Standard Notation and Local Variables

After reading this section, you might imagine that our customary notation is required; that is, you *must* use δ for the transition function, use A for the name of a DFA, and so on. We tend to use the same variables to denote the same thing across all examples, because it helps to remind you of the types of variables, much the way a variable i in a program is almost always of integer type. However, we are free to call the components of an automaton, or anything else, anything we wish. Thus, you are free to call a DFA M and its transition function T if you like.

Moreover, you should not be surprised that the same variable means different things in different contexts. For example, the DFA's of Examples 2.1 and 2.4 both were given a transition function called δ. However, the two transition functions are each local variables, belonging only to their examples. These two transition functions are very different and bear no relationship to one another.

2.2.5 The Language of a DFA

Now, we can define the *language* of a DFA $A = (Q, \Sigma, \delta, q_0, F)$. This language is denoted $L(A)$, and is defined by

$$L(A) = \{w \mid \hat{\delta}(q_0, w) \text{ is in } F\}$$

That is, the language of A is the set of strings w that take the start state q_0 to one of the accepting states. If L is $L(A)$ for some DFA A, then we say L is a *regular language*.

Example 2.5: As we mentioned earlier, if A is the DFA of Example 2.1, then $L(A)$ is the set of all strings of 0's and 1's that contain a substring 01. If A is instead the DFA of Example 2.4, then $L(A)$ is the set of all strings of 0's and 1's whose numbers of 0's and 1's are both even. □

2.2.6 Exercises for Section 2.2

Exercise 2.2.1: In Fig. 2.8 is a marble-rolling toy. A marble is dropped at A or B. Levers x_1, x_2, and x_3 cause the marble to fall either to the left or to the right. Whenever a marble encounters a lever, it causes the lever to reverse after the marble passes, so the next marble will take the opposite branch.

***** a) Model this toy by a finite automaton. Let the inputs A and B represent the input into which the marble is dropped. Let acceptance correspond to the marble exiting at D; nonacceptance represents a marble exiting at C.

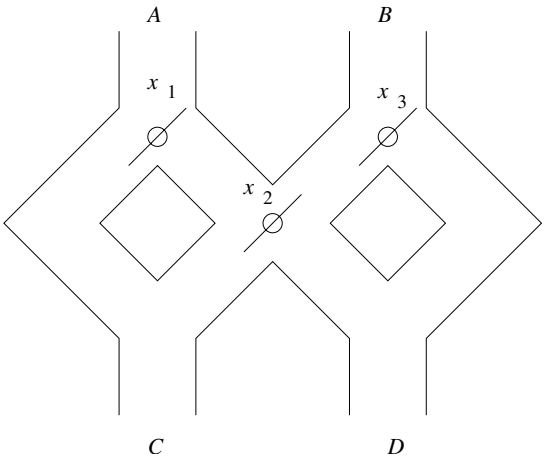

Figure 2.8: A marble-rolling toy

! b) Informally describe the language of the automaton.

c) Suppose that instead the levers switched *before* allowing the marble to pass. How would your answers to parts (a) and (b) change?

*! **Exercise 2.2.2:** We defined $\hat{\delta}$ by breaking the input string into any string followed by a single symbol (in the inductive part, Equation 2.1). However, we informally think of $\hat{\delta}$ as describing what happens along a path with a certain string of labels, and if so, then it should not matter how we break the input string in the definition of $\hat{\delta}$. Show that in fact, $\hat{\delta}(q, xy) = \hat{\delta}\big(\hat{\delta}(q, x), y\big)$ for any state q and strings x and y. *Hint*: Perform an induction on $|y|$.

! **Exercise 2.2.3:** Show that for any state q, string x, and input symbol a, $\hat{\delta}(q, ax) = \hat{\delta}\big(\delta(q, a), x\big)$. *Hint*: Use Exercise 2.2.2.

Exercise 2.2.4: Give DFA's accepting the following languages over the alphabet $\{0, 1\}$:

* a) The set of all strings ending in 00.

b) The set of all strings with three consecutive 0's (not necessarily at the end).

c) The set of strings with 011 as a substring.

! **Exercise 2.2.5:** Give DFA's accepting the following languages over the alphabet $\{0, 1\}$:

a) The set of all strings such that each block of five consecutive symbols contains at least two 0's.

b) The set of all strings whose tenth symbol from the right end is a 1.

c) The set of strings that either begin or end (or both) with 01.

d) The set of strings such that the number of 0's is divisible by five, and the number of 1's is divisible by 3.

!! **Exercise 2.2.6:** Give DFA's accepting the following languages over the alphabet $\{0,1\}$:

* a) The set of all strings beginning with a 1 that, when interpreted as a binary integer, is a multiple of 5. For example, strings 101, 1010, and 1111 are in the language; 0, 100, and 111 are not.

b) The set of all strings that, when interpreted *in reverse* as a binary integer, is divisible by 5. Examples of strings in the language are 0, 10011, 1001100, and 0101.

Exercise 2.2.7: Let A be a DFA and q a particular state of A, such that $\delta(q,a) = q$ for all input symbols a. Show by induction on the length of the input that for all input strings w, $\hat{\delta}(q,w) = q$.

Exercise 2.2.8: Let A be a DFA and a a particular input symbol of A, such that for all states q of A we have $\delta(q,a) = q$.

a) Show by induction on n that for all $n \geq 0$, $\hat{\delta}(q,a^n) = q$, where a^n is the string consisting of n a's.

b) Show that either $\{a\}^* \subseteq L(A)$ or $\{a\}^* \cap L(A) = \emptyset$.

*! **Exercise 2.2.9:** Let $A = (Q, \Sigma, \delta, q_0, \{q_f\})$ be a DFA, and suppose that for all a in Σ we have $\delta(q_0,a) = \delta(q_f,a)$.

a) Show that for all $w \neq \epsilon$ we have $\hat{\delta}(q_0,w) = \hat{\delta}(q_f,w)$.

b) Show that if x is a nonempty string in $L(A)$, then for all $k > 0$, x^k (i.e., x written k times) is also in $L(A)$.

*! **Exercise 2.2.10:** Consider the DFA with the following transition table:

	0	1
$\rightarrow A$	A	B
$*B$	B	A

Informally describe the language accepted by this DFA, and prove by induction on the length of an input string that your description is correct. *Hint*: When setting up the inductive hypothesis, it is wise to make a statement about what inputs get you to each state, not just what inputs get you to the accepting state.

! **Exercise 2.2.11 :** Repeat Exercise 2.2.10 for the following transition table:

	0	1
$\rightarrow *A$	B	A
$*B$	C	A
C	C	C

2.3 Nondeterministic Finite Automata

A "nondeterministic" finite automaton (*NFA*) has the power to be in several states at once. This ability is often expressed as an ability to "guess" something about its input. For instance, when the automaton is used to search for certain sequences of characters (e.g., keywords) in a long text string, it is helpful to "guess" that we are at the beginning of one of those strings and use a sequence of states to do nothing but check that the string appears, character by character. We shall see an example of this type of application in Section 2.4.

Before examining applications, we need to define nondeterministic finite automata and show that each one accepts a language that is also accepted by some DFA. That is, the NFA's accept exactly the regular languages, just as DFA's do. However, there are reasons to think about NFA's. They are often more succinct and easier to design than DFA's. Moreover, while we can always convert an NFA to a DFA, the latter may have exponentially more states than the NFA; fortunately, cases of this type are rare.

2.3.1 An Informal View of Nondeterministic Finite Automata

Like the DFA, an NFA has a finite set of states, a finite set of input symbols, one start state and a set of accepting states. It also has a transition function, which we shall commonly call δ. The difference between the DFA and the NFA is in the type of δ. For the NFA, δ is a function that takes a state and input symbol as arguments (like the DFA's transition function), but returns a set of zero, one, or more states (rather than returning exactly one state, as the DFA must). We shall start with an example of an NFA, and then make the definitions precise.

Example 2.6 : Figure 2.9 shows a nondeterministic finite automaton, whose job is to accept all and only the strings of 0's and 1's that end in 01. State q_0 is the start state, and we can think of the automaton as being in state q_0 (perhaps among other states) whenever it has not yet "guessed" that the final 01 has begun. It is always possible that the next symbol does not begin the final 01, even if that symbol is 0. Thus, state q_0 may transition to itself on both 0 and 1.

However, if the next symbol is 0, this NFA also guesses that the final 01 has begun. An arc labeled 0 thus leads from q_0 to state q_1. Notice that there are

than 2^n states, then there would be some state q such that D can be in state q after reading two different sequences of n bits, say $a_1a_2\cdots a_n$ and $b_1b_2\cdots b_n$.

Since the sequences are different, they must differ in some position, say $a_i \neq b_i$. Suppose (by symmetry) that $a_i = 1$ and $b_i = 0$. If $i = 1$, then q must be both an accepting state and a nonaccepting state, since $a_1a_2\cdots a_n$ is accepted (the nth symbol from the end is 1) and $b_1b_2\cdots b_n$ is not. If $i > 1$, then consider the state p that D enters after reading $i-1$ 0's. Then p must be both accepting and nonaccepting, since $a_ia_{i+1}\cdots a_n00\cdots 0$ is accepted and $b_ib_{i+1}\cdots b_n00\cdots 0$ is not.

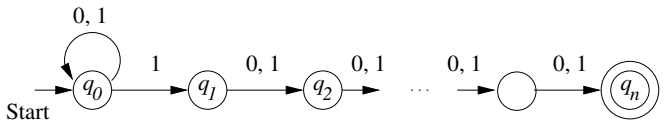

Figure 2.15: This NFA has no equivalent DFA with fewer than 2^n states

Now, let us see how the NFA N of Fig. 2.15 works. There is a state q_0 that the NFA is always in, regardless of what inputs have been read. If the next input is 1, N may also "guess" that this 1 will be the nth symbol from the end, so it goes to state q_1 as well as q_0. From state q_1, any input takes N to q_2, the next input takes it to q_3, and so on, until $n-1$ inputs later, it is in the accepting state q_n. The formal statement of what the states of N do is:

1. N is in state q_0 after reading any sequence of inputs w.
2. N is in state q_i, for $i = 1, 2, \ldots, n$, after reading input sequence w if and only if the ith symbol from the end of w is 1; that is, w is of the form $x1a_1a_2\cdots a_{i-1}$, where the a_j's are each input symbols.

We shall not prove these statements formally; the proof is an easy induction on $|w|$, mimicking Example 2.9. To complete the proof that the automaton accepts exactly those strings with a 1 in the nth position from the end, we consider statement (2) with $i = n$. That says N is in state q_n if and only if the nth symbol from the end is 1. But q_n is the only accepting state, so that condition also characterizes exactly the set of strings accepted by N. □

2.3.7 Exercises for Section 2.3

*** Exercise 2.3.1:** Convert to a DFA the following NFA:

	0	1
$\rightarrow p$	$\{p, q\}$	$\{p\}$
q	$\{r\}$	$\{r\}$
r	$\{s\}$	$\emptyset$
$*s$	$\{s\}$	$\{s\}$

The Pigeonhole Principle

In Example 2.13 we used an important reasoning technique called the *pigeonhole principle*. Colloquially, if you have more pigeons than pigeonholes, and each pigeon flies into some pigeonhole, then there must be at least one hole that has more than one pigeon. In our example, the "pigeons" are the sequences of n bits, and the "pigeonholes" are the states. Since there are fewer states than sequences, one state must be assigned two sequences.

The pigeonhole principle may appear obvious, but it actually depends on the number of pigeonholes being finite. Thus, it works for finite-state automata, with the states as pigeonholes, but does not apply to other kinds of automata that have an infinite number of states.

To see why the finiteness of the number of pigeonholes is essential, consider the infinite situation where the pigeonholes correspond to integers $1, 2, \ldots$. Number the pigeons $0, 1, 2, \ldots$, so there is one more pigeon than there are pigeonholes. However, we can send pigeon i to hole $i + 1$ for all $i \geq 0$. Then each of the infinite number of pigeons gets a pigeonhole, and no two pigeons have to share a pigeonhole.

Exercise 2.3.2: Convert to a DFA the following NFA:

	0	1
$\rightarrow p$	$\{q, s\}$	$\{q\}$
$*q$	$\{r\}$	$\{q, r\}$
r	$\{s\}$	$\{p\}$
$*s$	$\emptyset$	$\{p\}$

! Exercise 2.3.3: Convert the following NFA to a DFA and informally describe the language it accepts.

	0	1
$\rightarrow p$	$\{p, q\}$	$\{p\}$
q	$\{r, s\}$	$\{t\}$
r	$\{p, r\}$	$\{t\}$
$*s$	$\emptyset$	$\emptyset$
$*t$	$\emptyset$	$\emptyset$

! Exercise 2.3.4: Give nondeterministic finite automata to accept the following languages. Try to take advantage of nondeterminism as much as possible.

Dead States and DFA's Missing Some Transitions

We have formally defined a DFA to have a transition from any state, on any input symbol, to exactly one state. However, sometimes, it is more convenient to design the DFA to "die" in situations where we know it is impossible for any extension of the input sequence to be accepted. For instance, observe the automaton of Fig. 1.2, which did its job by recognizing a single keyword, `then`, and nothing else. Technically, this automaton is not a DFA, because it lacks transitions on most symbols from each of its states.

However, such an automaton is an NFA. If we use the subset construction to convert it to a DFA, the automaton looks almost the same, but it includes a *dead state*, that is, a nonaccepting state that goes to itself on every possible input symbol. The dead state corresponds to $\emptyset$, the empty set of states of the automaton of Fig. 1.2.

In general, we can add a dead state to any automaton that has *no more* than one transition for any state and input symbol. Then, add a transition to the dead state from each other state q, on all input symbols for which q has no other transition. The result will be a DFA in the strict sense. Thus, we shall sometimes refer to an automaton as a DFA if it has *at most* one transition out of any state on any symbol, rather than if it has *exactly one* transition.

***** a) The set of strings over alphabet $\{0, 1, \ldots, 9\}$ such that the final digit has appeared before.

b) The set of strings over alphabet $\{0, 1, \ldots, 9\}$ such that the final digit has *not* appeared before.

c) The set of strings of 0's and 1's such that there are two 0's separated by a number of positions that is a multiple of 4. Note that 0 is an allowable multiple of 4.

Exercise 2.3.5: In the only-if portion of Theorem 2.12 we omitted the proof by induction on $|w|$ that if $\hat{\delta}_D(q_0, w) = p$ then $\hat{\delta}_N(q_0, w) = \{p\}$. Supply this proof.

! **Exercise 2.3.6:** In the box on "Dead States and DFA's Missing Some Transitions," we claim that if N is an NFA that has at most one choice of state for any state and input symbol (i.e., $\delta(q, a)$ never has size greater than 1), then the DFA D constructed from N by the subset construction has exactly the states and transitions of N plus transitions to a new dead state whenever N is missing a transition for a given state and input symbol. Prove this contention.

Exercise 2.3.7: In Example 2.13 we claimed that the NFA N is in state q_i, for $i = 1, 2, \ldots, n$, after reading input sequence w if and only if the ith symbol from the end of w is 1. Prove this claim.

2.4 An Application: Text Search

In this section, we shall see that the abstract study of the previous section, where we considered the "problem" of deciding whether a sequence of bits ends in 01, is actually an excellent model for several real problems that appear in applications such as Web search and extraction of information from text.

2.4.1 Finding Strings in Text

A common problem in the age of the Web and other on-line text repositories is the following. Given a set of words, find all documents that contain one (or all) of those words. A search engine is a popular example of this process. The search engine uses a particular technology, called *inverted indexes*, where for each word appearing on the Web (there are 100,000,000 different words), a list of all the places where that word occurs is stored. Machines with very large amounts of main memory keep the most common of these lists available, allowing many people to search for documents at once.

Inverted-index techniques do not make use of finite automata, but they also take very large amounts of time for crawlers to copy the Web and set up the indexes. There are a number of related applications that are unsuited for inverted indexes, but are good applications for automaton-based techniques. The characteristics that make an application suitable for searches that use automata are:

1. The repository on which the search is conducted is rapidly changing. For example:
   (a) Every day, news analysts want to search the day's on-line news articles for relevant topics. For example, a financial analyst might search for certain stock ticker symbols or names of companies.
   (b) A "shopping robot" wants to search for the current prices charged for the items that its clients request. The robot will retrieve current catalog pages from the Web and then search those pages for words that suggest a price for a particular item.
2. The documents to be searched cannot be cataloged. For example, Amazon.com does not make it easy for crawlers to find all the pages for all the books that the company sells. Rather, these pages are generated "on the fly" in response to queries. However, we could send a query for books on a certain topic, say "finite automata," and then search the pages retrieved for certain words, e.g., "excellent" in a review portion.

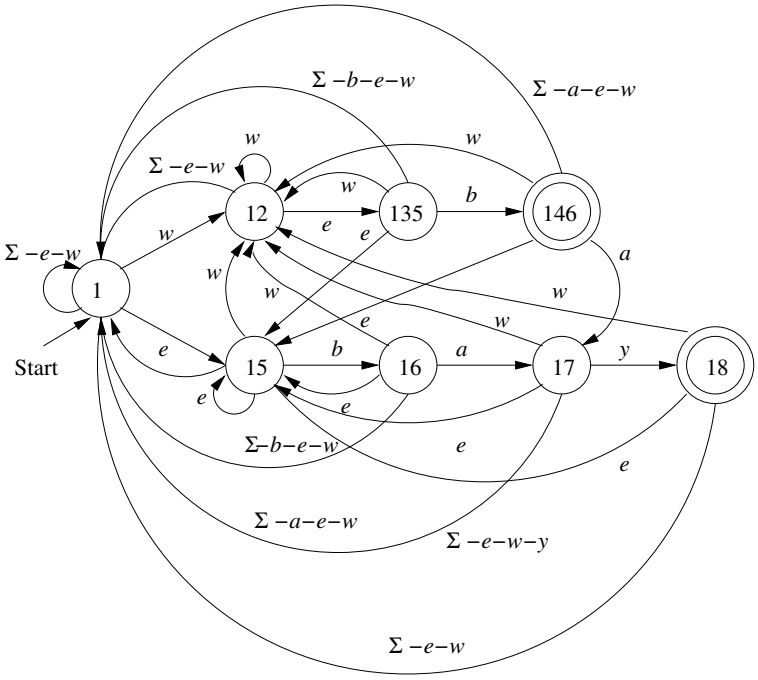

Figure 2.17: Conversion of the NFA from Fig. 2.16 to a DFA

each symbol x, where the p_i's go in the NFA, and let this DFA state have a transition labeled x to the DFA state consisting of q_0 and all the targets of the p_i's and q_0 on symbol x. On all symbols x such that there are no transitions out of any of the p_i's on symbol x, let this DFA state have a transition on x to that state of the DFA consisting of q_0 and all states that are reached from q_0 in the NFA following an arc labeled x.

For instance, consider state 135 of Fig. 2.17. The NFA of Fig. 2.16 has transitions on symbol b from states 3 and 5 to states 4 and 6, respectively. Therefore, on symbol b, 135 goes to 146. On symbol e, there are no transitions of the NFA out of 3 or 5, but there is a transition from 1 to 5. Thus, in the DFA, 135 goes to 15 on input e. Similarly, on input w, 135 goes to 12.

On every other symbol x, there are no transitions out of 3 or 5, and state 1 goes only to itself. Thus, there are transitions from 135 to 1 on every symbol in Σ other than b, e, and w. We use the notation $\Sigma - b - e - w$ to represent this set, and use similar representations of other sets in which a few symbols are removed from Σ. □

2.4.4 Exercises for Section 2.4

Exercise 2.4.1: Design NFA's to recognize the following sets of strings.

* a) `abc`, `abd`, and `aacd`. Assume the alphabet is $\{a, b, c, d\}$.

b) `0101`, `101`, and `011`.

c) `ab`, `bc`, and `ca`. Assume the alphabet is $\{a, b, c\}$.

Exercise 2.4.2: Convert each of your NFA's from Exercise 2.4.1 to DFA's.

2.5 Finite Automata With Epsilon-Transitions

We shall now introduce another extension of the finite automaton. The new "feature" is that we allow a transition on ϵ, the empty string. In effect, an NFA is allowed to make a transition spontaneously, without receiving an input symbol. Like the nondeterminism added in Section 2.3, this new capability does not expand the class of languages that can be accepted by finite automata, but it does give us some added "programming convenience." We shall also see, when we take up regular expressions in Section 3.1, how NFA's with ϵ-transitions, which we call *ϵ-NFA's*, are closely related to regular expressions and useful in proving the equivalence between the classes of languages accepted by finite automata and by regular expressions.

2.5.1 Uses of ϵ-Transitions

We shall begin with an informal treatment of ϵ-NFA's, using transition diagrams with ϵ allowed as a label. In the examples to follow, think of the automaton as accepting those sequences of labels along paths from the start state to an accepting state. However, each ϵ along a path is "invisible"; i.e., it contributes nothing to the string along the path.

Example 2.16: In Fig. 2.18 is an ϵ-NFA that accepts decimal numbers consisting of:

1. An optional + or − sign,
2. A string of digits,
3. A decimal point, and
4. Another string of digits. Either this string of digits, or the string (2) can be empty, but at least one of the two strings of digits must be nonempty.

Of particular interest is the transition from q_0 to q_1 on any of ϵ, +, or −. Thus, state q_1 represents the situation in which we have seen the sign if there is one, and perhaps some digits, but not the decimal point. State q_2 represents the situation where we have just seen the decimal point, and may or may not have seen prior digits. In q_4 we have definitely seen at least one digit, but not the decimal point. Thus, the interpretation of q_3 is that we have seen a

BASIS: If $|w| = 0$, then $w = \epsilon$. We know $\hat{\delta}_E(q_0, \epsilon) = \text{ECLOSE}(q_0)$. We also know that $q_D = \text{ECLOSE}(q_0)$, because that is how the start state of D is defined. Finally, for a DFA, we know that $\hat{\delta}(p, \epsilon) = p$ for any state p, so in particular, $\hat{\delta}_D(q_D, \epsilon) = \text{ECLOSE}(q_0)$. We have thus proved that $\hat{\delta}_E(q_0, \epsilon) = \hat{\delta}_D(q_D, \epsilon)$.

INDUCTION: Suppose $w = xa$, where a is the final symbol of w, and assume that the statement holds for x. That is, $\hat{\delta}_E(q_0, x) = \hat{\delta}_D(q_D, x)$. Let both these sets of states be $\{p_1, p_2, \ldots, p_k\}$.

By the definition of $\hat{\delta}$ for ϵ-NFA's, we compute $\hat{\delta}_E(q_0, w)$ by:

1. Let $\{r_1, r_2, \ldots, r_m\}$ be $\bigcup_{i=1}^{k} \delta_E(p_i, a)$.
2. Then $\hat{\delta}_E(q_0, w) = \text{ECLOSE}(\{r_1, r_2, \ldots, r_m\})$.

If we examine the construction of DFA D in the modified subset construction above, we see that $\delta_D(\{p_1, p_2, \ldots, p_k\}, a)$ is constructed by the same two steps (1) and (2) above. Thus, $\hat{\delta}_D(q_D, w)$, which is $\delta_D(\{p_1, p_2, \ldots, p_k\}, a)$ is the same set as $\hat{\delta}_E(q_0, w)$. We have now proved that $\hat{\delta}_E(q_0, w) = \hat{\delta}_D(q_D, w)$ and completed the inductive part. □

2.5.6 Exercises for Section 2.5

*** Exercise 2.5.1:** Consider the following ϵ-NFA.

	ϵ	a	b	c
$\to p$	$\emptyset$	$\{p\}$	$\{q\}$	$\{r\}$
q	$\{p\}$	$\{q\}$	$\{r\}$	$\emptyset$
$*r$	$\{q\}$	$\{r\}$	$\emptyset$	$\{p\}$

a) Compute the ϵ-closure of each state.

b) Give all the strings of length three or less accepted by the automaton.

c) Convert the automaton to a DFA.

Exercise 2.5.2: Repeat Exercise 2.5.1 for the following ϵ-NFA:

	ϵ	a	b	c
$\to p$	$\{q, r\}$	$\emptyset$	$\{q\}$	$\{r\}$
q	$\emptyset$	$\{p\}$	$\{r\}$	$\{p, q\}$
$*r$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Exercise 2.5.3: Design ϵ-NFA's for the following languages. Try to use ϵ-transitions to simplify your design.

a) The set of strings consisting of zero or more a's followed by zero or more b's, followed by zero or more c's.

! b) The set of strings that consist of either 01 repeated one or more times or 010 repeated one or more times.

! c) The set of strings of 0's and 1's such that at least one of the last ten positions is a 1.

2.6 Summary of Chapter 2

- *Deterministic Finite Automata*: A DFA has a finite set of states and a finite set of input symbols. One state is designated the start state, and zero or more states are accepting states. A transition function determines how the state changes each time an input symbol is processed.

- *Transition Diagrams*: It is convenient to represent automata by a graph in which the nodes are the states, and arcs are labeled by input symbols, indicating the transitions of that automaton. The start state is designated by an arrow, and the accepting states by double circles.

- *Language of an Automaton*: The automaton accepts strings. A string is accepted if, starting in the start state, the transitions caused by processing the symbols of that string one-at-a-time lead to an accepting state. In terms of the transition diagram, a string is accepted if it is the label of a path from the start state to some accepting state.

- *Nondeterministic Finite Automata*: The NFA differs from the DFA in that the NFA can have any number of transitions (including zero) to next states from a given state on a given input symbol.

- *The Subset Construction*: By treating sets of states of an NFA as states of a DFA, it is possible to convert any NFA to a DFA that accepts the same language.

- ϵ-*Transitions*: We can extend the NFA by allowing transitions on an empty input, i.e., no input symbol at all. These extended NFA's can be converted to DFA's accepting the same language.

- *Text-Searching Applications*: Nondeterministic finite automata are a useful way to represent a pattern matcher that scans a large body of text for one or more keywords. These automata are either simulated directly in software or are first converted to a DFA, which is then simulated.

2.7 Gradiance Problems for Chapter 2

The following is a sample of problems that are available on-line through the Gradiance system at `www.gradiance.com/pearson`. Each of these problems is worked like conventional homework. The Gradiance system gives you four

choices that sample your knowledge of the solution. If you make the wrong choice, you are given a hint or advice and encouraged to try the same problem again.

Problem 2.1: Examine the following DFA [shown on-line by the Gradiance system]. Identify in the list below the string that this automaton accepts.

Problem 2.2: The finite automaton below [shown on-line by the Gradiance system] accepts no word of length zero, no word of length one, and only two words of length two (01 and 10). There is a fairly simple recurrence equation for the number $N(k)$ of words of length k that this automaton accepts. Discover this recurrence and demonstrate your understanding by identifying the correct value of $N(k)$ for some particular k. Note: the recurrence does not have an easy-to-use closed form, so you will have to compute the first few values by hand. You do not have to compute $N(k)$ for any k greater than 14.

Problem 2.3: Here is the transition function of a simple, deterministic automaton with start state A and accepting state B:

	0	**1**
A	A	B
B	B	A

We want to show that this automaton accepts exactly those strings with an odd number of 1's, or more formally:

$$\delta(A, w) = B \text{ if and only if } w \text{ has an odd number of 1's.}$$

Here, δ is the extended transition function of the automaton; that is, $\delta(A, w)$ is the state that the automaton is in after processing input string w The proof of the statement above is an induction on the length of w. Below, we give the proof with reasons missing. You must give a reason for each step, and then demonstrate your understanding of the proof by classifying your reasons into the following three categories:

A) Use of the inductive hypothesis.

B) Reasoning about properties of deterministic finite automata, e.g., that if string $s = yz$, then $\delta(q, s) = \delta(\delta(q, y), z)$.

C) Reasoning about properties of binary strings (strings of 0's and 1's), e.g., that every string is longer than any of its proper substrings.

Basis ($|w| = 0$):

1. $w = \epsilon$ because:
2. $\delta(A, \epsilon) = A$ because:

3. ϵ has an even number of 0's because:

Induction ($|w| = n > 0$)

4. There are two cases: (a) when $w = x1$ and (b) when $w = x0$ because:

Case (a):

5. In case (a), w has an odd number of 1's if and only if x has an even number of 1's because:
6. In case (a), $\delta(A, x) = A$ if and only if w has an odd number of 1's because:
7. In case (a), $\delta(A, w) = B$ if and only if w has an odd number of 1's because:

Case (b):

8. In case (b), w has an odd number of 1's if and only if x has an odd number of 1' because:
9. In case (b), $\delta(A, x) = B$ if and only if w has an odd number of 1's because:
10. In case (b), $\delta(A, w) = B$ if and only if w has an odd number of 1's because:

Problem 2.4: Convert the following nondeterministic finite automaton [shown on-line by the Gradiance system] to a DFA, including the dead state, if necessary. Which of the following sets of NFA states is **not** a state of the DFA that is accessible from the start state of the DFA?

Problem 2.5: The following nondeterministic finite automaton [shown on-line by the Gradiance system] accepts which of the following strings?

Problem 2.6: Here is a nondeterministic finite automaton with epsilon-transitions [shown on-line by the Gradiance system]. Suppose we use the extended subset construction from Section 2.5.5 to convert this epsilon-NFA to a deterministic finite automaton with a dead state, with all transitions defined, and with no state that is inaccessible from the start state. Which of the following would be a transition of the DFA?

Problem 2.7: Here is an epsilon-NFA [shown on-line by the Gradiance system]. Suppose we construct an equivalent DFA by the construction of Section 2.5.5. That is, start with the epsilon-closure of the start state A. For each set of states S we construct (which becomes one state of the DFA), look at the transitions from this set of states on input symbol 0. See where those transitions lead, and take the union of the epsilon-closures of all the states reached on 0. This set of states becomes a state of the DFA. Do the same for the transitions out of S on input 1. When we have found all the sets of epsilon-NFA states that are constructed in this way, we have the DFA and its transitions. Carry out this construction of a DFA, and identify one of the states of this DFA (as a subset of the epsilon-NFA's states) from the list below.

Problem 2.8: Identify which automata [in a set of diagrams shown on-line by the Gradiance system] define the same language and provide the correct counterexample if they don't. Choose the correct statement from the list below.

Problem 2.9: Examine the following DFA [shown on-line by the Gradiance system]. This DFA accepts a certain language L. In this problem we shall consider certain other languages that are defined by their tails, that is, languages of the form $(0+1)*w$, for some particular string w of 0's and 1's. Call this language $L(w)$. Depending on w, $L(w)$ may be contained in L, disjoint from L, or neither contained nor disjoint from L (i.e., some strings of the form xw are in L and others are not). Your problem is to find a way to classify w into one of these three cases. Then, use your knowledge to classify the following languages:

1. $L(1111001)$, i.e., the language of regular expression $(0+1)*1111001$.
2. $L(11011)$, i.e., the language of regular expression $(0+1)*11011$.
3. $L(110101)$, i.e., the language of regular expression $(0+1)*110101$.
4. $L(00011101)$, i.e., the language of regular expression $(0+1)*00011101$.

Problem 2.10: Here is a nondeterministic finite automaton [shown on-line by the Gradiance system]. Convert this NFA to a DFA, using the "lazy" version of the subset construction described in Section 2.3.5, so only the accessible states are constructed. Which of the following sets of NFA states becomes a state of the DFA?

Problem 2.11: Here is a nondeterministic finite automaton [shown on-line by the Gradiance system]. Some input strings lead to more than one state. Find, in the list below, a string that leads from the start state A to three different states (possibly including A).

2.8 References for Chapter 2

The formal study of finite-state systems is generally regarded as originating with [2]. However, this work was based on a "neural nets" model of computing, rather than the finite automaton we know today. The conventional DFA was independently proposed, in several similar variations, by [1], [3], and [4]. The nondeterministic finite automaton and the subset construction are from [5].

1. D. A. Huffman, "The synthesis of sequential switching circuits," *J. Franklin Inst.* **257**:3-4 (1954), pp. 161–190 and 275–303.
2. W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervious activity," *Bull. Math. Biophysics* **5** (1943), pp. 115–133.
3. G. H. Mealy, "A method for synthesizing sequential circuits," *Bell System Technical Journal* **34**:5 (1955), pp. 1045–1079.

is an associative operator it does not matter in what order we group consecutive concatenations, although if there is a choice to be made, you should group them from the left. For instance, **012** is grouped (**01**)**2**.

3. Finally, all unions (+ operators) are grouped with their operands. Since union is also associative, it again matters little in which order consecutive unions are grouped, but we shall assume grouping from the left.

Of course, sometimes we do not want the grouping in a regular expression to be as required by the precedence of the operators. If so, we are free to use parentheses to group operands exactly as we choose. In addition, there is never anything wrong with putting parentheses around operands that you want to group, even if the desired grouping is implied by the rules of precedence.

Example 3.3: The expression $\mathbf{01}^* + \mathbf{1}$ is grouped $\big(\mathbf{0}(\mathbf{1}^*)\big) + \mathbf{1}$. The star operator is grouped first. Since the symbol **1** immediately to its left is a legal regular expression, that alone is the operand of the star. Next, we group the concatenation between **0** and $(\mathbf{1}^*)$, giving us the expression $\big(\mathbf{0}(\mathbf{1}^*)\big)$. Finally, the union operator connects the latter expression and the expression to its right, which is **1**.

Notice that the language of the given expression, grouped according to the precedence rules, is the string 1 plus all strings consisting of a 0 followed by any number of 1's (including none). Had we chosen to group the dot before the star, we could have used parentheses, as $(\mathbf{01})^* + \mathbf{1}$. The language of this expression is the string 1 and all strings that repeat 01, zero or more times. Had we wished to group the union first, we could have added parentheses around the union to make the expression $\mathbf{0}(\mathbf{1}^* + \mathbf{1})$. That expression's language is the set of strings that begin with 0 and have any number of 1's following. □

3.1.4 Exercises for Section 3.1

Exercise 3.1.1: Write regular expressions for the following languages:

***** a) The set of strings over alphabet $\{a, b, c\}$ containing at least one a and at least one b.

b) The set of strings of 0's and 1's whose tenth symbol from the right end is 1.

c) The set of strings of 0's and 1's with at most one pair of consecutive 1's.

! Exercise 3.1.2: Write regular expressions for the following languages:

***** a) The set of all strings of 0's and 1's such that every pair of adjacent 0's appears before any pair of adjacent 1's.

b) The set of strings of 0's and 1's whose number of 0's is divisible by five.

!! **Exercise 3.1.3:** Write regular expressions for the following languages:

a) The set of all strings of 0's and 1's not containing 101 as a substring.

b) The set of all strings with an equal number of 0's and 1's, such that no prefix has two more 0's than 1's, nor two more 1's than 0's.

c) The set of strings of 0's and 1's whose number of 0's is divisible by five and whose number of 1's is even.

! **Exercise 3.1.4:** Give English descriptions of the languages of the following regular expressions:

* a) $(\mathbf{1} + \epsilon)(\mathbf{00}^*\mathbf{1})^*\mathbf{0}^*$.

b) $(\mathbf{0}^*\mathbf{1}^*)^*\mathbf{000}(\mathbf{0} + \mathbf{1})^*$.

c) $(\mathbf{0} + \mathbf{10})^*\mathbf{1}^*$.

*! **Exercise 3.1.5:** In Example 3.1 we pointed out that $\emptyset$ is one of two languages whose closure is finite. What is the other?

3.2 Finite Automata and Regular Expressions

While the regular-expression approach to describing languages is fundamentally different from the finite-automaton approach, these two notations turn out to represent exactly the same set of languages, which we have termed the "regular languages." We have already shown that deterministic finite automata, and the two kinds of nondeterministic finite automata — with and without ϵ-transitions — accept the same class of languages. In order to show that the regular expressions define the same class, we must show that:

1. Every language defined by one of these automata is also defined by a regular expression. For this proof, we can assume the language is accepted by some DFA.

2. Every language defined by a regular expression is defined by one of these automata. For this part of the proof, the easiest is to show that there is an NFA with ϵ-transitions accepting the same language.

Figure 3.1 shows all the equivalences we have proved or will prove. An arc from class X to class Y means that we prove every language defined by class X is also defined by class Y. Since the graph is strongly connected (i.e., we can get from each of the four nodes to any other node) we see that all four classes are really the same.

must create a copy of the automaton of Fig. 3.18(a); we must not use the same copy that became part of Fig. 3.18(b). The complete automaton is shown in Fig. 3.18(c). Note that this ϵ-NFA, when ϵ-transitions are removed, looks just like the much simpler automaton of Fig. 3.15 that also accepts the strings that have a 1 in their next-to-last position. $\Box$

3.2.4 Exercises for Section 3.2

Exercise 3.2.1: Here is a transition table for a DFA:

	0	1
$\rightarrow q_1$	q_2	q_1
q_2	q_3	q_1
$*q_3$	q_3	q_2

***** a) Give all the regular expressions $R_{ij}^{(0)}$. *Note*: Think of state q_i as if it were the state with integer number i.

***** b) Give all the regular expressions $R_{ij}^{(1)}$. Try to simplify the expressions as much as possible.

c) Give all the regular expressions $R_{ij}^{(2)}$. Try to simplify the expressions as much as possible.

d) Give a regular expression for the language of the automaton.

***** e) Construct the transition diagram for the DFA and give a regular expression for its language by eliminating state q_2.

Exercise 3.2.2: Repeat Exercise 3.2.1 for the following DFA:

	0	1
$\rightarrow q_1$	q_2	q_3
q_2	q_1	q_3
$*q_3$	q_2	q_1

Note that solutions to parts (a), (b) and (e) are *not* available for this exercise.

Exercise 3.2.3: Convert the following DFA to a regular expression, using the state-elimination technique of Section 3.2.2.

	0	1
$\rightarrow *p$	s	p
q	p	s
r	r	q
s	q	r

Exercise 3.2.4: Convert the following regular expressions to NFA's with ϵ-transitions.

* a) $\mathbf{01}^*$.

b) $(\mathbf{0}+\mathbf{1})\mathbf{01}$.

c) $\mathbf{00}(\mathbf{0}+\mathbf{1})^*$.

Exercise 3.2.5: Eliminate ϵ-transitions from your ϵ-NFA's of Exercise 3.2.4. A solution to part (a) appears in the book's Web pages.

! **Exercise 3.2.6:** Let $A = (Q, \Sigma, \delta, q_0, \{q_f\})$ be an ϵ-NFA such that there are no transitions into q_0 and no transitions out of q_f. Describe the language accepted by each of the following modifications of A, in terms of $L = L(A)$:

* a) The automaton constructed from A by adding an ϵ-transition from q_f to q_0.

* b) The automaton constructed from A by adding an ϵ-transition from q_0 to every state reachable from q_0 (along a path whose labels may include symbols of Σ as well as ϵ).

c) The automaton constructed from A by adding an ϵ-transition to q_f from every state that can reach q_f along some path.

d) The automaton constructed from A by doing both (b) and (c).

!! **Exercise 3.2.7:** There are some simplifications to the constructions of Theorem 3.7, where we converted a regular expression to an ϵ-NFA. Here are three:

1. For the union operator, instead of creating new start and accepting states, merge the two start states into one state with all the transitions of both start states. Likewise, merge the two accepting states, having all transitions to either go to the merged state instead.

2. For the concatenation operator, merge the accepting state of the first automaton with the start state of the second.

3. For the closure operator, simply add ϵ-transitions from the accepting state to the start state and vice-versa.

Each of these simplifications, by themselves, still yield a correct construction; that is, the resulting ϵ-NFA for any regular expression accepts the language of the expression. Which subsets of changes (1), (2), and (3) may be made to the construction together, while still yielding a correct automaton for every regular expression?

*!! **Exercise 3.2.8:** Give an algorithm that takes a DFA A and computes the number of strings of length n (for some given n, not related to the number of states of A) accepted by A. Your algorithm should be polynomial in both n and the number of states of A. *Hint*: Use the technique suggested by the construction of Theorem 3.4.

Exercise 3.4.4: At the beginning of Section 3.4.6, we gave part of a proof that $(L^*M^*)^* = (L+M)^*$. Complete the proof by showing that strings in $(L^*M^*)^*$ are also in $(L+M)^*$.

! **Exercise 3.4.5:** Complete the proof of Theorem 3.13 by handling the cases where regular expression E is of the form FG or of the form F^*.

3.5 Summary of Chapter 3

- *Regular Expressions*: This algebraic notation describes exactly the same languages as finite automata: the regular languages. The regular-expression operators are union, concatenation (or "dot"), and closure (or "star").
- *Regular Expressions in Practice*: Systems such as UNIX and various of its commands use an extended regular-expression language that provides shorthands for many common expressions. Character classes allow the easy expression of sets of symbols, while operators such as one-or-more-of and at-most-one-of augment the usual regular-expression operators.
- *Equivalence of Regular Expressions and Finite Automata*: We can convert a DFA to a regular expression by an inductive construction in which expressions for the labels of paths allowed to pass through increasingly larger sets of states are constructed. Alternatively, we can use a state-elimination procedure to build the regular expression for a DFA. In the other direction, we can construct recursively an ϵ-NFA from regular expressions, and then convert the ϵ-NFA to a DFA, if we wish.
- *The Algebra of Regular Expressions*: Regular expressions obey many of the algebraic laws of arithmetic, although there are differences. Union and concatenation are associative, but only union is commutative. Concatenation distributes over union. Union is idempotent.
- *Testing Algebraic Identities*: We can tell whether a regular-expression equivalence involving variables as arguments is true by replacing the variables by distinct constants and testing whether the resulting languages are the same.

3.6 Gradiance Problems for Chapter 3

The following is a sample of problems that are available on-line through the Gradiance system at `www.gradiance.com/pearson`. Each of these problems is worked like conventional homework. The Gradiance system gives you four choices that sample your knowledge of the solution. If you make the wrong choice, you are given a hint or advice and encouraged to try the same problem again.

Problem 3.1: Here is a finite automaton [shown on-line by the Gradiance system]. Which of the following regular expressions defines the same language as the finite automaton? Hint: each of the correct choices uses component expressions. Some of these components are:

1. The ways to get from A to D without going through D.
2. The ways to get from D to itself, without going through D.
3. The ways to get from A to itself, without going through A.

It helps to write down these expressions first, and then look for an expression that defines all the paths from A to D.

Problem 3.2: When we convert an automaton to a regular expression, we need to build expressions for the labels along paths from one state to another state that do not go through certain other states. Below is a nondeterministic finite automaton with three states [shown on-line by the Gradiance system]. For each of the six orders of the three states, find regular expressions that give the set of labels along all paths from the first state to the second state that never go through the third state. Then identify one of these expressions from the list of choices below.

Problem 3.3: Identify from the list below the regular expression that generates all and only the strings over alphabet $\{0,1\}$ that end in 1.

Problem 3.4: Apply the construction in Fig. 3.16 and Fig. 3.17 to convert the regular expression $(0+1)^*(0+\epsilon)$ to an epsilon-NFA. Then, identify the true statement about your epsilon-NFA from the list below.

Problem 3.5: Consider the following identities for regular expressions; some are false and some are true. You are asked to decide which and in case it is false to provide the correct counterexample.

a) $R(S+T) = RS + RT$

b) $(R^*)^* = R^*$

c) $(R^*S^*)^* = (R+S)^*$

d) $(R+S)^* = R^* + S^*$

e) $S(RS+S)^*R = RR^*S(RR^*S)^*$

f) $(RS+R)^*R = R(SR+R)^*$

Problem 3.6: In this question you are asked to consider the truth or falsehood of six equivalences for regular expressions. If the equivalence is true, you must also identify the law from which it follows. In each case the statement $R = S$ is conventional shorthand for "$L(R) = L(S)$." The six proposed equivalences are:

1. $0^*1^* = 1^*0^*$
2. $01\emptyset = \emptyset$
3. $\epsilon 01 = 01$
4. $(0^* + 1^*)0 = 0^*0 + 1^*0$
5. $(0^*1)0^* = 0^*(10^*)$
6. $01 + 01 = 01$

Identify the correct statement from the list below.

Problem 3.7: Which of the following strings is **not** in the Kleene closure of the language $\{011, 10, 110\}$?

Problem 3.8: Here are seven regular expressions [shown on-line by the Gradiance system]. Determine the language of each of these expressions. Then, find in the list below a pair of equivalent expressions.

Problem 3.9: Converting a DFA such as the following [shown on-line by the Gradiance system]. to a regular expression requires us to develop regular expressions for limited sets of paths — those that take the automaton from one particular state to another particular state, without passing through some set of states. For the automaton above, determine the languages for the following limitations:

1. L_{AA} = the set of path labels that go from A to A without passing through C or D.
2. L_{AB} = the set of path labels that go from A to B without passing through C or D.
3. L_{BA} = the set of path labels that go from B to A without passing through C or D.
4. L_{BB} = the set of path labels that go from B to B without passing through C or D.

Then, identify a correct regular expression from the list below.

3.7 References for Chapter 3

The idea of regular expressions and the proof of their equivalence to finite automata is the work of S. C. Kleene [3]. However, the construction of an ϵ-NFA from a regular expression, as presented here, is the "McNaughton-Yamada construction," from [4]. The test for regular-expression identities by treating variables as constants was written down by J. Gischer [2]. Although thought to