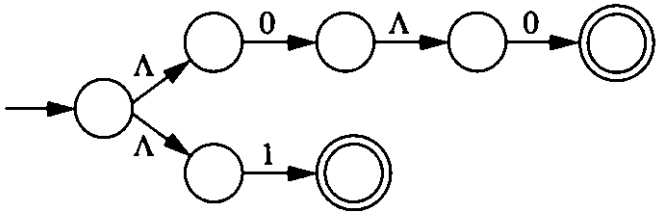
  The primitive (zero-operation) regular expressions appearing in *r*:



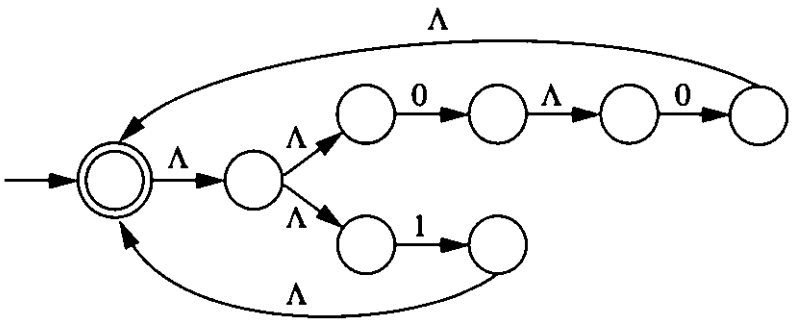
  The NFA-Λs corresponding to 00 and 10:



  The NFA-Λ corresponding to (00 + 1):

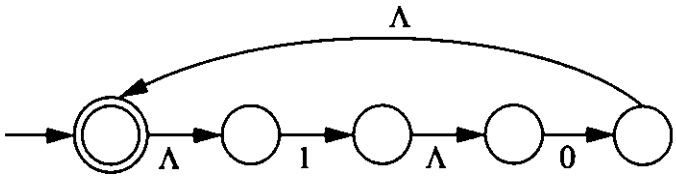


  The NFA-Λ corresponding to (00 + 1)*:

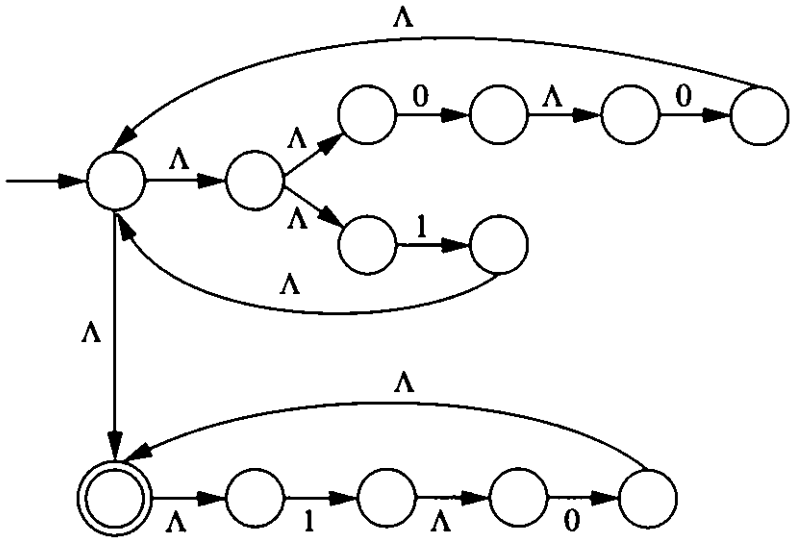


Applying Kleene's Theorem, Part 1 (Continued)

The NFA-Λ corresponding to (10)*:

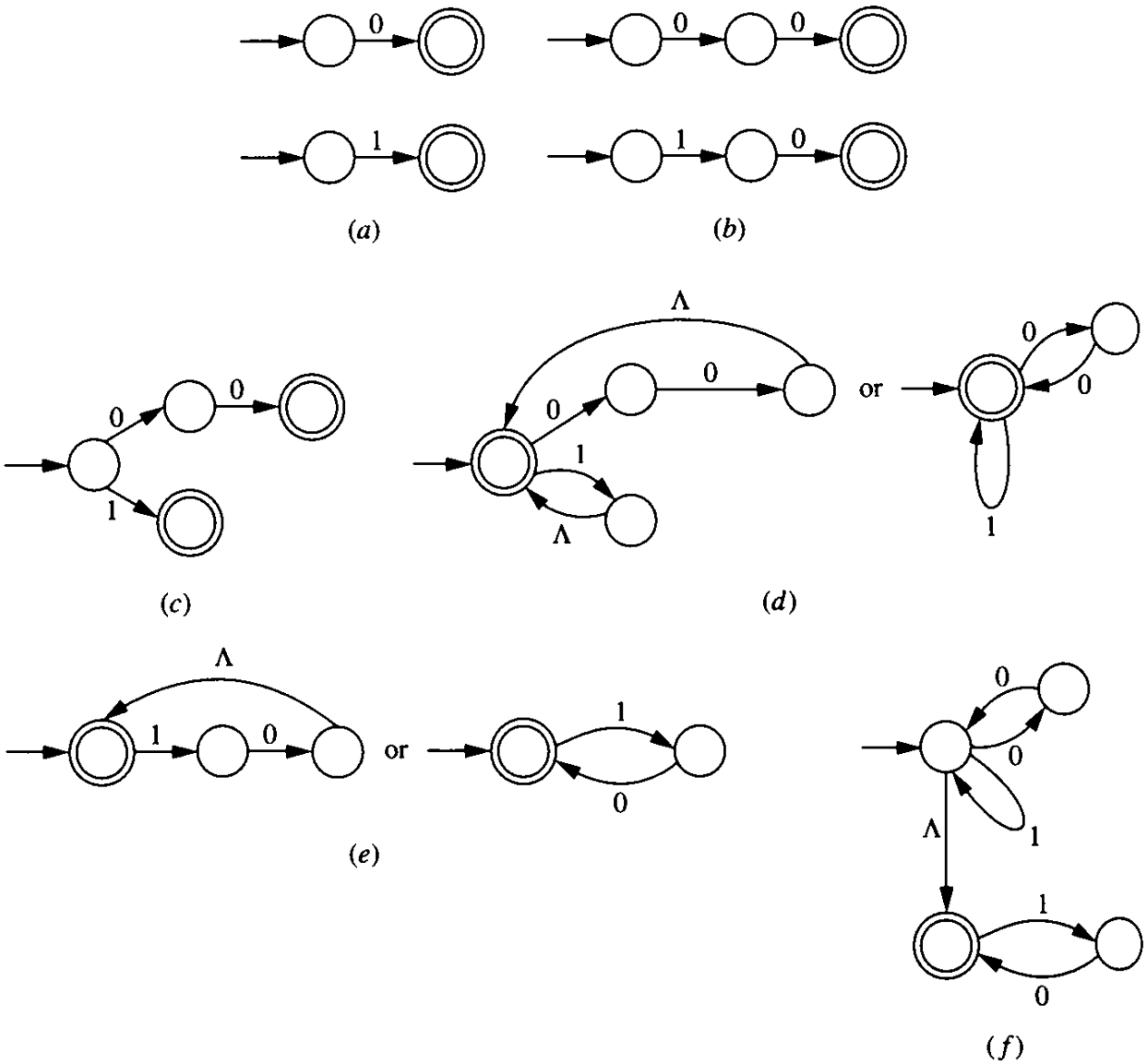


The resulting NFA-Λ formed by concatenation:



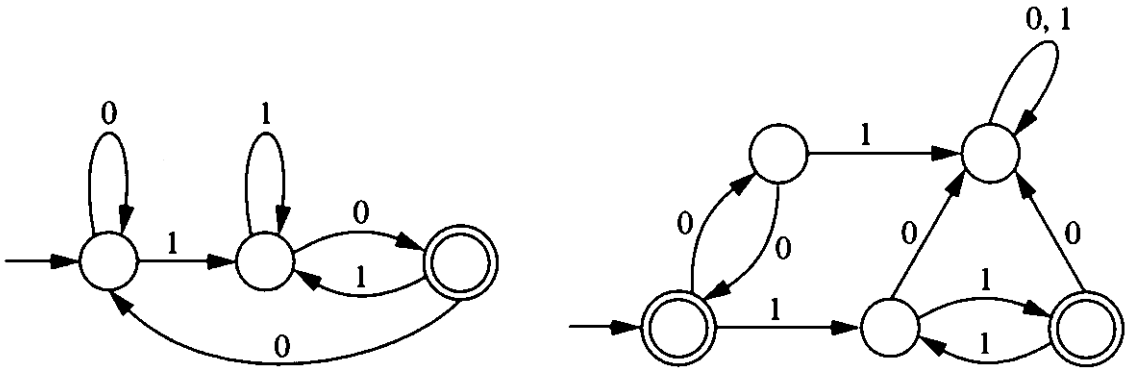
Applying Kleene's Theorem, Part 1 (Continued)

The general construction often produces unnecessary states and Λ-transitions. A smaller NFA-Λ can be constructed for (00 + 1)*(10)* by incorporating some obvious simplifications:

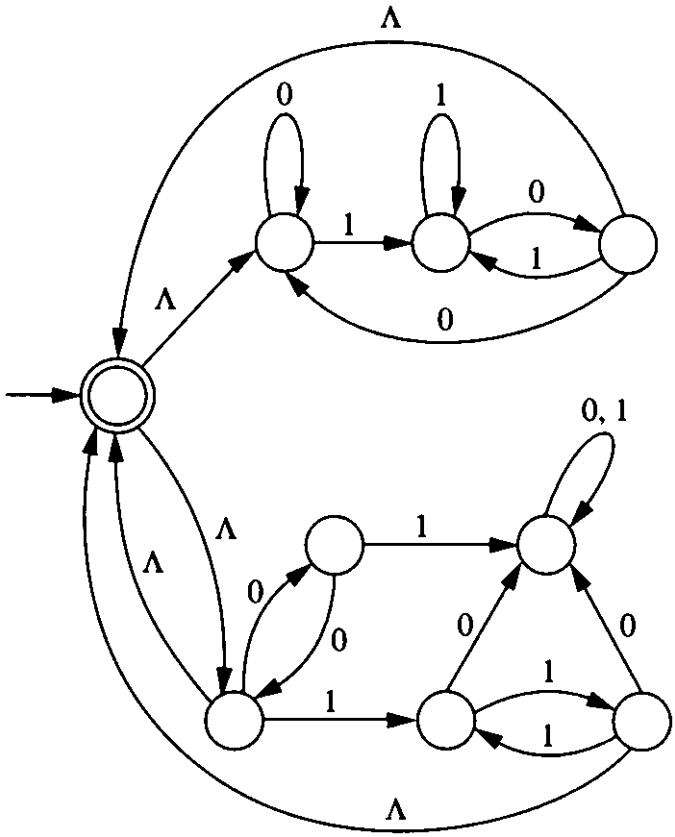


Applying Kleene's Theorem, Part 1 (Continued)

- The proof of Theorem 4.4 provides algorithms for constructing NFA-Λs to recognize the union, concatenation, and Kleene * of the languages accepted by other NFA-Λs. The following FAs accept the languages {0, 1}*{1, 0} and {00}*{11}*:



Applying the algorithms for union and Kleene * (with one simplification) yields the following NFA-Λ, which recognizes ({0, 1}*{10} ∪ {00}*{11}*)*:



Kleene's Theorem, Part 2

- **Theorem 4.5 (Kleene's Theorem, Part 2).** The language accepted by any finite automaton is regular.

  Proof: Let $L \subseteq \Sigma^*$ be accepted by the FA $M = (Q, \Sigma, q_0, A, \delta)$. Then

  $L = \{x \in \Sigma^* \mid \delta^*(q_0, x) \in A\}$

  which means that L is a finite union of sets of the form

  $\{x \in \Sigma^* \mid \delta^*(q_0, x) = q\}$

  If each one of these sets is regular, then L must be regular. In fact, we will prove that for any two states p and q, the following set is regular:

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

  To simplify the notation, relabel the states of M using the integers 1 through n, where n is the number of states. A string x in Σ^* represents a path from p to q *going through* s if there are nonnull strings y and z so that

  $x = yz \quad \delta^*(p, y) = s \quad \delta^*(s, z) = q$

  Note a path can go to a state, or from a state, without going through it. For $j \geq 0$, let $L(p, q, j)$ be the set of strings corresponding to paths from p to q that go through no state numbered higher than j.

  No string in $L(p, q)$ can go through a state numbered higher than n, because there are no states numbered higher than n. In other words,

  $L(p, q, n) = L(p, q)$

  The goal is to show that $L(p, q, n)$ is regular; this will follow if $L(p, q, j)$ is regular for every j with $0 \leq j \leq n$. In order to use ordinary induction, the proof will actually show that $L(p, q, j)$ is regular for every $j \geq 0$; this is not really a stronger statement, since for any $j \geq n$, $L(p, q, j) = L(p, q, n)$.

  Basis step. Going through no state numbered higher than 0 means going through no state at all, which means that the string can contain no more than one symbol. Therefore,

  $L(p, q, 0) \subseteq \Sigma \cup \{\Lambda\}$

  $L(p, q, 0)$ is regular because it is finite.

Kleene's Theorem, Part 2 (Continued)

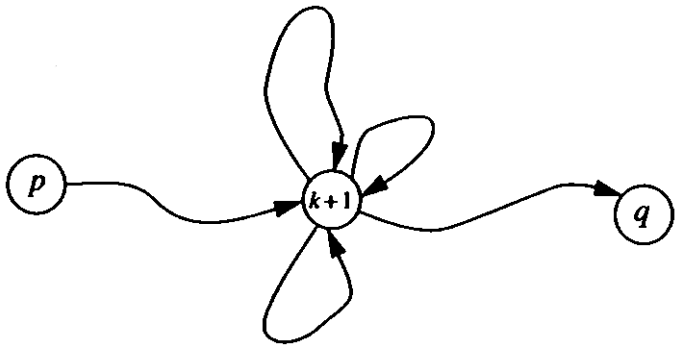
Induction hypothesis. $k \geq 0$ and for every p and q satisfying $0 \leq p, q \leq n$, the language $L(p, q, k)$ is regular.

Statement to show in induction step. For every p and q in the same range, $L(p, q, k + 1)$ is regular.

Proof of induction step. Since $L(p, q, k + 1) = L(p, q, k)$ if $k \geq n$, assume that $k < n$.

A string x is in $L(p, q, k + 1)$ if it represents a path from p to q that goes through no state numbered higher than $k + 1$. If this path bypasses $k + 1$, then it goes through no state higher than k, and $x \in L(p, q, k)$.

Otherwise, the path goes through $k + 1$. In this case, it goes from p to the first occurrence of $k + 1$, then loops from $k + 1$ back to itself zero or more times, then goes from the last occurrence of $k + 1$ to q:



Then $x = yzw$, where y corresponds to the path from p to the first occurrence of $k + 1$, z to all the loops from $k + 1$ back to itself, and w to the path from $k + 1$ to q. Therefore, $y \in L(p, k + 1, k)$, $w \in L(k + 1, q, k)$, and $z \in L(k + 1, k + 1, k)^*$.

Combining the two cases gives

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

Any string in the right-hand set belongs to $L(p, q, k + 1)$, since the corresponding path goes from p to q without going through any state higher than $k + 1$, so

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

Each language appearing in the right-hand set is regular because of the induction hypothesis, and $L(p, q, k + 1)$ is obtained from them by using the operations of union, concatenation, and Kleene *. Therefore, $L(p, q, k + 1)$ is regular.

Kleene's Theorem, Part 2 (Continued)

- Although the proof of Theorem 4.5 did not specify a formula for $L(p, q, 0)$, it is easy to derive.

- A summary of the formulas used in the proof of the theorem:

$$L(p, q, 0) = \begin{cases} \{a \in \Sigma \mid \delta(p, a) = q\} & \text{if } p \neq q \\ \{a \in \Sigma \mid \delta(p, a) = p\} \cup \{\Lambda\} & \text{if } p = q \end{cases}$$

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

$$L(p, q) = L(p, q, n)$$

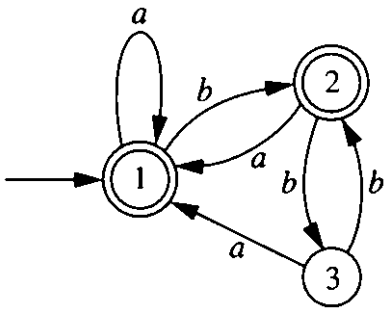
$$L = \bigcup_{q \in A} L(q_0, q)$$

These formulas can be used as the basis for an algorithm that finds a regular expression corresponding to a given FA.

Applying Kleene's Theorem, Part 2

- **Example 4.10:** Applying the Algorithm in the Proof of Theorem 4.5

  Let M be the following FA:



  To carry out the algorithm for finding a regular expression corresponding to M, it is useful to construct tables showing regular expressions $r(p, q, j)$ corresponding to the languages $L(p, q, j)$ for $0 \le j \le 2$:

p	$r(p, 1, 0)$	$r(p, 2, 0)$	$r(p, 3, 0)$
1	$a + \Lambda$	b	$\varnothing$
2	a	Λ	b
3	a	b	Λ

p	$r(p, 1, 1)$	$r(p, 2, 1)$	$r(p, 3, 1)$
1	a^*	a^*b	$\varnothing$
2	a^+	$\Lambda + a^+b$	b
3	a^+	a^*b	Λ

p	$r(p, 1, 2)$	$r(p, 2, 2)$	$r(p, 3, 2)$
1	$a^*(ba^+)^*$	$a^*(ba^+)^*b$	$a^*(ba^+)^*bb$
2	$a^+(ba^+)^*$	$(a^+b)^*$	$(a^+b)^*b$
3	$a^+ + a^*(ba^+)^+$	$a^*b(a^+b)^*$	$\Lambda + a^*b(a^+b)^*b$

  Although many of these table entries can be obtained by inspection, the formula can be used wherever necessary.

Applying Kleene's Theorem, Part 2 (Continued)

The expressions produced by the formulas of Theorem 4.5 can often be simplified. Some examples:

$$\begin{aligned} r(1, 3, 1) &= r(1, 3, 0) + r(1, 1, 0)r(1, 1, 0)^*r(1, 3, 0) \\ &= \varnothing \end{aligned}$$

$$\begin{aligned} r(3, 2, 1) &= r(3, 2, 0) + r(3, 1, 0)r(1, 1, 0)^*r(1, 2, 0) \\ &= b + a(a + \Lambda)^*b \\ &= \Lambda b + a^+b \\ &= a^*b \end{aligned}$$

$$\begin{aligned} r(1, 1, 2) &= r(1, 1, 1) + r(1, 2, 1)r(2, 2, 1)^*r(2, 1, 1) \\ &= a^* + a^*b(\Lambda + a^+b)^*a^+ \\ &= a^* + a^*(ba^+)^*ba^+ \\ &= a^* + a^*(ba^+)^+ \\ &= a^*(ba^+)^* \end{aligned}$$

$$\begin{aligned} r(3, 2, 2) &= r(3, 2, 1) + r(3, 2, 1)r(2, 2, 1)^*r(2, 2, 1) \\ &= a^*b + a^*b(a^+b)^*(\Lambda + a^+b) \\ &= a^*b + a^*b(a^+b)^* \\ &= a^*b(a^+b)^* \end{aligned}$$

M's accepting states are 1 and 2, so the regular expression r that denotes the language accepted by M is $r(1, 1, 3) + r(1, 2, 3)$:

$$\begin{aligned} r(1, 1, 3) &= r(1, 1, 2) + r(1, 3, 2)r(3, 3, 2)^*r(3, 1, 2) \\ &= a^*(ba^+)^* + a^*(ba^+)^*bb(\Lambda + a^*b(a^+b)^*b)^*(a^+ + a^*(ba^+)^+) \\ &= a^*(ba^+)^* + a^*(ba^+)^*bb(a^*(ba^+)^*bb)^*(a^+ + a^*(ba^+)^+) \\ &= a^*(ba^+)^* + (a^*(ba^+)^*bb)^+(a^+ + a^*(ba^+)^+) \end{aligned}$$

$$\begin{aligned} r(1, 2, 3) &= r(1, 2, 2) + r(1, 3, 2)r(3, 3, 2)^*r(3, 2, 2) \\ &= a^*(ba^+)^*b + a^*(ba^+)^*bb(a^*b(a^+b)^*b)^*a^*b(a^+b)^* \\ &= a^*(ba^+)^*b + a^*(ba^+)^*bb(a^*(ba^+)^*bb)^*a^*b(a^+b)^* \\ &= a^*(ba^+)^*b + (a^*(ba^+)^*bb)^+a^*(ba^+)^*b \\ &= (a^*(ba^+)^*bb)^*a^*(ba^+)^*b \end{aligned}$$

$$r = r(1, 1, 3) + r(1, 2, 3)$$

r can be further simplified if desired.