! 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

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.