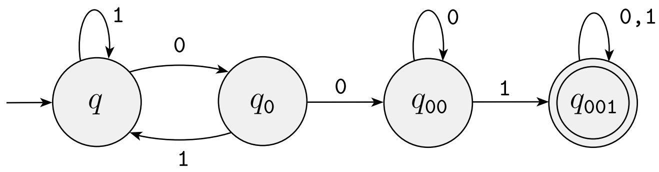


**FIGURE 1.22**
Accepts strings containing 001

## THE REGULAR OPERATIONS

In the preceding two sections, we introduced and defined finite automata and regular languages. We now begin to investigate their properties. Doing so will help develop a toolbox of techniques for designing automata to recognize particular languages. The toolbox also will include ways of proving that certain other languages are nonregular (i.e., beyond the capability of finite automata).

In arithmetic, the basic objects are numbers and the tools are operations for manipulating them, such as $+$ and $\times$. In the theory of computation, the objects are languages and the tools include operations specifically designed for manipulating them. We define three operations on languages, called the ***regular operations***, and use them to study properties of the regular languages.

**DEFINITION 1.23**

Let $A$ and $B$ be languages. We define the regular operations ***union***, ***concatenation***, and ***star*** as follows:

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

You are already familiar with the union operation. It simply takes all the strings in both $A$ and $B$ and lumps them together into one language.

The concatenation operation is a little trickier. It attaches a string from $A$ in front of a string from $B$ in all possible ways to get the strings in the new language.

The star operation is a bit different from the other two because it applies to a single language rather than to two different languages. That is, the star operation is a ***unary operation*** instead of a ***binary operation***. It works by attaching any number of strings in $A$ together to get a string in the new language. Because "any number" includes 0 as a possibility, the empty string $\varepsilon$ is always a member of $A^*$, no matter what $A$ is.

**EXAMPLE 1.24**

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

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

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

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

■

Let $\mathcal{N} = \{1, 2, 3, \ldots\}$ be the set of natural numbers. When we say that $\mathcal{N}$ is *closed under multiplication*, we mean that for any $x$ and $y$ in $\mathcal{N}$, the product $x \times y$ also is in $\mathcal{N}$. In contrast, $\mathcal{N}$ is not closed under division, as 1 and 2 are in $\mathcal{N}$ but $1/2$ is not. Generally speaking, a collection of objects is ***closed*** under some operation if applying that operation to members of the collection returns an object still in the collection. We show that the collection of regular languages is closed under all three of the regular operations. In Section 1.3, we show that these are useful tools for manipulating regular languages and understanding the power of finite automata. We begin with the union operation.

**THEOREM 1.25**

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

In other words, if $A_1$ and $A_2$ are regular languages, so is $A_1 \cup A_2$.

**PROOF IDEA** We have regular languages $A_1$ and $A_2$ and want to show that $A_1 \cup A_2$ also is regular. Because $A_1$ and $A_2$ are regular, we know that some finite automaton $M_1$ recognizes $A_1$ and some finite automaton $M_2$ recognizes $A_2$. To prove that $A_1 \cup A_2$ is regular, we demonstrate a finite automaton, call it $M$, that recognizes $A_1 \cup A_2$.

This is a proof by construction. We construct $M$ from $M_1$ and $M_2$. Machine $M$ must accept its input exactly when either $M_1$ or $M_2$ would accept it in order to recognize the union language. It works by *simulating* both $M_1$ and $M_2$ and accepting if either of the simulations accept.

How can we make machine $M$ simulate $M_1$ and $M_2$? Perhaps it first simulates $M_1$ on the input and then simulates $M_2$ on the input. But we must be careful here! Once the symbols of the input have been read and used to simulate $M_1$, we can't "rewind the input tape" to try the simulation on $M_2$. We need another approach.

Pretend that you are $M$. As the input symbols arrive one by one, you simulate both $M_1$ and $M_2$ simultaneously. That way, only one pass through the input is necessary. But can you keep track of both simulations with finite memory? All you need to remember is the state that each machine would be in if it had read up to this point in the input. Therefore, you need to remember a pair of states. How many possible pairs are there? If $M_1$ has $k_1$ states and $M_2$ has $k_2$ states, the number of pairs of states, one from $M_1$ and the other from $M_2$, is the product $k_1 \times k_2$. This product will be the number of states in $M$, one for each pair. The transitions of $M$ go from pair to pair, updating the current state for both $M_1$ and $M_2$. The accept states of $M$ are those pairs wherein either $M_1$ or $M_2$ is in an accept state.

### PROOF

Let $M_1$ recognize $A_1$, where $M_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$, and
$M_2$ recognize $A_2$, where $M_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$.

Construct $M$ to recognize $A_1 \cup A_2$, where $M = (Q, \Sigma, \delta, q_0, F)$.

1. $Q = \{(r_1, r_2) | \, r_1 \in Q_1 \text{ and } r_2 \in Q_2\}$.
   This set is the ***Cartesian product*** of sets $Q_1$ and $Q_2$ and is written $Q_1 \times Q_2$. It is the set of all pairs of states, the first from $Q_1$ and the second from $Q_2$.
2. $\Sigma$, the alphabet, is the same as in $M_1$ and $M_2$. In this theorem and in all subsequent similar theorems, we assume for simplicity that both $M_1$ and $M_2$ have the same input alphabet $\Sigma$. The theorem remains true if they have different alphabets, $\Sigma_1$ and $\Sigma_2$. We would then modify the proof to let $\Sigma = \Sigma_1 \cup \Sigma_2$.
3. $\delta$, the transition function, is defined as follows. For each $(r_1, r_2) \in Q$ and each $a \in \Sigma$, let
   $$\delta\big((r_1, r_2), a\big) = \big(\delta_1(r_1, a), \delta_2(r_2, a)\big).$$
   Hence $\delta$ gets a state of $M$ (which actually is a pair of states from $M_1$ and $M_2$), together with an input symbol, and returns $M$'s next state.
4. $q_0$ is the pair $(q_1, q_2)$.
5. $F$ is the set of pairs in which either member is an accept state of $M_1$ or $M_2$. We can write it as
   $$F = \{(r_1, r_2) | \, r_1 \in F_1 \text{ or } r_2 \in F_2\}.$$
   This expression is the same as $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$. (Note that it is *not* the same as $F = F_1 \times F_2$. What would that give us instead?[3])

[3] This expression would define $M$'s accept states to be those for which *both* members of the pair are accept states. In this case, $M$ would accept a string only if both $M_1$ *and* $M_2$ accept it, so the resulting language would be the *intersection* and not the union. In fact, this result proves that the class of regular languages is closed under intersection.

This concludes the construction of the finite automaton $M$ that recognizes the union of $A_1$ and $A_2$. This construction is fairly simple, and thus its correctness is evident from the strategy described in the proof idea. More complicated constructions require additional discussion to prove correctness. A formal correctness proof for a construction of this type usually proceeds by induction. For an example of a construction proved correct, see the proof of Theorem 1.54. Most of the constructions that you will encounter in this course are fairly simple and so do not require a formal correctness proof.

We have just shown that the union of two regular languages is regular, thereby proving that the class of regular languages is closed under the union operation. We now turn to the concatenation operation and attempt to show that the class of regular languages is closed under that operation, too.

**THEOREM 1.26**

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

In other words, if $A_1$ and $A_2$ are regular languages then so is $A_1 \circ A_2$.

To prove this theorem, let's try something along the lines of the proof of the union case. As before, we can start with finite automata $M_1$ and $M_2$ recognizing the regular languages $A_1$ and $A_2$. But now, instead of constructing automaton $M$ to accept its input if either $M_1$ or $M_2$ accept, it must accept if its input can be broken into two pieces, where $M_1$ accepts the first piece and $M_2$ accepts the second piece. The problem is that $M$ doesn't know where to break its input (i.e., where the first part ends and the second begins). To solve this problem, we introduce a new technique called nondeterminism.

# 1.2

## NONDETERMINISM

Nondeterminism is a useful concept that has had great impact on the theory of computation. So far in our discussion, every step of a computation follows in a unique way from the preceding step. When the machine is in a given state and reads the next input symbol, we know what the next state will be—it is determined. We call this ***deterministic*** computation. In a ***nondeterministic*** machine, several choices may exist for the next state at any point.

Nondeterminism is a generalization of determinism, so every deterministic finite automaton is automatically a nondeterministic finite automaton. As Figure 1.27 shows, nondeterministic finite automata may have additional features.