its initial state q_0. States q_0, q_1, and q_2 are all final, so any input string not containing three consecutive b's will be accepted. However, a run of three b's will drive M to state q_3, which is not final, and M will then remain in this state regardless of the symbols in the rest of the input string. State q_3 is said to be a *dead state*, and if M reaches state q_3 it is said to be *trapped*, since no further input can enable it to escape from this state.◇

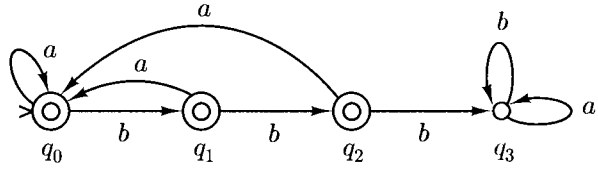


Figure 2-3

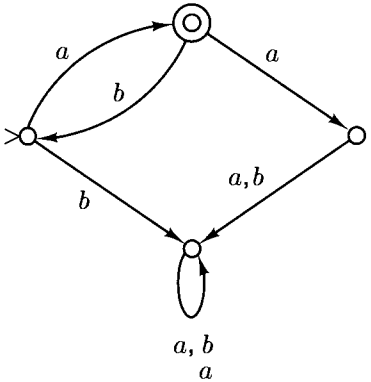
Problems for Section 2.1

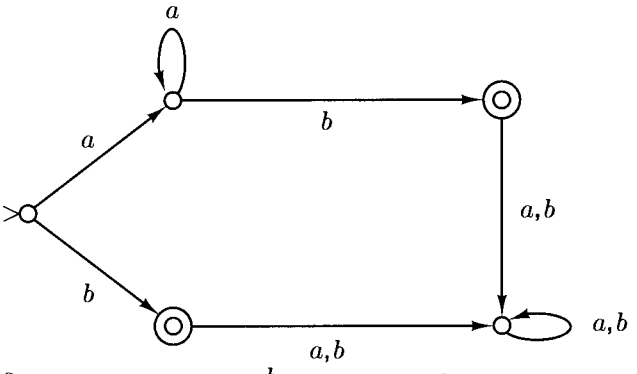
2.1.1. Let M be a deterministic finite automaton. Under exactly what circumstances is $e \in L(M)$? Prove your answer.

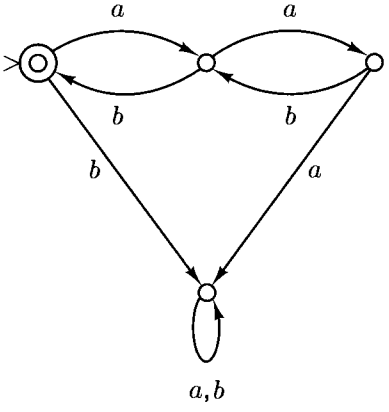
2.1.2. Describe informally the languages accepted by the deterministic finite automata shown in the next page.

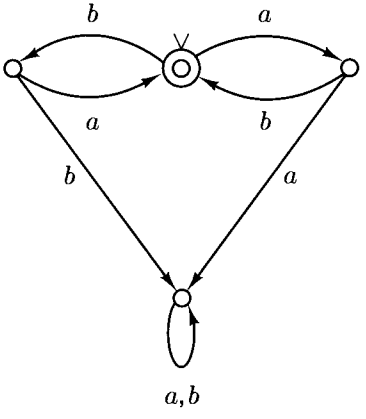
2.1.3. Construct deterministic finite automata accepting each of the following languages.
(a) $\{w \in \{a,b\}^* : \text{each } a \text{ in } w \text{ is immediately preceded by a } b\}$.
(b) $\{w \in \{a,b\}^* : w \text{ has } abab \text{ as a substring}\}$.
(c) $\{w \in \{a,b\}^* : w \text{ has neither } aa \text{ nor } bb \text{ as a substring}\}$.
(d) $\{w \in \{a,b\}^* : w \text{ has an odd number of } a\text{'s and an even number of } b\text{'s}\}$.
(e) $\{w \in \{a,b\}^* : w \text{ has both } ab \text{ and } ba \text{ as substrings}\}$.

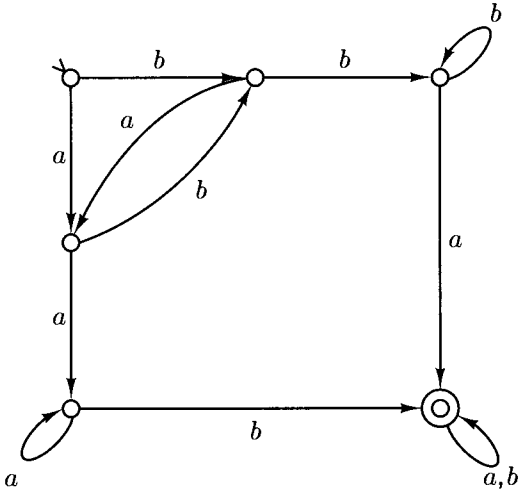
2.1.4. A **deterministic finite-state transducer** is a device much like a deterministic finite automaton, except that its purpose is not to accept strings or languages but to transform input strings into output strings. Informally, it starts in a designated initial state and moves from state to state, depending on the input, just as a deterministic finite automaton does. On each step, however, it emits (or writes onto an output tape) a string of zero or one or more symbols, depending on the current state and the input symbol. The state diagram for a deterministic finite-state transducer looks like that for a deterministic finite automaton, except that the label on an arrow looks like

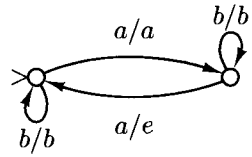

a
a
b
a,b
b
a, b


a
b
a
a,b
b
a,b
a,b


a
a
b
b
b
a
a,b


b
a
a
b
b
a
a,b


b
b
b
a
a
b
a
a
b
a
a,b



a/w, which means "if the input symbol is a, follow this arrow and emit output w". For example, the deterministic finite-state transducer over $\{a, b\}$ shown above transmits all b's in the input string but omits every other a.

(a) Draw state diagrams for deterministic finite-state transducers over $\{a, b\}$ that do the following.

(i) On input w, produce output a^n, where n is the number of occurrences of the substring ab in w.

(ii) On input w, produce output a^n, where n is the number of occurrences of the substring aba in w.

(iii) On input w, produce a string of length w whose ith symbol is an a if $i = 1$, or if $i > 1$ and the ith and $(i-1)$st symbols of w are different; otherwise, the ith symbol of the output is a b. For example, on input *aabba* the transducer should output *ababa*, and on input *aaaab* it should output *abbba*.

(b) Formally define

(i) a deterministic finite-state transducer;

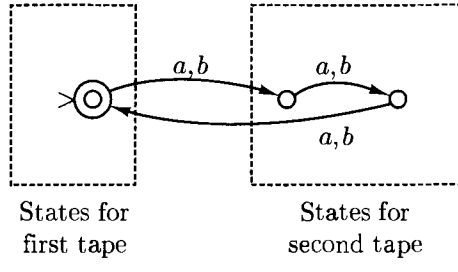
(ii) the notion of a configuration for such an automaton;

(iii) the yields in one step relation $\vdash$ between configurations;

(iv) the notion that such an automaton produces output u on input w;

(v) the notion that such an automaton computes a function.

2.1.5. A deterministic 2-tape finite automaton is a device like a deterministic finite automaton for accepting pairs of strings. Each state is in one of two sets; depending on which set the state is in, the transition function refers to the first or second tape. For example, the automaton shown below accepts all pairs of strings $(w_1, w_2) \in \{a, b\}^* \times \{a, b\}^*$ such that $|w_2| = 2|w_1|$.



(a) Draw state diagrams for deterministic 2-tape finite automata that accept each of the following.

(i) All pairs of strings (w_1, w_2) in $\{a,b\}^* \times \{a,b\}^*$ such that $|w_1| = |w_2|$, and $w_1(i) \neq w_2(i)$ for all i.

(ii) All pairs of strings (w_1, w_2) in $\{a,b\}^* \times \{a,b\}^*$ such that the length of w_2 is twice the number of a's in w_1 plus three times the number of b's in w_1.

(iii) $\{(a^n b, a^n b^m) : n, m \geq 0\}$.

(iv) $\{(a^n b, a^m b^n) : n, m \geq 0\}$.

(b) Formally define

(i) a deterministic 2-tape finite automaton;

(ii) the notion of a configuration for such an automaton;

(iii) the yields in one step relation $\vdash$ between configurations;

(iv) the notion that such an automaton accepts an ordered pair of strings;

(v) the notion that such an automaton accepts a set of ordered pairs of strings.

2.1.6. This problem refers to Problems 2.1.5 and 2.1.6. Show that if $f : \Sigma^* \mapsto \Sigma^*$ is a function that can be computed by a deterministic finite-state transducer, then $\{(w, f(w)) : w \in \Sigma^*\}$ is a set of pairs of strings accepted by some deterministic two-tape finite automaton.

2.1.7. We say that state q of a deterministic finite automaton $M = (K, \Sigma, \delta, q_0, F)$ is *reachable* if there exists $w \in \Sigma^*$ such that $(q_0, w) \vdash_M^* (q, e)$. Show that if we delete from M any nonreachable state, an automaton results that accepts the same language. Give an efficient algorithm for computing the set of all reachable states of a deterministic finite automaton.

2.2 NONDETERMINISTIC FINITE AUTOMATA

In this section we add a powerful and intriguing feature to finite automata. This feature is called **nondeterminism**, and is essentially the ability to change states in a way that is only partially determined by the current state and input symbol. That is, we shall now permit several possible "next states" for a given combination of current state and input symbol. The automaton, as it reads the input string, may choose at each step to go into any one of these legal next states; the choice is not determined by anything in our model, and is therefore said to be *nondeterministic*. On the other hand, the choice is not wholly unlimited either; only those next states that are legal from a given state with a given input symbol can be chosen.

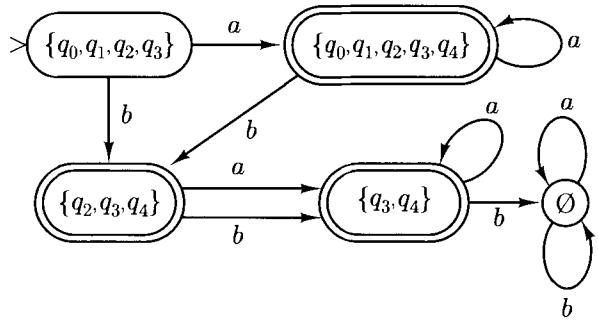


Figure 2-10

of F; so in the illustration, the three states $\{q_0, q_1, q_2, q_3, q_4\}$, $\{q_2, q_3, q_4\}$, and $\{q_3, q_4\}$ of M' are final.$\Diamond$

Example 2.2.5: Recall the $n+1$-state nondeterministic finite automaton in Example 2.2.2 with $\Sigma = \{a_1, \ldots, a_n\}$ for accepting the language $L = \{w \in \Sigma^* :$ there is a symbol $a_i \in \Sigma$ that does not occur in $w\}$. Intuitively, there is no way for a deterministic finite automaton to accept the same language with so few states.

Indeed, the construction is in this case exponential. The equivalent deterministic automaton M' has as initial state the set $s' = E(s) = \{s, q_1, q_2, \ldots, q_n\}$. Even in this case, M' has several irrelevant states —in fact, half of the states in 2^K are irrelevant. For example, state $\{s\}$ cannot be reached from s'; neither can any state that contains some q_i but not s. Alas, these are all the irrelevant states: as it turns out, all the remaining 2^n states of K' —that is to say, all states of the form $\{s\} \cup Q$ for some nonempty subset Q of $\{q_1, \ldots, q_n\}$— are reachable from s'.

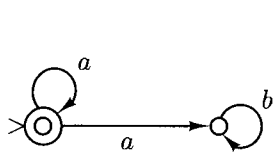
One might hope, of course, that other kinds of optimizations may reduce the number of states in M'. Section 2.5 contains a systematic study of such optimizations. Unfortunately, it follows from that analysis that in the present case of automaton M', the exponential number of states in M' *is* the absolute minimum possible.$\Diamond$

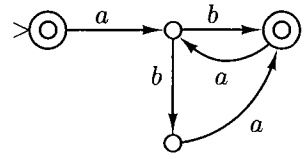
Problems for Section 2.2

2.2.1. (a) Which of the following strings are accepted by the nondeterministic finite automaton shown on the left below?

(i) a

(ii) aa





(iii) aab
(iv) e
(b) Repeat for the following strings and the nondeterministic finite automaton on the right above:
(i) e
(ii) ab
(iii) $abab$
(iv) aba
(v) $abaa$

2.2.2. Write regular expressions for the languages accepted by the nondeterministic finite automata of Problem 2.2.1.

2.2.3. Draw state diagrams for nondeterministic finite automata that accepts these languages.
(a) $(ab)^*(ba)^* \cup aa^*$
(b) $((ab \cup aab)^*a^*)^*$
(c) $((a^*b^*a^*)^*b)^*$
(d) $(ba \cup b)^* \cup (bb \cup a)*$

2.2.4. Some authors define a nondeterministic finite automaton to be a quintuple $(K, \Sigma, \Delta, S, F)$, where $K, \Sigma, \Delta,$, and F are as defined and S is a finite set of initial states, in the same way that F is a finite set of final states. The automaton may nondeterministically begin operating in any of these initial states.
(a) Show that the language $L \subseteq \{a_1, \ldots, a_n\}^*$ consisting of all strings that are missing at least one symbol (recall Example 2.2.2) would be accepted by such an automaton with n states $q_1, \ldots, q_n$, all of which are both final and initial, and the transition relation $\Delta = \{(q_i, a_j, q_i) : i \neq j\}$.
(b) Explain why this definition is not more general than ours in any significant way.

2.2.5. By what sequences of steps, other than the one presented in Example 2.2.1, can the nondeterministic finite automaton of Figure 2-7 accept the input $bababab$?

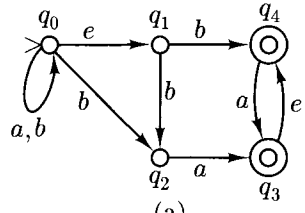
2.2.6. (a) Find a simple nondeterministic finite automaton accepting $(ab \cup aab \cup aba)^*$.

(b) Convert the nondeterministic finite automaton of Part (a) to a deterministic finite automaton by the method in Section 2.2.
(c) Try to understand how the machine constructed in Part (b) operates. Can you find an equivalent deterministic machine with fewer states?

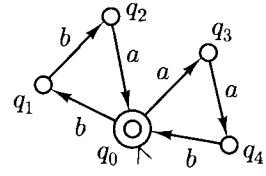
2.2.7. Repeat Problem 2.2.6 for the language $(a \cup b)^* aabab$.

2.2.8. Repeat Problem 2.2.6 for the language $(a \cup b)^* a(a \cup b)(a \cup b)(a \cup b)(a \cup b)$.

2.2.9. Construct deterministic finite automata equivalent to the nondeterministic automata shown below.



(a)



(b)

2.2.10. Describe exactly what happens when the construction of this section is applied to a finite automaton that is already deterministic.

2.3 FINITE AUTOMATA AND REGULAR EXPRESSIONS

The main result of the last section was that the class of languages accepted by finite automata remains the same even if a new and seemingly powerful feature —nondeterminism— is allowed. This suggests that the class of languages accepted by finite automata has a sort of *stability*: Two different approaches, one apparently more powerful than the other, end up defining the same class. In this section we shall prove another important characterization of this class of languages, further evidence of how remarkably stable it is: The class of languages accepted by finite automata, deterministic or nondeterministic, is the same as the class of *regular languages* —those that can be described by regular expressions, recall the discussion in Section 1.8.

We have introduced the class of regular languages as the closure of certain finite languages under the language operations of union, concatenation, and Kleene star. We must therefore start by proving similar closure properties of the class of languages accepted by finite automata:

Theorem 2.3.1: *The class of languages accepted by finite automata is closed under*

initial and final ones, and the generalized automaton has been reduced to a single transition from the initial state to the final state. We can now read the regular expression for M as the label of this transition:

$$R = R(4,5,5) = R(4,5,3) = a^*b(ba^*ba^*b \cup a)^*,$$

which is indeed $\{w \in \{a,b\}^* : w \text{ has } 3k+1 \text{ } b\text{'s for some } k \in \mathbf{N}\}$.$\diamondsuit$

Problems for Section 2.3

2.3.1. In part (d) of the proof of Theorem 2.3.1 why did we insist that M be deterministic? What happens if we interchange the final and nonfinal states of a nondeterministic finite automaton?

2.3.2. What goes wrong in the proof of Part (c) of Theorem 2.3.1 if we simply make s_1 final, and add arrows from all members of F_1 back to s_1 (without introducing a new starting state)?

2.3.3. Give a direct construction for the closure under intersection of the languages accepted by finite automata. (*Hint:* Consider an automaton whose set of states is the Cartesian product of the sets of states of the two original automata.) Which of the two constructions, the one given in the text or the one suggested in this problem, is more efficient when the two languages are given in terms of *nondeterministic* finite automata?

2.3.4. Using the construction in the proofs of Theorem 2.3.1, construct finite automata accepting these languages.
(a) $a^*(ab \cup ba \cup e)b^*$
(b) $((a \cup b)^*(e \cup c)^*)^*$
(c) $((ab)^* \cup (bc)^*)ab$

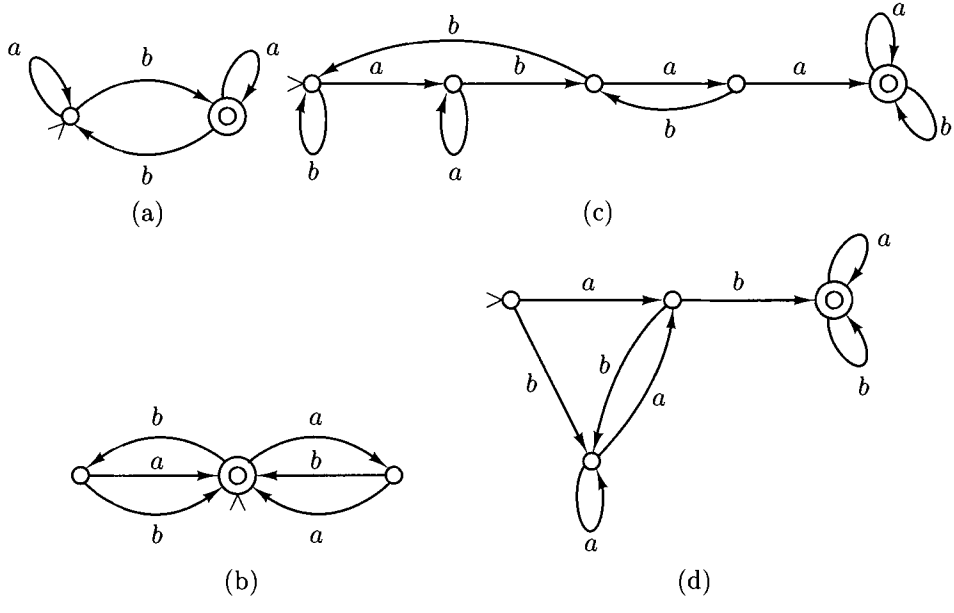
2.3.5. Construct a simple nondeterministic finite automaton to accept the language $(ab \cup aba)^*a$. Then apply to it the construction of Part (c) of the proof of Theorem 2.3.1 to obtain a nondeterministic finite automaton accepting $((ab \cup aba)^*a)^*$.

2.3.6. Let $L, L' \subseteq \Sigma^*$. Define the following languages.
1. $\text{Pref}(L) = \{w \in \Sigma^* : x = wy \text{ for some } x \in L, y \in \Sigma^*\}$ (the set of **prefixes** of L).
1. $\text{Suf}(L) = \{w \in \Sigma^* : x = yw \text{ for some } x \in L, y \in \Sigma^*\}$ (the set of **suffixes** of L).
3. $\text{Subseq}(L) = \{w_1 w_2 \ldots w_k : k \in \mathbf{N}, w_i \in \Sigma^* \text{ for } i = 1, \ldots, k, \text{ and there is a string } x = x_0 w_1 x_1 w_2 x_2 \ldots w_k x_k \in L\}$ (the set of **subsequences** of L).

4. $L/L' = \{w \in \Sigma^* : wx \in L \text{ for some } x \in L'\}$ (the **right quotient** of L by L').
5. $\text{Max}(L) = \{w \in L : \text{if } x \neq e \text{ then } wx \notin L\}$.
6. $L^R = \{w^R : w \in L\}$.

Show that if L is accepted by some finite automaton, then so is each of the following.

(a) $\text{Pref}(L)$
(b) $\text{Suf}(L)$
(c) $\text{Subseq}(L)$
(d) L/L', where L' is accepted by some finite automaton.
(e) L/L', where L' is *any* language.
(f) $\text{Max}(L)$
(g) L^R



2.3.7. Apply the construction in Example 2.3.2 to obtain regular expressions corresponding to each of the finite automata above. Simplify the resulting regular expressions as much as you can.

2.3.8. For any natural number $n \geq 1$ define the nondeterministic finite automaton $M_n = (K_n, \Sigma_n, \Delta_n, s_n, F_n)$ with $K_n = \{q_1, q_2, \ldots, q_n\}$, $s_n = q_1$, $F_n = \{q_1\}$, $\Sigma_n = \{a_{ij} : i, j = 1, \ldots, n\}$, and $\Delta_n = \{(i, a_{ij}, j) : i, j = 1, \ldots, n\}$.

(a) Describe $L(M_n)$ in English.

(b) Write a regular expression for $L(M_3)$.
(c) Write a regular expression for $L(M_5)$.
It is conjectured that, for all polynomials p there is an n such that no regular expression for $L(M_n)$ has length smaller than $p(n)$ symbols.

2.3.9. (a) By analogy with Problem 2.1.5, define a **nondeterministic 2-tape finite automaton**, and the notion that such an automaton accepts a particular set of ordered pairs of strings.
(b) Show that $\{(a^m b, a^n b^p) : n, m, p \geq 0, \text{ and } n = m \text{ or } n = p\}$ is accepted by some nondeterministic 2-tape finite automaton.
(c) We shall see (Problem 2.4.9) that nondeterministic 2-tape finite automata cannot always be converted into deterministic ones. This being the case, which of the constructions used in the proof of Theorem 2.3.1 can be extended to demonstrate closure properties of the following?
(i) The sets of pairs of strings accepted by nondeterministic 2-tape finite automata.
(ii) The sets of pairs of strings accepted by deterministic 2-tape finite automata.
Explain your answers.

2.3.10. A language L is **definite** if there is some k such that, for any string w, whether $w \in L$ depends only on the last k symbols of w.
(a) Rewrite this definition more formally.
(b) Show that every definite language is accepted by a finite automaton.
(c) Show that the class of definite languages is closed under union and complementation.
(d) Give an example of a definite language L such that L^* is not definite.
(e) Give an example of definite languages L_1, L_2 such that $L_1 L_2$ is not definite.

2.3.11. Let Σ and Δ be alphabets. Consider a function h from Σ to Δ^*. Extend h to a function from Σ^* to Δ^* as follows.

$$\begin{aligned} h(e) &= e. \\ h(w) &= h(w)h(\sigma) \text{ for any } w \in \Sigma^*, \sigma \in \Sigma. \end{aligned}$$

For example, if $\Sigma = \Delta = \{a, b\}$, $h(a) = ab$, and $h(b) = aab$, then

$$\begin{aligned} h(aab) &= h(aa)h(b) \\ &= h(a)h(a)h(b) \\ &= ababaab. \end{aligned}$$

Any function $h : \Sigma^* \mapsto \Delta^*$ defined in this way from a function $h : \Sigma \mapsto \Delta^*$ is called a **homomorphism**.

Let h be a homomorphism from Σ^* to Δ^*.

(a) Show that if $L \subseteq \Sigma^*$ is accepted by a finite automaton, then so is $h[L]$.

(b) Show that if L is accepted by a finite automaton, then so is $\{w \in \Sigma^* : h(w) \in L\}$. (*Hint:* Start from a deterministic finite automaton M accepting L, and construct one which, when it reads an input symbol a, tries to simulate what M would do on input $h(a)$.)

2.3.12. Deterministic finite-state transducers were introduced in Problem 2.1.4. Show that if L is accepted by a finite automaton, and f is computed by a deterministic finite-state transducer, then each of the following is true.

(a) $f[L]$ is accepted by a finite automaton.

(b) $f^{-1}[L]$ is accepted by a finite automaton.

2.4 LANGUAGES THAT ARE AND ARE NOT REGULAR

The results of the last two sections establish that the regular languages are closed under a variety of operations and that regular languages may be specified either by regular expressions or by deterministic or nondeterministic finite automata. These facts, used singly or in combinations, provide a variety of techniques for showing languages to be regular.

Example 2.4.1: Let $\Sigma = \{0, 1, \ldots, 9\}$ and let $L \subseteq \Sigma^*$ be the set of decimal representations for nonnegative integers (without redundant leading 0's) divisible by 2 or 3. For example, $0, 3, 6, 244 \in L$, but $1, 03, 00 \notin L$. Then L is regular. We break the proof into four parts.

Let L_1 be the set of decimal representations of nonnegative integers. Then it is easy to see that

$$L_1 = 0 \cup \{1, 2, \ldots, 9\}\Sigma^*,$$

which is regular since it is denoted by a regular expression.

Let L_2 be the set of decimal representations of nonnegative integers divisible by 2. Then L_2 is just the set of members of L, ending in 0, 2, 4, 6, or 8; that is,

$$L_2 = L_1 \cap \Sigma^*\{0, 2, 4, 6, 8\},$$

which is regular by Theorem 2.3.1(e).

Let L_3 be the set of decimal representations of nonnegative integers divisible by 3. Recall that a number is divisible by 3 if and only if the sum of its digits is divisible by 3. We construct a finite automaton that keeps track in its finite control of the sum modulo 3 of a string of digits. L_3 will then be the intersection

adversary must start by providing a number n; then you come up with a string w in the language that is longer than n; the adversary must now supply an appropriate decomposition of w into xyz; and, finally, you triumphantly point out i for which xy^iz is not in the language. If you have a strategy that always wins, no matter how brilliantly the adversary plays, then you have established that L is not regular.

It follows from this theorem that each of the two languages mentioned earlier in this section is not regular.

Example 2.4.2: The language $L = \{a^ib^i : i \geq 0\}$ is not regular, for if it were regular, Theorem 2.4.1 would apply for some integer n. Consider then the string $w = a^nb^n \in L$. By the theorem, it can be rewritten as $w = xyz$ such that $|xy| \leq n$ and $y \neq e$ —that is, $y = a^i$ for some $i > 0$. But then $xz = a^{n-i}b^n \notin L$, contradicting the theorem.$\Diamond$

Example 2.4.3: The language $L = \{a^n : n \text{ is prime}\}$ is not regular. For suppose it were, and let x, y, and z be as specified in Theorem 2.4.1. Then $x = a^p$, $y = a^q$, and and $z = a^r$, where $p, r \geq 0$ and $q > 0$. By the theorem, $xy^nz \in L$ for each $n \geq 0$; that is, $p + nq + r$ is prime for each $n \geq 0$. But this is impossible; for let $n = p + 2q + r + 2$; then $p + nq + r = (q + 1) \cdot (p + 2q + r)$, which is a product of two natural numbers, each greater than 1.$\Diamond$

Example 2.4.4: Sometimes it pays to use closure properties to show that a language is not regular. Take for example

$$L = \{w \in \{a, b\}^* : w \text{ has an equal number of } a\text{'s and } b\text{'s}\}.$$

L is not regular, because if L were indeed regular, then so would be $L \cap a^*b^*$ —by closure under intersection; recall Theorem 2.3.1(e). However, this latter language is precisely $\{a^nb^n : n \geq 0\}$, which we just showed is not regular.$\Diamond$

In fact, Theorem 2.4.1 can be strengthened substantially in several ways (see Problem 2.4.11 for one of them).

Problems for Section 2.4

2.4.1. An **arithmetic progression** is the set $\{p + qn : n = 0, 1, 2, \ldots\}$ for some $p, q \in N$.

(a) (Easy) Show that if $L \subseteq \{a\}^*$ and $\{n : a^n \in L\}$ is an arithmetic progression, then L is regular.

(b) Show that if $L \subseteq \{a\}^*$ and $\{n : a^n \in L\}$ is a union of finitely many arithmetic progressions, then L is regular.

(c) (Harder) Show that if $L \subseteq \{a\}^*$ is regular, then $\{n : a^n \in L\}$ is a union of finitely many arithmetic progressions. (This is the converse of Part (b).)

(d) Show that if Σ is any alphabet and $L \subseteq \Sigma^*$ is regular, then $\{|w| : w \in L\}$ is a union of finitely many arithmetic progressions. (*Hint:* Use Part (c).)

2.4.2. Let $D = \{0, 1\}$ and let $T = D \times D \times D$. A correct addition of two numbers in binary notation can be pictured as a string in T^* if we think of the symbols in T as vertical columns. For example,

$$\begin{array}{r} 0\ 1\ 0\ 1 \\ +\ 0\ 1\ 1\ 0 \\ \hline 1\ 0\ 1\ 1 \end{array}$$

would be pictured as the following string of four symbols.

$$\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}.$$

Show that the set of all strings in T^* that represent correct additions is a regular language.

2.4.3. Show that each of the following is or is not a regular language. The decimal notation for a number is the number written in the usual way, as a string over the alphabet $\{0, 1, \ldots, 9\}$. For example, the decimal notation for 13 is a string of length 2. In **unary notation**, only the symbol I is used; thus 5 would be represented as $IIIII$ in unary notation.

(a) $\{w : w$ is the unary notation for a number that is a multiple of $7\}$.

(b) $\{w : w$ is the decimal notation for a number that is a multiple of $7\}$

(c) $\{w : w$ is the unary notation for a number n such that there is a pair $p, p + 2$ of twin primes, both greater than $n\}$

(d) $\{w : w$ is, for some $n \geq 1$, the unary notation for $10^n\}$

(e) $\{w : w$ is, for some $n \geq 1$, the decimal notation for $10^n\}$

(f) $\{w : w$ is a sequence of decimal digits that occurs in the infinite decimal expansion of $1/7\}$ (For example, 5714 is such a sequence, since $1/7 = 0.14285714285714\ldots$)

2.4.4. Prove that $\{a^n b a^m b a^{m+n} : n, m \geq 1\}$ is not regular.

2.4.5. Using the pumping theorem and closure under intersection, show that the following are not regular.

(a) $\{ww^R : w \in \{a, b\}^*\}$

(b) $\{ww : w \in \{a, b\}^*\}$

(c) $\{w\overline{w} : w \in \{a, b\}^*\}$, where $\overline{w}$ stands for w with each occurrence of a replaced by b, and vice versa.

2.4.6. Call a string x over the alphabet $\{(,)\}$ *balanced* if the following hold: (i) in any prefix of x the number of ('s is no smaller than the number of)'s; (ii) the number of ('s in x equals that of)'s. That is, x is balanced if it could be derived from a legal arithmetic expression by omitting all variables, numbers, and operations. (See the next chapter for a less awkward definition.) Show that the set of all balanced strings in $\{(,)\}^*$ is not regular.

2.4.7. Show that for any deterministic finite automaton $M = (K, \Sigma, \delta, s, F)$, M accepts an infinite language if and only if M accepts some string of length greater than or equal to $|K|$ and less than $2|K|$.

2.4.8. Are the following statements true or false? Explain your answer in each case. (In each case, a fixed alphabet Σ is assumed.)

(a) Every subset of a regular language is regular.
(b) Every regular language has a regular proper subset.
(c) If L is regular, then so is $\{xy : x \in L \text{ and } y \notin L\}$.
(d) $\{w : w = w^R\}$ is regular.
(e) If L is a regular language, then so is $\{w : w \in L \text{ and } w^R \in L\}$.
(f) If C is any set of regular languages, then $\bigcup C$ is a regular language.
(g) $\{xyx^R : x, y \in \Sigma^*\}$ is regular.

2.4.9. The notion of a deterministic 2-tape finite automaton was defined in Problem 2.1.5. Show that $\{(a^n b, a^m b^p) : n, m, p > 0, n = m \text{ or } n = p\}$ is not accepted by any deterministic 2-tape finite automaton. (*Hint:* Suppose this set were accepted by some deterministic 2-tape finite automaton M. Then M accepts $(a^n b, a^{n+1} b^n)$ for every n. Show by a pumping argument that it also accepts $(a^n b, a^{n+1} b^{n+q})$ for some $n \geq 0$ and $q > 0$, a contradiction.) By Problem 2.3.9, then, nondeterministic 2-tape finite automata cannot always be converted to deterministic ones, and by Problem 2.1.5, the sets accepted by deterministic 2-tape finite automata are not closed under union.

.4.10. A **2-head finite automaton** is a finite automaton with two tape heads that may move independently, but from left to right only, on the input tape. As with a 2-tape finite automaton (Problem 2.1.5), the state set is divided into two parts; each part corresponds to reading and moving one tape head. A string is accepted if both heads wind up together at the end of the string with the finite control in a final state. 2-head finite automata may be either deterministic or nondeterministic. Using a state-diagram notation of your own design, show that the following languages are accepted by 2-head finite automata.

(a) $\{a^n b^n : n \geq 0\}$
(b) $\{wcw : w \in \{a, b\}^*\}$
(c) $\{a^1 b a^2 b a^3 b \ldots b a^k b : k \geq 1\}$

In which cases can you make your machines deterministic?

2.4.11. This problem establishes a stronger version of the Pumping Theorem 2.4.1; the goal is to make the "pumped" part as long as possible. Let $M = (K, \Sigma, \delta, s, F)$ be a deterministic finite automaton, and let w be any string in $L(M)$ of length at least $|K|$. Show that there are strings x, y, and z such that $w = xyz$, $|y| \geq (|w| - |K| + 1)/|K|$, and $xy^n z \in L(M)$ for each $n \geq 0$.

2.4.12. Let $D = \{0, 1\}$ and let $T = D \times D \times D$. A correct *multiplication* of two numbers in binary notation can also be represented as a string in T^*. For example, the multiplication $10 \times 5 = 50$, or

$$\begin{array}{r} 0\ 0\ 1\ 0\ 1\ 0 \\ \times\ 0\ 0\ 0\ 1\ 0\ 1 \\ \hline 1\ 1\ 0\ 0\ 1\ 0 \end{array}$$

would be pictured as the following string of six symbols.

$$\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}.$$

Show that the set of all strings in T^* that represent correct multiplications is *not* a regular language. (*Hint:* Consider the multiplication $(2^n + 1) \times (2^n + 1)$.)

2.4.13. Let $L \subseteq \Sigma^*$ be a language, and define $L_n = \{x \in L : |x| \leq n\}$. The *density* of L is the function $d_L(n) = |L_n|$.

(a) What is the density of $(a \cup b)^*$?

(b) What is the density of $ab^*ab * ab^*a$?

(c) What is the density of $(ab \cup aab)^*$?

(d) Show that the density of any regular language is either bounded from above by a polynomial, or bounded from below by an exponential (a function of the form 2^{cn} for some n). In other words, densities of regular languages cannot be functions of intermediate rate of growth such as $n^{\log n}$. (*Hint:* Consider a deterministic finite automaton accepting L, and all cycles —closed paths without repetitions of nodes— in the state diagram of this automaton. What happens if no two cycles share a node? What happens if there are two cycles that share a node?)

2.5 STATE MINIMIZATION

In the last section our suspicion that deterministic finite automata are poor models of computers was verified: Computation based on finite automata cannot

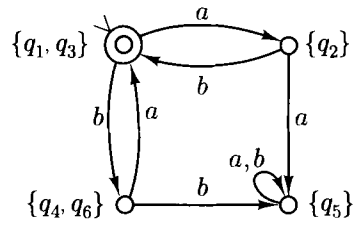


Figure 2-22

Figure 2-22. As expected, it is isomorphic with the standard automaton shown in Figure 2-21.◇

Example 2.5.4: Recall the language $L \subseteq \{a_1, \ldots, a_n\}^*$ of all strings that do not contain occurrences of all n symbols, and the corresponding deterministic finite automaton with 2^n states (Example 2.2.5). We can now show that these states are all necessary. The reason is that $\approx_L$ has 2^n equivalence classes. Namely, for each subset A of Σ, let L_A be the set of all strings containing occurrences of all symbols in A, and of no symbols in $\Sigma - A$ (for example, $L_\emptyset = \{e\}$). Then it is not hard to see that the 2^n sets L_A are precisely the equivalence classes of $\approx_L$. Because if $x \in L_A$ and $y \in L_B$ for two distinct subsets A and B of Σ, then for any $z \in L_{\Sigma - B}$, $xz \in L$, and $yz \notin L$ (here we assumed that B is not contained in A; otherwise, reverse the rôles of A and B).◇

Recall that there is a *nondeterministic* finite automaton with $n + 1$ states that accepts the same language (Example 2.2.2). Although deterministic automata are exactly as powerful as nondeterministic ones *in principle*, determinism comes with a price in the number of states which is, at worst, exponential. To put it in a different way, and in fact a way that anticipates the important issues of *computational complexity* discussed in Chapters 6 and 7: When the number of states is taken into account, *nondeterminism is exponentially more powerful than determinism in the domain of finite automata.*

Problems for Section 2.5

2.5.1. (a) Give the equivalence classes under $\approx_L$ for these languages:
(i) $L = (aab \cup ab)^*$.
(ii) $L = \{x : x \text{ contains an occurrence of } aababa\}$.
(iii) $L = \{xx^R : x \in \{a, b\}^*\}$.
(iv) $L = \{xx : x \in \{a, b\}^*\}$.
(v) $L_n = \{a, b\}^* a \{a, b\}^n$, where $n > 0$ is a fixed integer.
(vi) The language of balanced parentheses (Problem 2.4.6).

(b) For those languages in (a) for which the answer is finite, give a deterministic finite automaton with the smallest possible number of states that accepts the corresponding language.

2.5.2. Call a string $x \in \Sigma^*$ *square-free* if it cannot be written as $x = uvvw$ for some $u, v, w \in \Sigma^*$, $v \neq e$. For example, *lewis* and *christos* are square-free, but *harry* and *papadimitriou* are not. Show that, if $|\Sigma| > 1$, then the set of all square-free strings in Σ^* is not regular.

2.5.3. For each of the finite automata (deterministic or nondeterministic) considered in Problems 2.1.2 and 2.2.9, find the minimum-state equivalent deterministic finite automaton.

2.5.4. A **two-way finite automaton** is like a deterministic finite automaton, except that the reading head can go backwards as well as forwards on the input tape. If it tries to back up off the left end of the tape, it stops operating without accepting the input. Formally, a two-way finite automaton M is a quintuple $(K, \Sigma, \delta, s, F)$, where K, Σ, s, and F are as defined for deterministic finite automata, and δ is a function from $K \times \Sigma$ to $K \times \{\leftarrow, \rightarrow\}$; the $\leftarrow$ or $\rightarrow$ indicates the direction of head movement. A **configuration** is a member of $K \times \Sigma^* \times \Sigma^*$; configuration (p, u, v) indicates that the machine is in state p with the head on the first symbol of v and with u to the left of the head. If $v = e$, configuration (p, u, e) means that M has completed its operation on u, and ended up in state p.

We write $(p_1, u_1, v_1) \vdash_M (p_2, u_2, v_2)$ if and only if $v_1 = \sigma v$ for some $\sigma \in \Sigma$, $\delta(p_1, \sigma) = (p_2, \epsilon)$, and either

1. $\epsilon = \rightarrow$ and $u_2 = u_1\sigma, v_2 = v$, or
2. $\epsilon = \leftarrow, u_1 = u\sigma'$ for some $u \in \Sigma^*$, and $v_2 = \sigma' v_1$.

As usual, $\vdash_M^*$ is the reflexive, transitive closure of $\vdash_M$. M accepts w if and only if $(s, e, w) \vdash_M^* (f, w, e)$ for some $f \in F$. In this problem you will use the Myhill-Nerode Theorem (Theorem 2.5.2 and its corollary) to show that a language accepted by a two-way finite automaton is accepted by a one-way finite automaton. Thus, the apparent power to move the head to the left does not enhance the power of finite automata.

Let M be a two-way finite automaton as just defined.

(a) Let $q \in K$ and $w \in \Sigma^*$. Show that there is at most one $p \in K$ such that $(q, e, w) \vdash_M^* (p, w, e)$.

(b) Let t be some fixed element *not* in K. For any $w \in \Sigma^*$, define a function $\chi_w : K \mapsto K \cup \{t\}$ as follows.

$$\chi_w(q) = \begin{cases} p, & \text{if } (q, e, w) \vdash_M^* (p, w, e) \\ t, & \text{otherwise} \end{cases}$$

By Part (a), χ_w is well defined. Also, for any $w \in \Sigma^*$, define $\theta_w : K \times \Sigma \mapsto K \cup \{t\}$ as follows:

$$\theta_w(q,a) = \begin{cases} p, & \text{if } (q,w,a) \vdash_M^+ (p,w,a) \text{ but it is not the case that} \\ & \quad (q,w,a) \vdash_M^+ (r,w,a) \vdash_M^+ (p,w,a) \text{ for any } r \neq p, \\ t, & \text{if there is no } p \in K \text{ such that } (q,w,a) \vdash_M^+ (p,w,a) \end{cases}$$

(Here by $\vdash_M^+$ we mean the transitive (not reflexive) closure of $\vdash_M$, that is, the "yields in one or more steps" relation on configurations.) Now suppose that $w, v \in \Sigma^*$, $\chi_w = \chi_v$, and $\theta_w = \theta_v$. Show that, for any $u \in \Sigma^*$, M accepts wu if and only if M accepts vu.

(c) Show that, if $L(M)$ is the language accepted by a deterministic two-way automaton, then $L(M)$ is accepted by some ordinary (one-way) deterministic finite automaton. (*Hint:* Use (b) above to show that $\approx_{L(M)}$ has finitely many equivalence classes.)

(d) Conclude that there is an exponential algorithm which, given a deterministic two-way automaton M, constructs an equivalent deterministic finite automaton. (*Hint:* How many different functions χ_w and θ_w can there be, as a function of $|K|$ and $|\Sigma|$?)

(e) Design a deterministic two-way finite automaton with $O(n)$ states accepting the language $L_n = \{a,b\}^* a \{a,b\}^n$ (recall Problem 2.5.1(a)(v)). Comparing with Problem 2.5.1(b)(v), conclude that the exponential growth in (d) above is necessary.

(f) Can the argument and construction in this problem be extended to *nondeterministic* two-way finite automata?

2.6 ALGORITHMS FOR FINITE AUTOMATA

Many of the results in this chapter were concerned with different ways of representing a regular language: as a language accepted by a finite automaton, deterministic or nondeterministic, and as a language generated by a regular expression. In the previous section we saw how, given any deterministic finite automaton, we can find the equivalent deterministic finite automaton with the fewest possible states. All these results are constructive, in that their proofs immediately suggest *algorithms* which, given a representation of one kind, produce a representation of any one of the others. In this subsection we shall make such algorithms more explicit, and we shall analyze roughly their complexity.

We start from the algorithm for converting a nondeterministic finite automaton to a deterministic one (Theorem 2.2.1); let us pinpoint its complexity. The input to the algorithm is a nondeterministic finite automaton $M =$

—a construction which we know is potentially exponential. Fortunately, in the case of nondeterministic automata arising in string-matching applications, the subset construction *is always efficient*, and the resulting deterministic automaton M_x has exactly the same number of states as the original nondeterministic one (see Figure 2-25(b)). It is clearly the minimal equivalent automaton. This automaton M_x is therefore an algorithm for testing whether $w \in L_x$ in time $\mathcal{O}(|w|)$, for any string $w \in \Sigma^*$.

Still, this algorithm has a drawback that makes it unsuitable for the many practical applications of string matching. In real applications, the underlying alphabet Σ has several dozens, often hundreds, of symbols. A deterministic finite automaton rendered as an algorithm must execute for each input symbol a long sequence of if statements, one for each symbol of the alphabet (recall the first algorithm in this subsection). In other words, the $\mathcal{O}$ notation in the $\mathcal{O}(|w|)$ running time of the algorithm "hides" a potentially large constant: the running time is in fact $\mathcal{O}(|\Sigma||w|)$. For a clever remedy, see Problem 2.6.3.

Problems for Section 2.6

2.6.1. Show that these two regular expressions do not represent the same language: $aa(a \cup b)^* \cup (bb)^*a^*$ and $(ab \cup ba \cup a)^*$. Do so
(a) by subjecting them to a general algorithm; and
(b) by finding a string generated by one and not by the other.

2.6.2. (a) What is the sequence of S_i's produced if the nondeterministic finite automaton in Example 2.6.1 is presented with input *bbabbabba*?
(b) Prove that the algorithm for running a nondeterministic finite automaton with m states on an input of length n takes time $\mathcal{O}(m^2n)$.
(c) Suppose that the given nondeterministic finite automaton has at most p transitions from each state. Show that an $\mathcal{O}(mnp)$ algorithm is possible.

2.6.3. Let Σ be an alphabet, $x = a_1 \ldots a_n \in \Sigma^*$, and consider the nondeterministic finite automaton $M_x = (K, \Sigma, \Delta, s, F)$, where $K = \{q_0, q_1, \ldots, q_n\}$, $\Delta = \{(q_{i-1}, a_i, q_i) : i = 0, \ldots, n-1\} \cup \{(q_i, a, q_i) : a \in \Sigma, i \in \{0, n\}\}$ (recall Figure 2-25).
(a) Show that $L(M_x) = \{w \in \Sigma^* : x \text{ is a substring of } w\}$.
(b) Show that the deterministic finite automaton M'_x with the fewest states that is equivalent to M_x also has $n+1$ states. What is the worst-case time required for its construction, as a function of n?
(c) Show that there is a nondeterministic finite automaton M''_x equivalent to M_x, also with $n+1$ states $\{q_0, q_1, \ldots, q_n\}$, and with the following important property: *Each state except q_0 and q_n has exactly two transitions out of it, of which one is an e-transition.* (*Hint:* Replace each

backwards transition in the deterministic finite automaton on Figure 2-25 by an appropriate e-transition; generalize.)

(d) Argue that M_x'' remedies the problem of the hidden constant $|\Sigma|$ discussed in the last paragraph of the text.

(e) Give an algorithm for constructing M_x'' from x. What is the complexity of your algorithm as a function of n?

(f) Devise an $\mathcal{O}(n)$ algorithm for the problem in (e) above. (*Hint:* Suppose that the e-transitions of M_x'' are $(q_i, e, q_{f(i)}), i = 1, \ldots, n-1$. Show how to compute $f(i)$, based on the values $f(j)$ for $j < i$. A clever "amortized" analysis of this computation gives the $\mathcal{O}(n)$ bound.)

(g) Suppose that $\Sigma = \{a, b\}$ and $x = aabbaab$. Construct M_x, M_x', and M_x''. Run each of these automata on the input *aababbaaabbaaabbaabb*.

REFERENCES

Some of the first papers on finite automata were

- G. H. Mealy "A method for synthesizing sequential circuits," *Bell System Technical Journal, 34,* 5 , pp. 1045–1079, 1955, and
- E. F. Moore "Gedanken experiments on sequential machines," *Automata Studies,* ed. C. E. Shannon and J. McCarthy, pp. 129–53. Princeton: Princeton University Press, 1956.

The classical paper on finite automata (containing Theorem 2.2.1) is

- M. O. Rabin and D. Scott "Finite automata and their decision problems," *IBM Journal of Research and Development, 3,* pp. 114–25, 1959.

Theorem 2.3.2, stating that finite automata accept regular languages, is due to Kleene:

- S. C. Kleene "Representation of events by nerve nets," in *Automata Studies,* ed. C. E. Shannon and J. McCarthy, pp. 3–42. Princeton: Princeton University Press, 1956.

Our proof of this theorem follows the paper

- R. McNaughton and H. Yamada "Regular expressions and state graphs for automata," *IEEE Transactions on Electronic Computers, EC-9,* 1 pp. 39–47, 1960.

Theorem 2.4.1 (the "pumping lemma") is from

- V. Bar-Hillel, M. Perls, and E. Shamir "On formal properties of simple phrase structure grammars," *Zeitschrift fur Phonetik, Sprachwissenschaft, und Kommunikationsforschung, 14,* pp. 143–172, 1961.

Finite-state transducers (Problem 2.1.4) were introduced in

- S. Ginsburg "Examples of abstract machines," *IEEE Transactions on Electronic Computers, EC-11,* 2, pp. 132–135, 1962.

Two-tape finite state automata (Problems 2.1.5 and 2.4.7) are examined in

- M. Bird "The equivalence problem for deterministic two-tape automata," *Journal of Computer and Systems Sciences, 7,* pp. 218–236, 1973.

The Myhill-Nerode Theorem (Theorem 2.5.2) is from