Find a simple regular expression for the regular language recognized by this NFA. *Hint:* Transform the NFA into a DFA, and then find the minimum-state DFA.

Challenge

12. What can you say about the regular language accepted by a DFA in which all states are final?

11.4 Regular Language Topics

We've already seen characterizations of regular languages by regular expressions, languages accepted by DFAs, and languages accepted by NFAs. In this section we'll introduce still another characterization of regular languages in terms of certain restricted grammars. We'll discuss some properties of regular languages that can be used to find languages that are not regular.

Regular Grammars

A regular language can be described by a special kind of grammar in which the productions take a certain form. A grammar is called a *regular grammar* if each production takes one of the following forms, where the capital letters are nonterminals and w is a nonempty string of terminals:

$$\begin{aligned} &S \rightarrow \Lambda, \\ &S \rightarrow w, \\ &S \rightarrow T, \\ &S \rightarrow wT. \end{aligned}$$

The thing to keep in mind here is that only one nonterminal can appear on the right side of a production, and it must appear at the right end of the right side. For example, the productions $A \rightarrow aBc$ and $S \rightarrow TU$ are are not part of a regular grammar. But the production $A \rightarrow abcA$ is OK.

The most important aspect of grammar writing is knowledge of the language under discussion. We should also remember that grammars are not unique. So we shouldn't be surprised when two people come up with two different grammars for the same language.

To start things off, we'll look at a few regular grammars for some simple regular languages. Each line of the following list describes a regular language in terms of a regular expression and a regular grammar. As you look through the following list, cover up the grammar column with your hand and try to discover your own version of a regular grammar for the regular language of each regular expression.

Regular Expression	*Regular Grammar*
a^*	$S \rightarrow \Lambda \mid aS$
$(a + b)^*$	$S \rightarrow \Lambda \mid aS \mid bS$
$a^* + b^*$	$S \rightarrow \Lambda \mid A \mid B$ $A \rightarrow a \mid aA$ $B \rightarrow b \mid bB$
a^*b	$S \rightarrow b \mid aS$
ba^*	$S \rightarrow bA$ $A \rightarrow \Lambda \mid aA$
$(ab)^*$	$S \rightarrow \Lambda \mid abS$

The last three examples in the preceding list involve products of languages. Most problems occur in trying to construct a regular grammar for a language that is the product of languages. Let's look at an example to see whether we can get some insight into constructing such grammars.

Suppose we want to construct a regular grammar for the language of the regular expression a^*bc^*. First we observe that the strings of a^*bc^* start with either the letter a or the letter b. We can represent this property by writing down the following two productions, where S is the start symbol:

$$S \rightarrow a\ S \mid b\ C.$$

These productions allow us to derive strings of the form bC, abC, $aabC$, and so on. Now all we need is a definition for C to derive the language of c^*. The following two productions do the job:

$$C \rightarrow \Lambda \mid c\ C.$$

Therefore a regular grammar for a^*bc^* can be written as follows:

$$\begin{aligned} S &\rightarrow a\ S \mid b\ C \\ C &\rightarrow \Lambda \mid c\ C. \end{aligned}$$

EXAMPLE 1 Sample Regular Grammars

We'll consider some regular languages, all of which consist of strings of *a*'s followed by strings of *b*'s. The largest language of this form is the language $\{a^m b^n \mid m, n \in \mathbb{N}\}$, which is represented by the regular expression a^*b^*. A regular grammar for this language can be written as follows:

$$S \to \Lambda \mid aS \mid B$$
$$B \to b \mid bB.$$

Let's look at four sublanguages of $\{a^m b^n \mid m, n \in \mathbb{N}\}$ that are defined by whether each string contains occurrences of *a* or *b*. The following list shows each language together with a regular expression and a regular grammar.

Language	*Expression*	*Regular Grammar*
$\{a^m b^n \mid m \geq 0 \text{ and } n > 0\}$	a^*bb^*	$S \to aS \mid B$ $B \to b \mid bB.$
$\{a^m b^n \mid m > 0 \text{ and } n \geq 0\}$	aa^*b^*	$S \to aA$ $A \to aA \mid B$ $B \to \Lambda \mid bB.$
$\{a^m b^n \mid m > 0 \text{ and } n > 0\}$	aa^*bb^*	$S \to a\,A$ $A \to aA \mid B$ $B \to b \mid bB.$
$\{a^m b^n \mid m > 0 \text{ or } n > 0\}$	$aa^*b^* + a^*bb^*$	$S \to aA \mid bB$ $A \to \Lambda \mid aA \mid B$ $B \to \Lambda \mid b\,B.$ ♦

Any regular language has a regular grammar; conversely, any regular grammar generates a regular language. To see this, we'll give two algorithms: one to transform an NFA to a regular grammar and the other to transform a regular grammar to an NFA, where the language accepted by the NFA is identical to the language generated by the regular grammar.

NFA to Regular Grammar (11.11)

Perform the following steps to construct a regular grammar that generates the language of a given NFA:

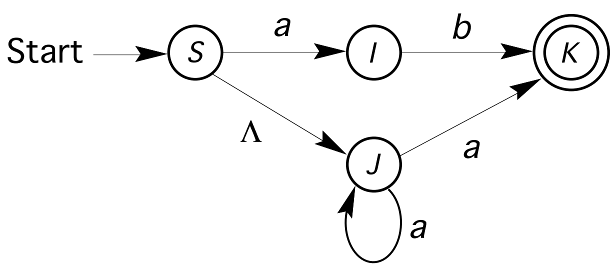
1. Rename the states to a set of uppercase letters.
2. The start symbol is the NFA's start state.
3. For each state transition from I to J labeled with a, create the production $I \to aJ$.

4. For each state transition from I to J labeled with Λ, create the production $I \to J$.
5. For each final state K, create the production $K \to \Lambda$.

It's easy to see that the language of the NFA and the language of the constructed grammar are the same. Just notice that each state transition in the NFA corresponds exactly with a production in the grammar so that the acceptance path in the NFA for some string corresponds to a derivation by the grammar for the same string. Let's do an example.

EXAMPLE 2 From NFA to Regular Grammar

Let's see how (11.11) transforms the following NFA into a regular grammar:



The algorithm takes this NFA and constructs the following regular grammar with start symbol S:

$$\begin{aligned} &S \to aI \mid J \\ &I \to bK \\ &J \to aJ \mid aK \\ &K \to \Lambda. \end{aligned}$$

For example, to accept the string aa, the NFA follows the path S, J, J, K with edges labeled Λ, a, a, respectively. The grammar derives this string with the following sequence of productions:

$$S \to J, \quad J \to aJ, \quad J \to aK, \quad K \to \Lambda. \quad \blacklozenge$$

Now let's look at the converse problem of constructing an NFA from a regular grammar. For the opposite transformation we'll first take a regular grammar and rewrite it so that all the productions have one of two forms $S \to x$ or $S \to xT$, where x is either Λ or a single letter. Let's see how to do this so that we don't lose any generality. For example, if we have a production like

$$A \to bcB,$$

we can replace it by the following two productions, where C is a new nonterminal:

$$A \to bC \quad \text{and} \quad C \to cB.$$

Now let's look at an algorithm that does the job of transforming a regular grammar into an NFA.

Regular Grammar to NFA (11.12)

Perform the following steps to construct an NFA that accepts the language of a given regular grammar:

1. If necessary, transform the grammar so that all productions have the form $A \to x$ or $A \to xB$, where x is either a single letter or Λ.
2. The start state of the NFA is the grammar's start symbol.
3. For each production $I \to aJ$, construct a state transition from I to J labeled with the letter a.
4. For each production $I \to J$, construct a state transition from I to J labeled with Λ.
5. If there are productions of the form $I \to a$ for some letter a, then create a single new state symbol F. For each production $I \to a$, construct a state transition from I to F labeled with a.
6. The final states of the NFA are F together with all I for which there is a production $I \to \Lambda$.

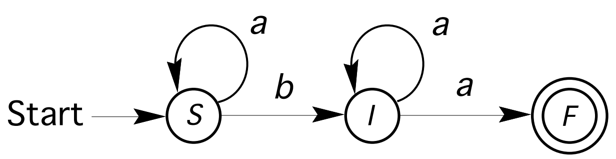
It's easy to see that the language of the NFA is the same as the language of the given regular grammar because the productions used in the derivation of any string correspond exactly with the state transitions on the path of acceptance for the string. Here's an example.

EXAMPLE 3 From Regular Grammar to NFA

Let's use (11.12) to transform the following regular grammar into an NFA:

$$\begin{aligned} S &\to aS \mid bI \\ I &\to a \mid aI. \end{aligned}$$

Since there is a production $I \to a$, we need to introduce a new state F, which then gives us the following NFA:



♦

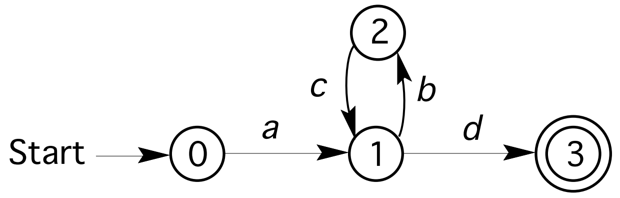
Properties of Regular Languages

We need to face the fact that not all languages are regular. To see this, let's look at a classic example. Suppose we want to find a DFA or NFA to recognize the following language.

$$\{a^n b^n \mid n \geq 0\}.$$

After a few attempts at trying to find a DFA or an NFA or a regular expression or a regular grammar, we might get the idea that it can't be done. But how can we be sure that a language is not regular? We can try to prove it. A proof usually proceeds by assuming that the language is regular and then trying to find a contradiction of some kind. For example, we might be able to find some property of regular languages that the given language doesn't satisfy. So let's look at a few properties of regular languages.

One useful property of regular languages comes from the observation that any DFA for an infinite regular language must contain a loop to recognize infinitely many strings. For example, suppose a DFA with four states accepts the 4-letter string *abcd.* To accept *abcd* the DFA must enter five states. For example, if the states of the DFA are numbered 0, 1, 2, and 3, where 0 is the start state and 3 is the final state, then there must be a path through the DFA starting at 0 and ending at 3 with edges labeled *a*, *b*, *c*, and *d.* For example, if the path is 0, 1, 2, 1, 3, then the following graph represents a portion of the DFA that contains the the path to accept *abcd*.



Of course, the loop 1, 2, 1 can be traveled any number of times. For example, the path 0, 1, 2, 1, 2, 1, 3 accepts the string *abcbcd*. So the DFA will accept the strings, *ad*, *abcd*, *abcbcd*, ..., $a(bc)^n d$, ... This is the property that we want to describe.

We'll generalize the idea illustrated in our little example. Suppose a DFA with *m* states recognizes an infinite regular language. If *s* is a string