lost the 0's of $y$. Since $y \neq \epsilon$ we know that there can be no more than $n - 1$ 0's among $x$ and $z$. Thus, after assuming $L_{eq}$ is a regular language, we have proved a fact known to be false, that $xz$ is in $L_{eq}$. We have a proof by contradiction of the fact that $L_{eq}$ is not regular.

Example 4.3: Let us show that the language $L_{pr}$ consisting of all strings of 1's whose length is a prime is not a regular language. Suppose it were. Then there would be a constant $n$ satisfying the conditions of the pumping lemma. Consider some prime $p \geq n + 2$; there must be such a $p$, since there are an infinity of primes. Let $w = 1^p$.

By the pumping lemma, we can break $w = xyz$ such that $y \neq \epsilon$ and $|xy| \leq n$. Let $|y| = m$. Then $|xz| = p - m$. Now consider the string $xy^{p-m}z$, which must be in $L_{pr}$ by the pumping lemma, if $L_{pr}$ really is regular. However,

$$|xy^{p-m}z| = |xz| + (p-m)|y| = p - m + (p-m)m = (m+1)(p-m)$$

It looks like $|xy^{p-m}z|$ is not a prime, since it has two factors $m + 1$ and $p - m$. However, we must check that neither of these factors are 1, since then $(m+1)(p-m)$ might be a prime after all. But $m+1 > 1$, since $y \neq \epsilon$ tells us $m \geq 1$. Also, $p - m > 1$, since $p \geq n + 2$ was chosen, and $m \leq n$ since

$$m = |y| \leq |xy| \leq n$$

Thus, $p - m \geq 2$.

Again we have started by assuming the language in question was regular, and we derived a contradiction by showing that some string not in the language was required by the pumping lemma to be in the language. Thus, we conclude that $L_{pr}$ is not a regular language.

4.1.3 Exercises for Section 4.1

Exercise 4.1.1: Prove that the following are not regular languages.

a) $\{0^n1^n \mid n \geq 1\}$. This language, consisting of a string of 0’s followed by an equal-length string of 1’s, is the language $L_{01}$ we considered informally at the beginning of the section. Here, you should apply the pumping lemma in the proof.

b) The set of strings of balanced parentheses. These are the strings of characters “(" and ")" that can appear in a well-formed arithmetic expression.

* c) $\{0^n10^n \mid n \geq 1\}$.

d) $\{0^n1^m2^n \mid n$ and $m$ are arbitrary integers$\}$.

e) $\{0^n1^m \mid n \leq m\}$.

f) $\{0^n1^{2n} \mid n \geq 1\}$. 
Exercise 4.1.2: Prove that the following are not regular languages.

* a) \{0^n \mid n \text{ is a perfect square}\}.

b) \{0^n \mid n \text{ is a perfect cube}\}.

c) \{0^n \mid n \text{ is a power of 2}\}.

d) The set of strings of 0’s and 1’s whose length is a perfect square.

e) The set of strings of 0’s and 1’s that are of the form \(uw\), that is, some string repeated.

f) The set of strings of 0’s and 1’s that are of the form \(w^R\), that is, some string followed by its reverse. (See Section 4.2.2 for a formal definition of the reversal of a string.)

g) The set of strings of 0’s and 1’s of the form \(w\overline{w}\), where \(\overline{w}\) is formed from \(w\) by replacing all 0’s by 1’s, and vice-versa; e.g., \(\overline{100} = 011\), and 011100 is an example of a string in the language.

h) The set of strings of the form \(w1^n\), where \(w\) is a string of 0’s and 1’s of length \(n\).

Exercise 4.1.3: Prove that the following are not regular languages.

a) The set of strings of 0’s and 1’s, beginning with a 1, such that when interpreted as an integer, that integer is a prime.

b) The set of strings of the form \(0^i1^j\) such that the greatest common divisor of \(i\) and \(j\) is 1.

Exercise 4.1.4: When we try to apply the pumping lemma to a regular language, the “adversary wins,” and we cannot complete the proof. Show what goes wrong when we choose \(L\) to be one of the following languages:

* a) The empty set.

* b) \{00, 11\}.

* c) \((00 + 11)^*\).

d) \(01^*0^*1\).
state $q$ at least once. If we subtract from $L_3$ all the languages $L(E_q^*)$ for $q$ in $Q$, then we have the accepting computations of $A$ that visit all the states. Call this language $L_4$. By Theorem 4.10 we know $L_4$ is also regular.

Our final step is to construct $L$ from $L_4$ by getting rid of the state components. That is, $L = h(L_4)$. Now, $L$ is the set of strings in $\Sigma^*$ that are accepted by $A$ and that visit each state of $A$ at least once during their acceptance. Since regular languages are closed under homomorphisms, we conclude that $L$ is regular. □

### 4.2.5 Exercises for Section 4.2

**Exercise 4.2.1:** Suppose $h$ is the homomorphism from the alphabet $\{0, 1, 2\}$ to the alphabet $\{a, b\}$ defined by: $h(0) = a$, $h(1) = ab$, and $h(2) = ba$.

* a) What is $h(0120)$?
* b) What is $h(21120)$?
* c) If $L$ is the language $L(01^*2)$, what is $h(L)$?
* d) If $L$ is the language $L(0 + 12)$, what is $h(L)$?
* e) Suppose $L$ is the language $\{ababa\}$, that is, the language consisting of only the one string $ababa$. What is $h^{-1}(L)$?
* f) If $L$ is the language $L(a(ba)^*)$, what is $h^{-1}(L)$?

**Exercise 4.2.2:** If $L$ is a language, and $a$ is a symbol, then $L/a$, the quotient of $L$ and $a$, is the set of strings $w$ such that $wa$ is in $L$. For example, if $L = \{a, aab, baa\}$, then $L/a = \{e, ba\}$. Prove that if $L$ is regular, so is $L/a$. *Hint:* Start with a DFA for $L$ and consider the set of accepting states.

**Exercise 4.2.3:** If $L$ is a language, and $a$ is a symbol, then $a \setminus L$ is the set of strings $w$ such that $aw$ is in $L$. For example, if $L = \{a, aab, baa\}$, then $a \setminus L = \{e, ab\}$. Prove that if $L$ is regular, so is $a \setminus L$. *Hint:* Remember that the regular languages are closed under reversal and under the quotient operation of Exercise 4.2.2.

**Exercise 4.2.4:** Which of the following identities are true?

a) $(L/a)a = L$ (the left side represents the concatenation of the languages $L/a$ and $\{a\}$).

b) $a(a \setminus L) = L$ (again, concatenation with $\{a\}$, this time on the left, is intended).

c) $(La)/a = L$.

d) $a \setminus (aL) = L$. 
stages, each of which requires $O(s^2)$ time. First, we take the previous set of states and find their successors on input symbol $a$. Next, we compute the $\epsilon$-closure of this set of states. The initial set of states for the simulation is the $\epsilon$-closure of the initial state of the NFA.

Lastly, if the representation of $L$ is a regular expression of size $s$, we can convert to an $\epsilon$-NFA with at most $2s$ states, in $O(s)$ time. We then perform the simulation above, taking $O(ns^2)$ time on an input $w$ of length $n$.

### 4.3.4 Exercises for Section 4.3

* **Exercise 4.3.1:** Give an algorithm to tell whether a regular language $L$ is infinite. *Hint:* Use the pumping lemma to show that if the language contains any string whose length is above a certain lower limit, then the language must be infinite.

**Exercise 4.3.2:** Give an algorithm to tell whether a regular language $L$ contains at least 100 strings.

**Exercise 4.3.3:** Suppose $L$ is a regular language with alphabet $\Sigma$. Give an algorithm to tell whether $L = \Sigma^*$, i.e., all strings over its alphabet.

**Exercise 4.3.4:** Give an algorithm to tell whether two regular languages $L_1$ and $L_2$ have at least one string in common.

**Exercise 4.3.5:** Give an algorithm to tell, for two regular languages $L_1$ and $L_2$ over the same alphabet $\Sigma$, whether there is any string in $\Sigma^*$ that is in neither $L_1$ nor $L_2$.

### 4.4 Equivalence and Minimization of Automata

In contrast to the previous questions — emptiness and membership — whose algorithms were rather simple, the question of whether two descriptions of two regular languages actually define the same language involves considerable intellectual mechanics. In this section we discuss how to test whether two descriptors for regular languages are *equivalent*, in the sense that they define the same language. An important consequence of this test is that there is a way to minimize a DFA. That is, we can take any DFA and find an equivalent DFA that has the minimum number of states. In fact, this DFA is essentially unique: given any two minimum-state DFA’s that are equivalent, we can always find a way to rename the states so that the two DFA’s become the same.

#### 4.4.1 Testing Equivalence of States

We shall begin by asking a question about the states of a single DFA. Our goal is to understand when two distinct states $p$ and $q$ can be replaced by a single state that behaves like both $p$ and $q$. We say that states $p$ and $q$ are *equivalent* if:
successors under input symbol $a_1$ are indistinguishable. Then, the successors of those states on input $a_2$ are indistinguishable, and so on, until we conclude that $p$ and $q$ are indistinguishable.

Since $N$ has fewer states than $M$, there are two states of $M$ that are indistinguishable from the same state of $N$, and therefore indistinguishable from each other. But $M$ was designed so that all its states are distinguishable from each other. We have a contradiction, so the assumption that $N$ exists is wrong, and $M$ in fact has as few states as any equivalent DFA for $A$. Formally, we have proved:

**Theorem 4.26:** If $A$ is a DFA, and $M$ the DFA constructed from $A$ by the algorithm described in the statement of Theorem 4.24, then $M$ has as few states as any DFA equivalent to $A$. □

In fact we can say something even stronger than Theorem 4.26. There must be a one-to-one correspondence between the states of any other minimum-state $N$ and the DFA $M$. The reason is that we argued above how each state of $M$ must be equivalent to one state of $N$, and no state of $M$ can be equivalent to two states of $N$. We can similarly argue that no state of $N$ can be equivalent to two states of $M$, although each state of $N$ must be equivalent to one of $M$'s states. Thus, the minimum-state DFA equivalent to $A$ is unique except for a possible renaming of the states.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$→A$</td>
<td>$B$</td>
<td>$A$</td>
</tr>
<tr>
<td>$B$</td>
<td>$A$</td>
<td>$C$</td>
</tr>
<tr>
<td>$C$</td>
<td>$D$</td>
<td>$B$</td>
</tr>
<tr>
<td>$*$</td>
<td>$D$</td>
<td>$A$</td>
</tr>
<tr>
<td>$E$</td>
<td>$D$</td>
<td>$F$</td>
</tr>
<tr>
<td>$F$</td>
<td>$G$</td>
<td>$E$</td>
</tr>
<tr>
<td>$G$</td>
<td>$F$</td>
<td>$G$</td>
</tr>
<tr>
<td>$H$</td>
<td>$G$</td>
<td>$D$</td>
</tr>
</tbody>
</table>

Figure 4.14: A DFA to be minimized

### 4.4.5 Exercises for Section 4.4

* **Exercise 4.4.1:** In Fig. 4.14 is the transition table of a DFA.
  a) Draw the table of distinguishabilities for this automaton.
  b) Construct the minimum-state equivalent DFA.

**Exercise 4.4.2:** Repeat Exercise 4.4.1 for the DFA of Fig. 4.15.
Figure 4.15: Another DFA to minimize

**Exercise 4.4.3:** Suppose that \( p \) and \( q \) are distinguishable states of a given DFA \( A \) with \( n \) states. As a function of \( n \), what is the tightest upper bound on how long the shortest string that distinguishes \( p \) from \( q \) can be?

### 4.5 Summary of Chapter 4

- **The Pumping Lemma:** If a language is regular, then every sufficiently long string in the language has a nonempty substring that can be “pumped,” that is, repeated any number of times while the resulting strings are also in the language. This fact can be used to prove that many different languages are not regular.

- **Operations That Preserve the Property of Being a Regular Language:** There are many operations that, when applied to regular languages, yield a regular language as a result. Among these are union, concatenation, closure, intersection, complementation, difference, reversal, homomorphism (replacement of each symbol by an associated string), and inverse homomorphism.

- **Testing Emptiness of Regular Languages:** There is an algorithm that, given a representation of a regular language, such as an automaton or regular expression, tells whether or not the represented language is the empty set.

- **Testing Membership in a Regular Language:** There is an algorithm that, given a string and a representation of a regular language, tells whether or not the string is in the language.

- **Testing Distinguishability of States:** Two states of a DFA are distinguishable if there is an input string that takes exactly one of the two states to an accepting state. By starting with only the fact that pairs consisting