T065001: Introduction to Formal Languages

Lecture 4: Finite-state automata, regular languages, nondeterminism (3)

Chapter 1.3 in Sipser's textbook

2025-05-12

(Lecture slides by Yih-Kuen Tsay)

(From Chapter 1.1)

Definition (1.23)

Let A and B be languages. The three *regular operations* are defined as follows:

- **Union**: $A \cup B = \{x \mid x \in A \text{ or } x \in B\}$.
- **Concatenation**: $A \circ B = \{xy \mid x \in A \text{ and } y \in B\}$.
- **Star**: $A^* = \{x_1 x_2 \dots x_k \mid k \geq 0 \text{ and each } x_i \in A\}$.

EXAMPLE **1.24**

Let the alphabet Σ be the standard 26 letters $\{\mathtt{a}, \mathtt{b}, \dots, \mathtt{z}\}$. If $A = \{\mathtt{good}, \mathtt{bad}\}$ and $B = \{\mathtt{boy}, \mathtt{girl}\}$, then

$$A \cup B = \{\mathtt{good}, \mathtt{bad}, \mathtt{boy}, \mathtt{girl}\},$$

$$A \circ B = \{\mathtt{goodboy}, \mathtt{goodgirl}, \mathtt{badboy}, \mathtt{badgirl}\}, \text{ and}$$

$$A^* = \{\varepsilon, \text{good, bad, goodgood, goodbad, badgood, badbad,} \\ \text{goodgoodgood, goodgoodbad, goodbadgood, goodbadbad, } \dots\}.$$

■

(From Chapter 1.2)

Theorem (1.45)

The class of regular languages is closed under the union operation.

Theorem (1.47)

The class of regular languages is closed under the concatenation operation.

Theorem (1.49)

The class of regular languages is closed under the star operation.

Regular Expressions

- We can use the regular operations (union, concatenation, star) to build up expressions, called *regular expressions*, to describe languages.
- The *value* of a regular expression is a *language*.
- For example, the value of $(0 \cup 1)0^*$ is the language consisting of all strings starting with a 0 or 1 followed by any number of 0s.

Formal Definition of a Regular Expression

Definition (1.52)

We say that R is a *regular expression* if R is

1. a for some $a \in \Sigma$,
2. ε,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions, or
6. (R_1^*), where R_1 is a regular expression.

Formal Definition of a Regular Expression

Definition (1.52)

We say that R is a *regular expression* if R is

1. a for some $a \in \Sigma$,
2. ε,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions, or
6. (R_1^*), where R_1 is a regular expression.

The regular expressions in 1. and 2. represent the languages $\{a\}$ and $\{\varepsilon\}$.

Formal Definition of a Regular Expression

Definition (1.52)

We say that R is a *regular expression* if R is

1. a for some $a \in \Sigma$,
2. ε,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions, or
6. (R_1^*), where R_1 is a regular expression.

The regular expressions in 1. and 2. represent the languages $\{a\}$ and $\{\varepsilon\}$.

The regular expression in 3. represents the empty language $\{\}$.

Formal Definition of a Regular Expression

Definition (1.52)

We say that R is a *regular expression* if R is

1. a for some $a \in \Sigma$,
2. ε,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions, or
6. (R_1^*), where R_1 is a regular expression.

The regular expressions in 1. and 2. represent the languages $\{a\}$ and $\{\varepsilon\}$.

The regular expression in 3. represents the empty language $\{\}$.

The regular expression in 4. represents the language that is the union of the two languages represented by R_1 and R_2. Analogously for 5. and 6.

Regular expressions (cont.)

To simplify the notation, we often omit redundant symbols in a regular expression.

Previous example: $(0 \cup 1)0^*$ represents the language consisting of all strings that start with a 0 or a 1 and are followed by zero or more 0s.

Here, $(0 \cup 1)0^*$ actually means $((\{0\} \cup \{1\}) \circ (\{0\}^*))$.

- 0 and 1 are shorthand for the sets $\{0\}$ and $\{1\}$.
- Also note that the concatenation symbol $\circ$ is usually implicit.
- Parentheses may be omitted. If so then the priority order is: star operation first, then concatenation, then union.

Some Examples

Another example: What does the regular expression $(0 \cup 1)^*$ represent?

Some Examples

Another example: What does the regular expression $(0 \cup 1)^*$ represent?
It takes the language represented by $(0 \cup 1)$, which is $\{0, 1\}$, and applies the star operation.
$\Rightarrow (0 \cup 1)^*$ represents the language of all possible strings of 0s and 1s.

Some Examples

Another example: What does the regular expression $(0 \cup 1)^*$ represent? It takes the language represented by $(0 \cup 1)$, which is $\{0, 1\}$, and applies the star operation.
$\Rightarrow$ $(0 \cup 1)^*$ represents the language of all possible strings of 0s and 1s.

Additional notation: For any alphabet $\Sigma = \{a_1, a_2, \ldots, a_k\}$, we let Σ stand for the regular expression $a_1 \cup a_2 \cup \cdots \cup a_k$, which represents the language consisting of all strings of length 1 over this alphabet.
$\Rightarrow$ The regular expression Σ^* represents the language of all strings over Σ.

Some Examples

Another example: What does the regular expression $(0 \cup 1)^*$ represent? It takes the language represented by $(0 \cup 1)$, which is $\{0, 1\}$, and applies the star operation.
$\Rightarrow$ $(0 \cup 1)^*$ represents the language of all possible strings of 0s and 1s.

Additional notation: For any alphabet $\Sigma = \{a_1, a_2, \ldots, a_k\}$, we let Σ stand for the regular expression $a_1 \cup a_2 \cup \cdots \cup a_k$, which represents the language consisting of all strings of length 1 over this alphabet.
$\Rightarrow$ The regular expression Σ^* represents the language of all strings over Σ.

Examples: Let $\Sigma = \{0, 1\}$. Then $\Sigma^* 1$ represents the language of all strings of 0s and 1s that end with a 1.

Some Examples

Another example: What does the regular expression $(0 \cup 1)^*$ represent? It takes the language represented by $(0 \cup 1)$, which is $\{0, 1\}$, and applies the star operation.
$\Rightarrow$ $(0 \cup 1)^*$ represents the language of all possible strings of 0s and 1s.

Additional notation: For any alphabet $\Sigma = \{a_1, a_2, \ldots, a_k\}$, we let Σ stand for the regular expression $a_1 \cup a_2 \cup \cdots \cup a_k$, which represents the language consisting of all strings of length 1 over this alphabet.
$\Rightarrow$ The regular expression Σ^* represents the language of all strings over Σ.

Examples: Let $\Sigma = \{0, 1\}$. Then $\Sigma^* 1$ represents the language of all strings of 0s and 1s that end with a 1. Similarly, $(0\Sigma^*) \cup (\Sigma^* 1)$ represents the language of all strings of 0s and 1s that start with a 0 or end with a 1.

Notation

Remark 1: When we want to distinguish between a regular expression and the language that it represents, we let $L(R)$ denote the language that is represented by the regular expression R.

(This is consistent with the notation $L(M)$ for the language that is recognized by a DFA or NFA M.)

Notation

Remark 1: When we want to distinguish between a regular expression and the language that it represents, we let $L(R)$ denote the language that is represented by the regular expression R.

(This is consistent with the notation $L(M)$ for the language that is recognized by a DFA or NFA M.)

Remark 2: For convenience, we also introduce the notation R^+, defined by $R^+ = RR^*$.

In other words, R^+ represents the language consisting of all strings that are concatenations of one or more strings from $L(R)$.

Note that $L(R^+ \cup \varepsilon) = L(R^*)$.

Notation

Remark 1: When we want to distinguish between a regular expression and the language that it represents, we let $L(R)$ denote the language that is represented by the regular expression R.

(This is consistent with the notation $L(M)$ for the language that is recognized by a DFA or NFA M.)

Remark 2: For convenience, we also introduce the notation R^+, defined by $R^+ = RR^*$.

In other words, R^+ represents the language consisting of all strings that are concatenations of one or more strings from $L(R)$.

Note that $L(R^+ \cup \varepsilon) = L(R^*)$.

Remark 3: R^k is defined as follows: $R^k = \overbrace{RR\dots R}^{k \text{ times}}$

More Examples

Let the alphabet Σ be $\{0, 1\}$.

- $L(01 \cup 10) =$

More Examples

Let the alphabet Σ be $\{0, 1\}$.

- $L(01 \cup 10) = \{01,\ 10\}$

More Examples

Let the alphabet Σ be $\{0, 1\}$.

- $L(01 \cup 10) = \{01,\ 10\}$
- $L(0^*10^*) =$

More Examples

Let the alphabet Σ be $\{0, 1\}$.

- $L(01 \cup 10) = \{01,\ 10\}$
- $L(0^*10^*) = \{w \mid w \text{ contains exactly one } 1\}$

More Examples

Let the alphabet Σ be $\{0, 1\}$.

- $L(01 \cup 10) = \{01,\ 10\}$
- $L(0^*10^*) = \{w \mid w \text{ contains exactly one } 1\}$
- $L(\Sigma^*1\Sigma^*) =$

More Examples

Let the alphabet Σ be $\{0, 1\}$.

- $L(01 \cup 10) = \{01,\ 10\}$
- $L(0^*10^*) = \{w \mid w \text{ contains exactly one } 1\}$
- $L(\Sigma^*1\Sigma^*) = \{w \mid w \text{ contains at least one } 1\}$

More Examples

Let the alphabet Σ be $\{0, 1\}$.

- $L(01 \cup 10) = \{01,\ 10\}$
- $L(0^*10^*) = \{w \mid w \text{ contains exactly one } 1\}$
- $L(\Sigma^*1\Sigma^*) = \{w \mid w \text{ contains at least one } 1\}$
- $L(\Sigma^*001\Sigma^*) =$

More Examples

Let the alphabet Σ be $\{0, 1\}$.

- $L(01 \cup 10) = \{01,\ 10\}$
- $L(0^*10^*) = \{w \mid w \text{ contains exactly one } 1\}$
- $L(\Sigma^*1\Sigma^*) = \{w \mid w \text{ contains at least one } 1\}$
- $L(\Sigma^*001\Sigma^*) = \{w \mid w \text{ contains the string } 001 \text{ as a substring}\}$

More Examples

Let the alphabet Σ be $\{0, 1\}$.

- $L(01 \cup 10) = \{01,\ 10\}$
- $L(0^*10^*) = \{w \mid w \text{ contains exactly one } 1\}$
- $L(\Sigma^*1\Sigma^*) = \{w \mid w \text{ contains at least one } 1\}$
- $L(\Sigma^*001\Sigma^*) = \{w \mid w \text{ contains the string } 001 \text{ as a substring}\}$
- $L((\Sigma\Sigma)^*) =$

More Examples

Let the alphabet Σ be $\{0, 1\}$.

- $L(01 \cup 10) = \{01,\ 10\}$
- $L(0^*10^*) = \{w \mid w \text{ contains exactly one } 1\}$
- $L(\Sigma^*1\Sigma^*) = \{w \mid w \text{ contains at least one } 1\}$
- $L(\Sigma^*001\Sigma^*) = \{w \mid w \text{ contains the string } 001 \text{ as a substring}\}$
- $L((\Sigma\Sigma)^*) = \{w \mid w \text{ is a string of even length}\}$

More Examples

Let the alphabet Σ be $\{0, 1\}$.

- $L(01 \cup 10) = \{01,\ 10\}$
- $L(0^*10^*) = \{w \mid w \text{ contains exactly one } 1\}$
- $L(\Sigma^*1\Sigma^*) = \{w \mid w \text{ contains at least one } 1\}$
- $L(\Sigma^*001\Sigma^*) = \{w \mid w \text{ contains the string } 001 \text{ as a substring}\}$
- $L((\Sigma\Sigma)^*) = \{w \mid w \text{ is a string of even length}\}$
- $L(0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1) =$

More Examples

Let the alphabet Σ be $\{0, 1\}$.

- $L(01 \cup 10) = \{01,\ 10\}$
- $L(0^*10^*) = \{w \mid w$ contains exactly one $1\}$
- $L(\Sigma^*1\Sigma^*) = \{w \mid w$ contains at least one $1\}$
- $L(\Sigma^*001\Sigma^*) = \{w \mid w$ contains the string 001 as a substring$\}$
- $L((\Sigma\Sigma)^*) = \{w \mid w$ is a string of even length$\}$
- $L(0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1) =$
  $\{w \mid w$ starts and ends with the same symbol$\}$

More Examples

Let the alphabet Σ be $\{0, 1\}$.

- $L(01 \cup 10) = \{01,\ 10\}$
- $L(0^*10^*) = \{w \mid w \text{ contains exactly one } 1\}$
- $L(\Sigma^*1\Sigma^*) = \{w \mid w \text{ contains at least one } 1\}$
- $L(\Sigma^*001\Sigma^*) = \{w \mid w \text{ contains the string } 001 \text{ as a substring}\}$
- $L((\Sigma\Sigma)^*) = \{w \mid w \text{ is a string of even length}\}$
- $L(0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1) =$
  $\{w \mid w \text{ starts and ends with the same symbol}\}$
- $L((0 \cup \varepsilon)(1 \cup \varepsilon)) =$

More Examples

Let the alphabet Σ be $\{0, 1\}$.

- $L(01 \cup 10) = \{01,\ 10\}$
- $L(0^*10^*) = \{w \mid w \text{ contains exactly one } 1\}$
- $L(\Sigma^*1\Sigma^*) = \{w \mid w \text{ contains at least one } 1\}$
- $L(\Sigma^*001\Sigma^*) = \{w \mid w \text{ contains the string } 001 \text{ as a substring}\}$
- $L((\Sigma\Sigma)^*) = \{w \mid w \text{ is a string of even length}\}$
- $L(0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1) =$
  $\{w \mid w \text{ starts and ends with the same symbol}\}$
- $L((0 \cup \varepsilon)(1 \cup \varepsilon)) = \{\varepsilon,\ 0,\ 1,\ 01\}$

More Examples

Let the alphabet Σ be $\{0, 1\}$.

- $L(01 \cup 10) = \{01,\ 10\}$
- $L(0^*10^*) = \{w \mid w \text{ contains exactly one } 1\}$
- $L(\Sigma^*1\Sigma^*) = \{w \mid w \text{ contains at least one } 1\}$
- $L(\Sigma^*001\Sigma^*) = \{w \mid w \text{ contains the string } 001 \text{ as a substring}\}$
- $L((\Sigma\Sigma)^*) = \{w \mid w \text{ is a string of even length}\}$
- $L(0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1) =$
  $\{w \mid w \text{ starts and ends with the same symbol}\}$
- $L((0 \cup \varepsilon)(1 \cup \varepsilon)) = \{\varepsilon,\ 0,\ 1,\ 01\}$
- $L(1^*(01^+)^*) =$

More Examples

Let the alphabet Σ be $\{0, 1\}$.

- $L(01 \cup 10) = \{01,\ 10\}$
- $L(0^*10^*) = \{w \mid w \text{ contains exactly one } 1\}$
- $L(\Sigma^*1\Sigma^*) = \{w \mid w \text{ contains at least one } 1\}$
- $L(\Sigma^*001\Sigma^*) = \{w \mid w \text{ contains the string } 001 \text{ as a substring}\}$
- $L((\Sigma\Sigma)^*) = \{w \mid w \text{ is a string of even length}\}$
- $L(0\Sigma^*0 \cup 1\Sigma^*1 \cup 0 \cup 1) =$
  $\{w \mid w \text{ starts and ends with the same symbol}\}$
- $L((0 \cup \varepsilon)(1 \cup \varepsilon)) = \{\varepsilon,\ 0,\ 1,\ 01\}$
- $L(1^*(01^+)^*) = \{w \mid \text{every } 0 \text{ in } w \text{ is followed by at least one } 1\}$

Applications

- Regular expressions have an important role in computer science applications involving text.

  In particular, regular expressions are useful for pattern matching and input validation, and are supported by many text editors, Unix-based operating systems, Perl, Java, Python, etc.

Applications

- Regular expressions have an important role in computer science applications involving text.

  In particular, regular expressions are useful for pattern matching and input validation, and are supported by many text editors, Unix-based operating systems, Perl, Java, Python, etc.

  As another example, regular expressions are employed in the design of compilers that translate computer programs from one programming language to another.

  In the lexical analysis phase, the input text needs to be broken into a sequence of “tokens” that can be described with regular expressions.

  Example: A numerical constant (such as `72`, `3.14159`, `-.01`, etc.) may be described as a member of the language represented by $(+ \cup - \cup \varepsilon)\,(D^+ \cup (D^+ . \, D^*) \cup (D^* . \, D^+))$, where $D = \{0, 1, 2, \ldots, 9\}$.

Regular Expressions vs. Finite Automata

Theorem (1.54)

A language is regular if and only if some regular expression describes it.

- This theorem has two directions:
- If a language is described by a regular expression, then it is regular.
- If a language is regular, then it is described by a regular expression.
- We prove them separately (Lemma 1.55 and Lemma 1.60).

Regular Expressions vs. Finite Automata

Lemma (1.55)

If a language is described by a regular expression, then it is regular.

To prove the lemma, we will show how to convert any regular expression R over an alphabet Σ into an NFA N with $L(N) = L(R)$.

(The following method just finds an NFA that works, i.e., it does not necessarily find an NFA equivalent to R with the fewest possible states.)

By Definition 1.52 given earlier today, there are six possibilities for R.

Regular Expressions vs. Finite Automata

Definition (1.52)

We say that R is a *regular expression* if R is

1. a for some $a \in \Sigma$,
2. ε,
3. $\emptyset$,
4. $(R_1 \cup R_2)$, where R_1 and R_2 are regular expressions,
5. $(R_1 \circ R_2)$, where R_1 and R_2 are regular expressions, or
6. (R_1^*), where R_1 is a regular expression.

Regular Expressions vs. Finite Automata

Lemma (1.55)

If a language is described by a regular expression, then it is regular.

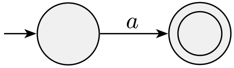
To prove the lemma, we will show how to convert any regular expression R over an alphabet Σ into an NFA N with $L(N) = L(R)$.

(The following method just finds an NFA that works, i.e., it does not necessarily find an NFA equivalent to R with the fewest possible states.)

By Definition 1.52 given earlier today, there are six possibilities for R.

For case 1.:

1. $R = a$ for some $a \in \Sigma$.
   $N = (\{q_1, q_2\}, \Sigma, \delta, q_1, \{q_2\})$, where $\delta(q_1, a) = \{q_2\}$,
   $\delta(r, b) = \emptyset$ for $r \neq q_1$ or $b \neq a$.



Regular Expressions vs. Finite Automata

Lemma (1.55)

If a language is described by a regular expression, then it is regular.

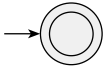
To prove the lemma, we will show how to convert any regular expression R over an alphabet Σ into an NFA N with $L(N) = L(R)$.

(The following method just finds an NFA that works, i.e., it does not necessarily find an NFA equivalent to R with the fewest possible states.)

By Definition 1.52 given earlier today, there are six possibilities for R.

For case 2.:

2. $R = \varepsilon$.
   $N = (\{q\}, \Sigma, \delta, q, \{q\})$, where $\delta(r, b) = \emptyset$ for any r and b.



Regular Expressions vs. Finite Automata

Lemma (1.55)

If a language is described by a regular expression, then it is regular.

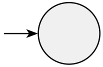
To prove the lemma, we will show how to convert any regular expression R over an alphabet Σ into an NFA N with $L(N) = L(R)$.

(The following method just finds an NFA that works, i.e., it does not necessarily find an NFA equivalent to R with the fewest possible states.)

By Definition 1.52 given earlier today, there are six possibilities for R.

For case 3.:

3. $R = \emptyset$.
   $N = (\{q\}, \Sigma, \delta, q, \emptyset)$, where $\delta(r, b) = \emptyset$ for any r and b.



Regular Expressions vs. Finite Automata

Lemma (1.55)

If a language is described by a regular expression, then it is regular.

To prove the lemma, we will show how to convert any regular expression R over an alphabet Σ into an NFA N with $L(N) = L(R)$.

(The following method just finds an NFA that works, i.e., it does not necessarily find an NFA equivalent to R with the fewest possible states.)

By Definition 1.52 given earlier today, there are six possibilities for R.

For cases 4., 5., and 6.:

Use the constructions demonstrated in Figures 1.46, 1.48, and 1.50 from Chapter 1.2.

(From Chapter 1.2)

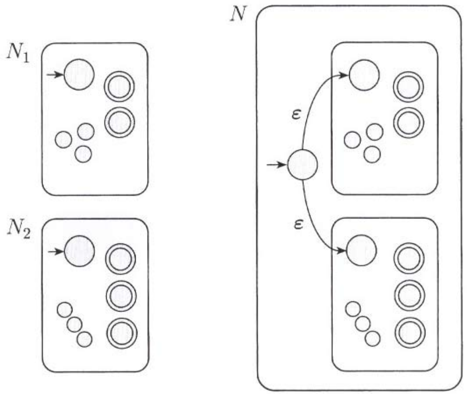


FIGURE **1.46**
Construction of an NFA N to recognize $A_1 \cup A_2$

(From Chapter 1.2)

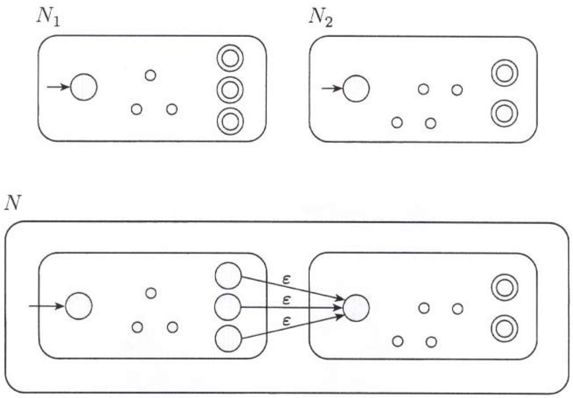


FIGURE 1.48
Construction of N to recognize $A_1 \circ A_2$

(From Chapter 1.2)

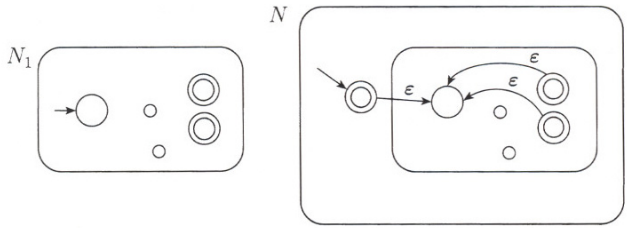


FIGURE 1.50
Construction of N to recognize A^*

Regular Expressions vs. Finite Automata

Lemma (1.55)

If a language is described by a regular expression, then it is regular.

Summary of the six cases in the proof:

1. $R = a$ for some $a \in \Sigma$.
   $N = (\{q_1, q_2\}, \Sigma, \delta, q_1, \{q_2\})$, where $\delta(q_1, a) = \{q_2\}$,
   $\delta(r, b) = \emptyset$ for $r \neq q_1$ or $b \neq a$.
2. $R = \varepsilon$.
   $N = (\{q\}, \Sigma, \delta, q, \{q\})$, where $\delta(r, b) = \emptyset$ for any r and b.
3. $R = \emptyset$.
   $N = (\{q\}, \Sigma, \delta, q, \emptyset)$, where $\delta(r, b) = \emptyset$ for any r and b.
4. $R = R_1 \cup R_2$. Closed under union.
5. $R = R_1 \circ R_2$. Closed under concatenation.
6. $R = R_1^*$. Closed under star.

Converting a Regular Expression into an NFA

Example 1:

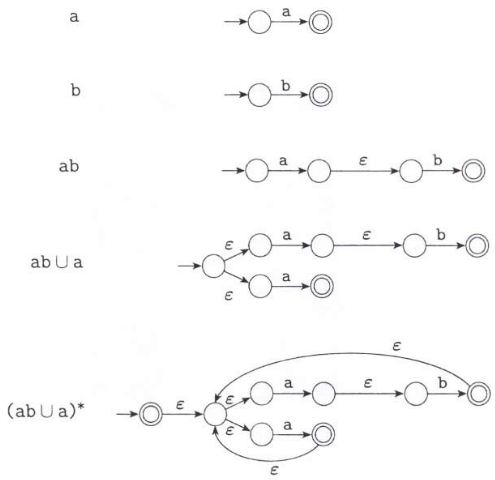


FIGURE 1.57
Building an NFA from the regular expression $(ab \cup a)^*$

Converting a Regular Expression into an NFA

Example 2:

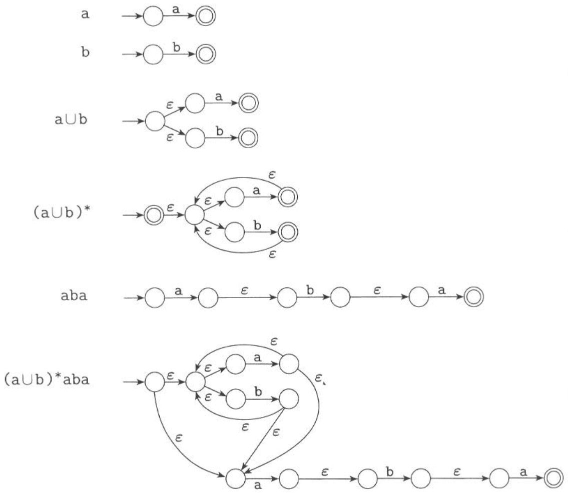


FIGURE **1.59**
Building an NFA from the regular expression $(a \cup b)^*aba$

Regular Expressions vs. Finite Automata

Next, we will prove the other direction of Theorem 1.54.

Lemma (1.60)

If a language is regular, then it is described by a regular expression.

- Every regular language is recognized by some DFA.
- We describe a procedure for converting DFAs into equivalent regular expressions.
- For this purpose, we introduce a new type of finite automaton called a *generalized nondeterministic finite automaton* (GNFA).
- We show how to convert DFAs into GNFAs and then GNFAs into regular expressions.

Regular Expressions vs. Finite Automata

In a GNFA, the transition arrows are labeled by regular expressions.

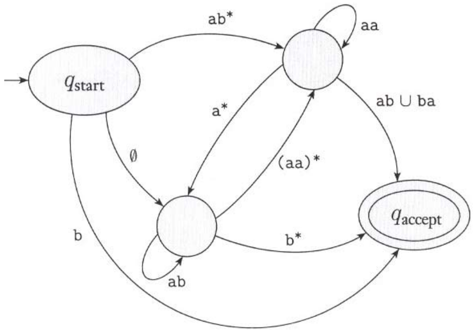


FIGURE **1.61**
A generalized nondeterministic finite automaton

Regular Expressions vs. Finite Automata

In a GNFA, the transition arrows are labeled by regular expressions.

Every non-accept state has an arrow to every state except the start state.

There is exactly one accept state. It must be different from the start state and it has no outgoing arrows.

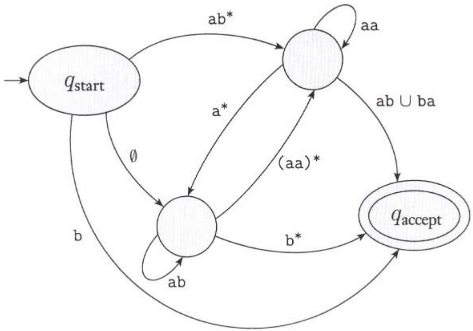


FIGURE **1.61**
A generalized nondeterministic finite automaton

Definition of a GNFA

Definition (1.52)

A *generalized nondeterministic finite automaton* is a 5-tuple $(Q, \Sigma, \delta, q_{\text{start}}, q_{\text{accept}})$, where

1. Q is the finite set of states,
2. Σ is the input alphabet,
3. $\delta : (Q - \{q_{\text{accept}}\}) \times (Q - \{q_{\text{start}}\}) \longrightarrow \mathcal{R}$ is the transition function (where $\mathcal{R}$ is the collection of all regular expressions over Σ),
4. q_{start} is the start state, and
5. q_{accept} is the accept state.

Computation of a GNFA

A GNFA accepts a string w in Σ^* if $w = w_1 w_2 \dots$
w_k, where each w_i is in Σ^*, and a sequence of states $q_0, q_1, \dots, q_k$ exists such that

1. $q_0 = q_{\mathrm{start}}$,
2. $q_k = q_{\mathrm{accept}}$, and
3. for each i, we have $w_i \in L(R_i)$, where $R_i = \delta(q_{i-1}, q_i)$.

Regular Expressions vs. Finite Automata

Lemma (1.60)

If a language is regular, then it is described by a regular expression.

Proof: Let A be any regular language and let M be a DFA recognizing A. To construct a regular expression R with $L(R) = L(M)$:

Regular Expressions vs. Finite Automata

Lemma (1.60)

If a language is regular, then it is described by a regular expression.

Proof: Let A be any regular language and let M be a DFA recognizing A. To construct a regular expression R with $L(R) = L(M)$:

1. Convert M into an equivalent GNFA G with two extra states as follows: For each pair of states and direction, replace all transitions by a single transition labeled by the union of the labels. Add a new start state with an ε-transition to the old one, a new accept state with ε-transitions from the old ones, and $\emptyset$-transitions wherever arrows are missing.
2. Let k be the number of states in G.
3. If $k = 2$: Let R be the label on the only transition in G and stop.
4. If $k > 2$: Convert G into an equivalent GNFA G' with $k - 1$ states as described on the next two slides, let $G = G'$, and go to step 2.

Converting a GNFA

In step 4, use the following method to convert G into an equivalent GNFA G' with $k-1$ states:

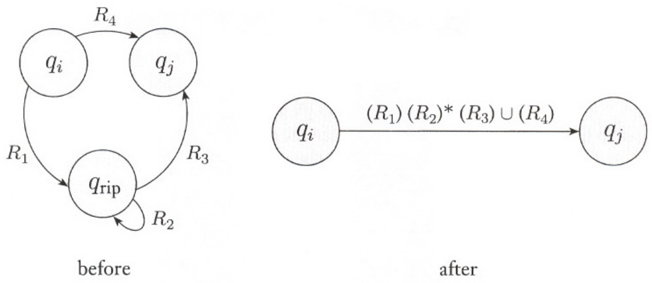


FIGURE 1.63
Constructing an equivalent GNFA with one fewer state

Converting a GNFA

More precisely, in step 4.:

If $k > 2$, select $q_{\mathrm{rip}} \in Q$ different from q_{start} and q_{accept}.
Let G' be $(Q', \Sigma, \delta', q_{\mathrm{start}}, q_{\mathrm{accept}})$, where

$$Q' = Q - \{q_{\mathrm{rip}}\}$$

and for any $q_i \in Q' - \{q_{\mathrm{accept}}\}$ and any $q_j \in Q' - \{q_{\mathrm{start}}\}$,

$$\delta'(q_i, q_j) = (R_1)(R_2)^*(R_3) \cup (R_4),$$

where $R_1 = \delta(q_i, q_{\mathrm{rip}})$, $R_2 = \delta(q_{\mathrm{rip}}, q_{\mathrm{rip}})$, $R_3 = \delta(q_{\mathrm{rip}}, q_j)$, and
$R_4 = \delta(q_i, q_j)$.

Then G' has $k - 1$ states and is equivalent to G.

Converting a DFA into a Regular Expression

In summary, any DFA can be converted into a regular expression by applying the construction in the proof of Lemma 1.60.

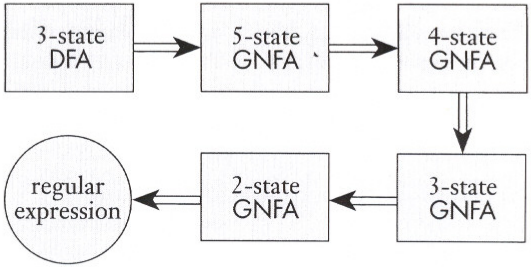


FIGURE 1.62
Typical stages in converting a DFA to a regular expression

Converting a DFA into a Regular Expression

Example 3:

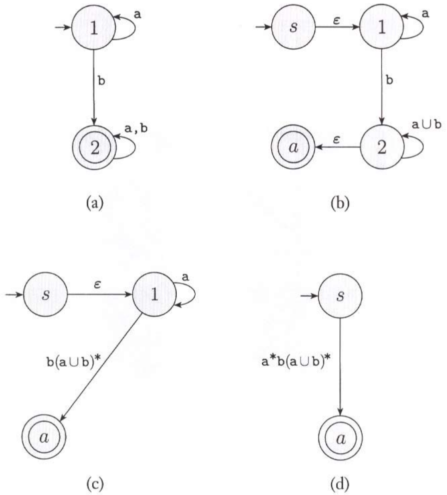


FIGURE **1.67**
Converting a two-state DFA to an equivalent regular expression

Converting a DFA into a Regular Expression

Example 4:

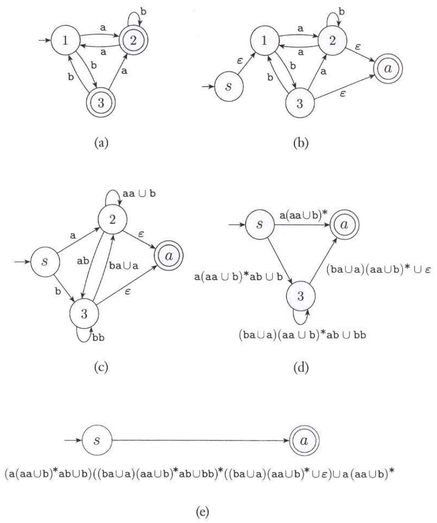


FIGURE 1.69
Converting a three-state DFA to an equivalent regular expression