

Solutions to Selected Exercises for Chapter 9 of R. Gregory Taylor, *Models of Computation and Formal Languages* (Oxford University Press: New York, 1998)

© Oxford University Press

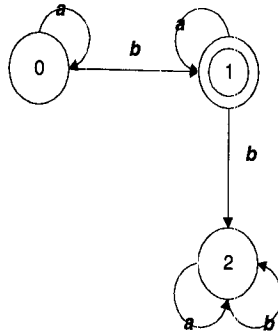
Solutions to Exercises for § 9.1

- 9.1.3 (a) Language consisting of all and only the words over Σ containing exactly three occurrences of 1
(b) All binary strings of length 2 or more that begin and end with a 0
(c) All binary strings
(d) All binary strings of length at least 3 whose next-next-to-last bit is 0
(e) Let us describe the bit pairs 00 and 11 as *like pairs*; 01 and 10 are *unlike pairs*. Then this language consists of an arbitrary sequence of like pairs, followed by a pair sequence containing an even number of unlike pairs.
- 9.1.4. $(011|213141516171819)^+$. $(011|213141516171819)^+$ is a regular expression that denotes the language in question. (Literals involving scientific notation are not being included here.)
- 9.1.5 (a) $(011)^*0$
(b) $(011)^*1$
(c) $(011)^*000$
- 9.1.6 (a) The language denoted is just the language containing all and only those words in which at least one occurrence of b follows directly upon an occurrence of a . But this language is denoted by the simpler expression $(alb)^*ab(alb)^*$.
(b) The language denoted is just the language containing all and only those words in which either (1) at least one occurrence of b follows directly upon an occurrence of a or (2) at least one occurrence of a follows directly upon an occurrence of b . But this language is denoted by the simpler expression $(alb)^*((ab)|(ba))(alb)^*$.
- 9.1.7 (a) $b^*ab^*ab^*a(alb)^*$
(b) $(alba)^*$
(c) $((alb)(alb))^*$
(d) $((alb)(alb))^*(alb)$
(e) $((alb)(alb)(alb))^*$

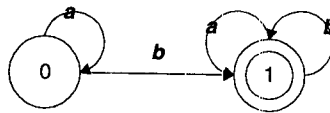
Solutions to Exercises for § 9.2

- 9.2.5. Only $abbba$ and aa are accepted. The language accepted is denoted by regular expression ab^*a .

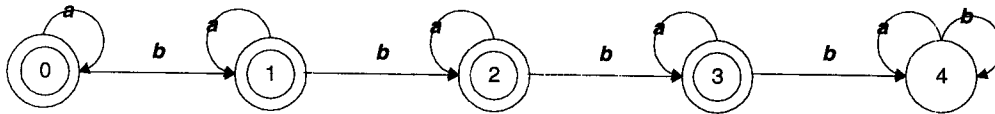
9.2.6 (a)



(b)



(c)



9.2.7 (a) $Q \setminus F$

(b) $\Sigma^* \setminus \{ab^n \mid n \geq 0\}$ with $\Sigma = \{a, b\}$

(c) **Proposition.** Let finite-state automaton $M = \langle \Sigma, Q, q_{init}, F, \delta_M \rangle$ be given. Let finite-state automaton $M' = \langle \Sigma, Q, q_{init}, Q \setminus F, \delta_M \rangle$. Then $L(M') = \Sigma^* \setminus L(M)$.

Proof. Suppose that $w \in L(M')$. Then since M' is fully defined, there exists a path labeled by w from q_{init} to a nonaccepting state of M . That is, $w \notin L(M)$. Each step here is reversible.

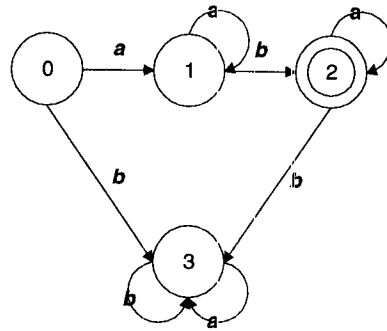
Q.E.D.

9.2.8 (a) $\{ab^nw \mid w \in \Sigma^*\} = L(ab(ab)^*)$

(b) This machine accepts that language consisting of all and only words w with $n_a(w) = n_b(w)$ and such that any prefix of w contains at most one more a than b or at most one more b than a .

9.2.9 (a) The language accepted is denoted by regular expression $b^*ab(ab)^*$.

(b)



9.2.11. $L(M)$ is Σ^* .

Solutions to Exercises for § 9.4

9.4.2. Our argument is indirect. So we start by assuming that

$$L = \{a^n \mid n \text{ is a perfect square}\} = \{a^1, a^4, a^9, a^{16}, \dots\}$$

is FSA-acceptable. Let M be a finite-state machine that accepts L . Since L is, by hypothesis, FSA-acceptable and since L is plainly infinite, the Pumping Lemma applies and tells us that there exist words u, w, v with $w \neq \epsilon$ such that

$$uw^0v, \quad uw^1v, \quad uw^2v, \quad uw^3v, \dots$$

are all in L so that

$$\begin{array}{cccc} |uw^0v|, & |uw^1v|, & |uw^2v|, & |uw^3v|, \dots \\ \downarrow & \downarrow & \downarrow & \downarrow \\ n, & n+m, & n+2m, & n+3m, \dots \end{array}$$

must all be perfect squares. Note that words $u, w,$ and v must consist of a 's only. Letting $n=|u|+|v|$ and letting $m=|w| \neq 0$, we see that natural numbers

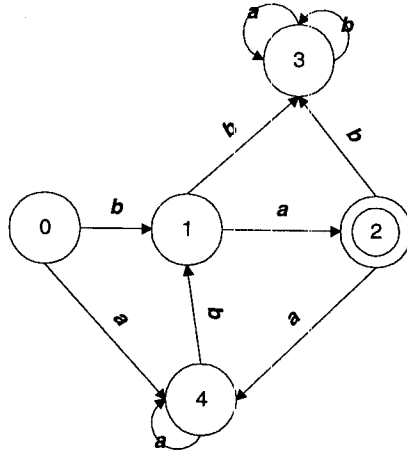
must all be perfect squares. But this contradicts Theorem 0.8. We conclude that L is not FSA-acceptable after all. Q.E.D.

9.4.4 (a) No contradiction results if we take w in the statement of Lemma 9.2 to consist of a 's only. In other words, the Pumping Lemma, although applicable to L , does not produce a contradiction in this particular case.

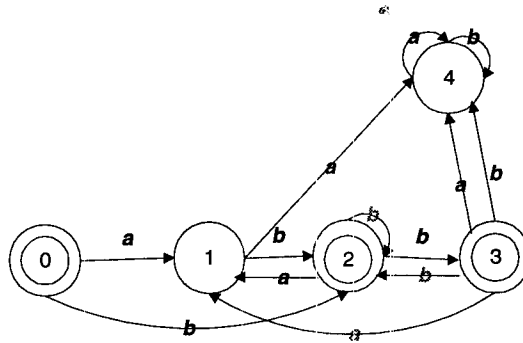
(b) The answer is, of course, no. Language L is FSA-acceptable. But we see this as the result of presenting an accepting finite-state automaton or as the result of some other positive argument. In general, p does not follow from having failed to prove $\neg p$. Along the same lines, bear in mind that the Pumping Lemma is useful in obtaining negative results only.

Solutions to Exercises for § 9.5

9.5.4.



9.5.5.



The construction requires adding an a -arc from accepting state q_2 to state q_1 . However, such an arc already exists in M' 's state diagram.

Solutions to Exercises for § 9.6

9.6.1.

$$\begin{aligned}
 M_{0,5}^{-1} &= \emptyset \\
 M_{0,5}^0 &= \emptyset \\
 M_{0,5}^1 &= L(ab) \\
 M_{0,5}^2 &= L(ab|aab) \\
 M_{0,5}^3 &= L(ab|aab|bab) \\
 M_{0,5}^4 &= L(ab|aab|bab|bba^*b|baa^+b) \\
 M_{0,5}^5 &= L[(ab|aab|bab|bba^*b|baa^+b)(ab)^*]
 \end{aligned}$$

9.6.2. By Theorem 9.5, $\{a^n b^n | n \geq 0\}$ is not FSA-acceptable. By Theorem 9.12, it is not regular either.

9.6.3. This line of reasoning would be circular in that our proof of Theorem 9.12 appealed to closure results for the family of FSA-acceptable languages.

Solutions to Exercises for § 9.8

- 9.8.3 (a) G 's nonterminals are S , S' , and C . Its terminals are a , b , and c . G has six productions.
 (b) $\{a^n b^{2n} c^{3n} | n \geq 0\}$
 Word ε is generated.
- 9.8.4 (a) Word ε is not generated. In fact, G generates language \emptyset .
 (b) If G is a generative grammar every production right-hand side of which contains at least one nonterminal, then $L(G)$ will be empty.
- 9.8.5 (a) G generates the language denoted by regular expression $((aa)^+ b^+)^+$. Word ε is not generated.
 (b) Grammar G generates ε only if G contains empty production(s). Note that this is a necessary but not a sufficient condition of generation of ε .
- 9.8.7 (a) $S \Rightarrow XYZ$
 $\Rightarrow IXWZ$
 $\Rightarrow IXYIZ$
 $\Rightarrow IOXUIZ$
 $\Rightarrow IOXIUZ$
 $\Rightarrow IOXIYOZ$
 $\Rightarrow IOXYIOZ$
 $\Rightarrow IOIOZ$
 $\Rightarrow IOIO$
- (b) $S \Rightarrow XYZ$
 $\Rightarrow IXWZ$
 $\Rightarrow IXYIZ$
 $\Rightarrow IIXWIZ$
 $\Rightarrow IIXIWZ$
 $\Rightarrow IIXIYIZ$
 $\Rightarrow IIXYIIZ$
 $\Rightarrow IIIIIZ$
 $\Rightarrow IIII$
- (c) $S \Rightarrow XYZ$
 $\Rightarrow OXUZ$
 $\Rightarrow OXYOZ$
 $\Rightarrow OIXW0Z$
 $\Rightarrow OIXOWZ$
 $\Rightarrow OIXOYIZ$
 $\Rightarrow OIXYOIZ$
 $\Rightarrow OIOIZ$
 $\Rightarrow OIOI$

- (d) $S \Rightarrow XYZ$
 $\Rightarrow 0XUZ$
 $\Rightarrow 0XY0Z$
 $\Rightarrow 00XU0Z$
 $\Rightarrow 00X0UZ$
 $\Rightarrow 00X0Y0Z$
 $\Rightarrow 00XY00Z$
 $\Rightarrow 0000Z$
 $\Rightarrow 0000$

Solutions to Exercises for § 9.9

- 9.9.2 (a) $S \rightarrow X_{q_0}$
 $X_{q_0} \rightarrow aX_{q_1}bX_{q_3}$
 $X_{q_1} \rightarrow aX_{q_2}bX_{q_1}$
 $X_{q_2} \rightarrow aX_{q_5}bX_{q_5}$
 $X_{q_3} \rightarrow aX_{q_4}bX_{q_5}$
 $X_{q_4} \rightarrow aX_{q_5}bX_{q_3}$
 $X_{q_5} \rightarrow aX_{q_5}bX_{q_5}$

(b) Path P is given by the node sequence $q_0q_3q_4q_3q_4q_3q_4$. Derivation D appears below.

- $S \Rightarrow X_{q_0}$
 $\Rightarrow bX_{q_3}$
 $\Rightarrow baX_{q_4}$
 $\Rightarrow babX_{q_3}$
 $\Rightarrow babaX_{q_4}$
 $\Rightarrow bababX_{q_3}$
 $\Rightarrow bababaX_{q_4}$
 $\Rightarrow bababa$

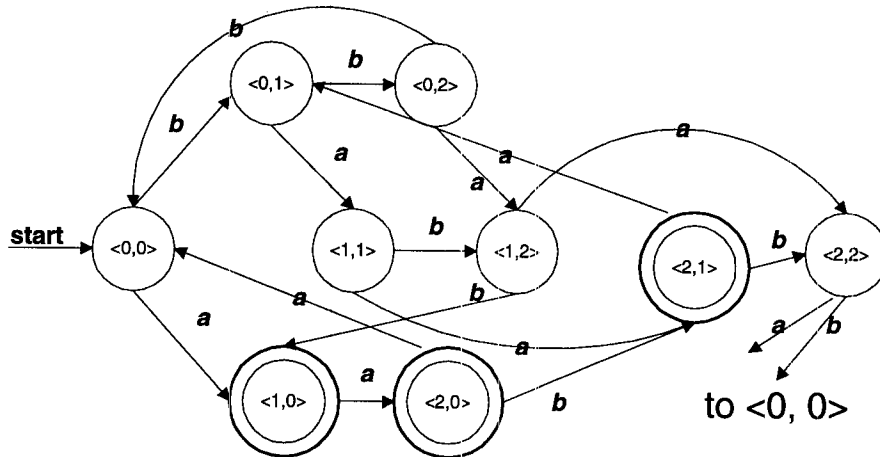
The path corresponds exactly to the (subscripts of) the nonterminals appearing on the right in successive lines of the derivation.

Solutions to Exercises for § 9.10

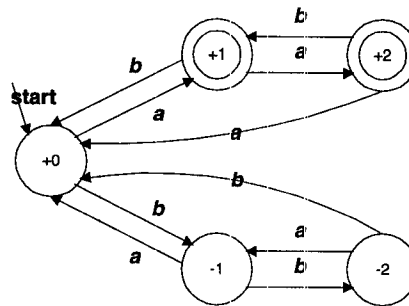
- 9.10.1 (a) Writing “ Φ ” for $(A|B|C|D|F|G|\dots|X|Y|Z)$, we have $\Phi^*a\Phi^*e\Phi^*i\Phi^*o\Phi^*u\Phi^*$
 (b) $A^*B^*C^*D^*\dots X^*Y^*Z^*$
- 9.10.2. $/*((A|B|C|\dots|Z|a|\dots|z|0|1|\dots|9|\$|\&|\dots|!|\%|/))*((A|B|C|\dots|Z|a|\dots|z|0|1|\dots|9|\$|\&|\dots|!|\%))*(*/*/*)*/*/$
- 9.10.4 (a) $(a|(ba))^*(b|(ab))^*$
 (b) $(\epsilon|a|ba)((bb^*b)|(aa^*a))^*(\epsilon|b|a|ab|abb)$ (Tony Mendoza suggests $b^*((abb^+|a)^*(b|\epsilon))$ as an alternative.)
- 9.10.5 (a) The language consisting of all and only words such that any odd-length sequence of bs precedes any odd-length sequence of as

- (b) The language consisting of all and only words containing no occurrence of three consecutive b s
 (c) The language of all and only those words w such that both $n_a(w)$ and $n_b(w)$ are even

9.10.8 (d)

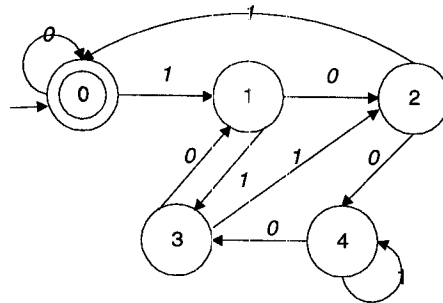


(e)



This machine is instructive to the extent that it can be used to illustrate a remark that we made in introducing finite-state automata. (The point we are about to make could be made using the machines of (a)-(d) as well.) Namely, there can be no finite-state automaton that accepts the language L consisting of all and only those words w over $\Sigma = \{a, b\}$ such that the difference $n_a(w) - n_b(w)$ is 20, say. Intuitively, this is because an infinite amount of storage would be required, although the Pumping Lemma can be used to prove that L is not FSA-acceptable. (How?) (Language L would be Turing-acceptable, of course, because the unbounded quantity of storage required to carry out the needed computation for input words w of arbitrary length is available in the case of Turing machines.) However, what a finite-state automaton *can* do is keep track, not of the absolute difference between $n_a(w)$ and $n_b(w)$ but, rather, of whether that difference modulo 3 is 0, 1, or 2. This more limited task requires only finite storage. See also Exercise 9.10.26.

9.10.9.



- 9.10.15 (a) This is a regular language denoted by regular expression $(aa)^*$.
 (b) This is not a regular language; use the Pumping Lemma (Lemma 9.2).
 (c) This is a regular language: it is the complement of the language denoted by $(alb)^*(bbb)(alb)^*$.
 (d) This is a regular language: it is the union of (i) the language denoted by $(alb)^*(bbb)(alb)^*$ and (ii) the complement of the language denoted by $(alb)^*(aa)(alb)^*$.
 (e) This is not a regular language: if it were regular, then $L(a^*b^*) \cap \{w | n_a(w) = n_b(w)\} = \{a^n b^n | n \geq 0\}$ would be regular and, by Theorem 9.12, FSA-acceptable as well, contradicting Theorem 9.5.

- 9.10.17 (a) Since $|w| \geq n$, we see that the unique path P labeled by w in M 's state diagram visits $n+1$ or more states. So at least one state is visited twice in P , which means that P must contain a closed path P_1 . Suppose that P_1 is labeled by nonempty word w_1 so that w is of the form uw_1v for (possibly empty) words u and v . Then $uw_1^0v = uv$, $uw_1^1v = uw_1v$, uw_1^2v , uw_1^3v , and so on, are all in $L(M)$.

(b) By $L(M)$ infinite, we have that there is no upper bound on the length of words in $L(M)$. So there must be words w with $|w| \geq 2n$. Let w_0 be a shortest word w with $|w| \geq 2n$. Let P be the unique path labeled by w_0 in M 's state diagram. Since M has but n states and since P visits at least $2n+1$ states, there exists a closed path P_1 within P labeled by nonempty word w_1 so that w_0 is of the form uw_1v for (possibly empty) words u and v . Further, by Corollary 0.1 of § 0.7, we can assume without loss of generality that P_1 is a cycle and hence that $|w_1| \leq n$. Clearly, uv is also in $L(M)$. Supposing that $|uv| \geq 2n$ contradicts the definition of w_0 as a word w of shortest length with $|w| \geq 2n$, since $|uv| < |w_0|$. So $|uv| < 2n$. But since uv is the result of removing w_1 with $|w_1| \leq n$ from w_0 , we have that $|uv| \geq n$ as well.

(c) Let n be the number of states in finite-state automaton M and let Σ be M 's alphabet. We first enumerate all words w over Σ with $n \leq |w| < 2n$. This list is of finite length. For each word w on the list, check to see whether M accepts w . If one of them is accepted, then we may stop and conclude by (a) that $L(M)$ is infinite. On the other hand, if M accepts no word on the list, then by (the contrapositive of) (b), we may conclude that $L(M)$ is finite.

- 9.10.19 (a) Suppose that $L=L_r$ is defined by regular expression r . We use induction on the complexity of r .

Base case.

- (i) Suppose r is \emptyset , in which case $L=L_r$ is the empty language. Then $L^R=L$ and is hence regular.
- (ii) Suppose that r is ϵ , in which case $L=L_r$ is unit language $\{\epsilon\}$. Then again $L^R=L$ and is hence regular.
- (iii) Suppose that r is a , in which case L is unit language $\{a\}$. Then $L^R=\{a\}$ and is hence regular by Definition 9.1(iii). (The case where r is b is similar.)

Inductive case.

- (iv)(a) Suppose that r is (slt) , in which case $L=L_r$ is $L_s \cup L_t$. By induction hypothesis both L_s^R and L_t^R are regular. But then, by Theorem 9.1, so is their union, which is just L^R .

(iv)(b) Suppose that r is $(s.t)$, in which case $L=L_r$ is $L_s.L_t$. By induction hypothesis both L_s^R and L_t^R are regular. But then, by Theorem 9.1, so is their concatenation $L_t^R.L_s^R$, which is just L^R .

(iv)(c) Suppose that r is (s^*) , in which case $L=L_r$ is $(L_s)^*$. By induction hypothesis L_s^R is regular. But then, by Theorem 9.1, so is its Kleene closure $(L_s^R)^*=[(L_s)^*]^R=L^R$. Q.E.D.

(b) Suppose that $L=L_r$ is defined by regular expression r . We use induction on the complexity of r .

Base case.

(i) Suppose r is \emptyset , in which case $L=L_r$ is the empty language. Then $L^{-1}=L$ and is hence regular.

(ii) Suppose that r is ϵ , in which case $L=L_r$ is unit language $\{\epsilon\}$. Then again $L^{-1}=L$ and is hence regular.

(iii) Suppose that r is a , in which case L is unit language $\{a\}$. Then $L^{-1}=\{b\}$ and is hence regular by Definition 9.1(iii). (The case where r is b is similar.)

Inductive case.

(iv)(a) Suppose that r is (slt) , in which case L is $L_s \cup L_t$. By induction hypothesis both L_s^{-1} and L_t^{-1} are regular. But then, by Theorem 9.1, so is their union, which is just L^{-1} .

(iv)(b) Suppose that r is $(s.t)$, in which case L is $L_s.L_t$. By induction hypothesis both L_s^{-1} and L_t^{-1} are regular. But then, by Theorem 9.1, so is their concatenation $L_s^{-1}.L_t^{-1}$, which is just L^{-1} .

(iv)(c) Suppose that r is (s^*) , in which case $L=L_r$ is $(L_s)^*$. By induction hypothesis L_s^{-1} is regular. But then, by Theorem 9.1, so is its Kleene closure $(L_s^{-1})^*=[(L_s)^*]^{-1}=L^{-1}$. Q.E.D.

9.10.20. By assumption, L is regular. By Exercise 9.10.19(a), so is L^R . But then by Theorems 9.9, 9.12, and 9.13, so is $L \cap L^R=L'$.

9.10.21. Let G be a left-linear grammar. Let G' be the right-linear grammar formed from G by reversing the right-hand side of every production of G . Clearly, $L(G')$ is $[L(G)]^R$. Moreover, $L(G')=[L(G)]^R$ is regular by Theorem 9.15, from which it follows, by Exercise 9.10.19(a), that $[[L(G)]^R]^R=L(G)$ is regular as well.

9.10.22. By Exercise 9.10.19(a), language L^R is regular. It follows from Theorem 9.16 that L^R is generated by a right-linear grammar G . Reversing all production right-hand sides, one obtains a left-linear grammar G^R generating $(L^R)^R=L$.

9.10.23 (b) Typically, the class of identifiers of a given programming language is defined by first characterizing some super-class S of alphanumeric strings, beginning with a letter, and perhaps permitting certain special characters in certain positions. One then excludes some finite collection R of reserved words. But both S and R will be regular languages. Hence, by closure under set difference, so is the class of identifiers $S \setminus R$.

9.10.25 (a) If we agree that the processing of one input symbol amounts to one "step" in M 's computation, it follows that M computes in linear time, i.e., in time $O(n)$ for n the length of input.

(b) If execution of ϵ -moves are assumed to be instantaneous, requiring no time, then the time analysis is again $O(n)$ trivially. On the other hand, if we decide to charge for ϵ -moves as well, then the analysis is somewhat trickier although instructive. It is still true that M computes in time $\Omega(n)$ since M must read in its entire input. However, we should like to say more.

Analogous to our convention with respect to nondeterministic Turing machines, let us agree that the cost of M 's computation for accepted word w will be the minimum length of any path labeled by w from start state q_0 to some accepting state. If w is not accepted, then we shall use the minimum over all paths labeled by w . Next, $time_M(n)$ will be the maximum cost of M 's computation for any input word of length n . (Thus $time_M(n)$, in the nondeterministic case, exhibits the familiar "maxmin" character. See

§ 2.6)

Further, we note the following.

- (1) M has only a fixed finite number of states and, hence, its transition diagram can have only a fixed finite number N of arcs labeled by ϵ . Moreover, N is $O(1)$, i.e., independent of input length n .
- (2) It is possible that the state diagram of M contains cycles each arc of which is labeled by ϵ . However, any finite path P labeled by input word w and containing such cycles may be replaced by another shorter path P' that is labeled by w and that lacks such cycles. (Since full definition implies that, for every input word w , there exists some finite path labeled by w , we can, by *maxmin*, ignore infinite paths as irrelevant.)
- (3) Having eliminated all ϵ -labeled cycles, we can see by (1) that, within any remaining finite path labeled by w , no ϵ -labeled subpath can be of length greater than N .

From the foregoing, it should now be evident that any path labeled by w may be assumed to contain at most $n-1$ ϵ -labeled subpaths, each of length $O(1)$. The length of this path itself is hence $O(n)$. Consequently, the minimum length of all such paths is $O(n)$. Also, the maximum, taken over all input words of length n , of these minimum lengths will also be $O(n)$. That is, $time_M(n)$ is $O(n)$ even in the nondeterministic case.

(c) If we consider a finite-state automaton to possess an input buffer, then it is reasonable to take length of input itself as the space requirement of that machine. On this view, every machine would compute in linear space. That is no doubt the most reasonable point of view. But, in this connection, we mention the following. As noted in Exercise 9.10.12(a), it is straightforward to transform any deterministic finite-state automaton into a single-tape Turing machine that accepts the very same language. The same holds true of any nondeterministic finite-state automata. Further, it is not much harder to transform such finite-state automata into two-tape, off-line Turing machines. Moreover, since the latter have no worktapes, they compute in $O(0)$ space. We may conclude that any regular language is accepted by an off-line Turing machine that requires $O(0)$ space.

(d) We have suggested one way of thinking about time and two different ways of thinking about the space requirements of finite-state automata. These interpretations of time and space are, in and of themselves, quite reasonable. On the other hand, it is also obvious that, no matter how we do things, time and space will not enable us to make any distinctions within the class of finite-state automata (or within the class of regular languages) since they will all of them have precisely the same time requirements and precisely the same space requirements. Consequently, from within the theory of regular languages itself, such analyses are entirely pointless. On the other hand, from within the theory of Turing-acceptable languages, it is not uninteresting to note that every regular language is acceptable in linear time and $O(0)$ space.

9.10.26. Suppose that language L is accepted by multitape Turing machine M in space $O(1)$. By Exercise 2.3.19 we may assume without loss of generality that M consists of read-only input tape and write-only output tape and nothing more. It follows that the sort of processing of its input word of which M is now capable may be carried out by a finite-state machine processing its input from left to right one character at a time. It follows that L is regular.