Solutions to Exercises for § 0.14

0.14.1 (a) This exercise is trivial, which is our point in bringing it to the reader's attention. Namely, if, for every natural number $C \geq 1$, there exists natural number n_0 with $f(n) \leq \frac{1}{C} g(n)$ for $n \geq n_0$, then letting $C=1$ gives $f(n) \leq 1 \cdot g(n)$ for $n \geq n_0$, which is to say that $f(n)$ is $O(g(n))$.

(b) If $f(n)$ and $g(n)$ differ by a constant, then $f(n)$ will be $O(g(n))$ but $f(n)$ will not be $o(g(n))$, for example, if $g(n)=f(n)+10$, say. Similarly, for any two polynomial functions $f(n)$ and $g(n)$ of like degree and with positive leading coefficients, we have that $f(n)$ is $O(g(n))$ and, in fact, $\Theta(g(n))$. But $f(n)$ will not be $o(g(n))$ nor will $g(n)$ be $o(f(n))$.

0.14.2.[hwk] Suppose that the class of total, unary functions were countable. In that case, we may assume an enumeration

$$f_0, f_1, f_2, \ldots$$

Set $f^*(n)=f_n(n)+1$ and note that f^* is unary and total. Now derive a contradiction.

Solutions to Selected Exercises for Chapter 1

Solutions to Exercises for § 1.1

1.1.2.[hwk] (a)

i	$quot$	$b[i]$
—	783	—
0	391	1
1	195	1
2	97	1
3	48	1
4	24	0
5	12	0
6	6	0
7	3	0
8	1	1
9	0	1

The binary equivalent of 783_{10} is read in the right-hand column from bottom to top.

(b) The algorithm is determinate in the sense that, at any point in carrying it out, it is always clear what the next step of the algorithm is. In addition, it is determinate in the sense that it terminates after a finite number of steps, having produced a certain definite output. Finally, this output is unambiguous in the sense that there is no question as to how it is to be interpreted.

1.1.4. Direct apprehension is suggested by the speed with which the brothers classify numbers—as either prime or composite apparently. Given our own slowness in carrying out the familiar algorithm for deciding whether a given number is prime, we want to say that the brothers simply *see* that a given natural number has one or the other property. On the other hand, the fact that classification of a 13-digit number, although yet fast, takes them considerably longer than classification of a 10-digit number, say, suggests computation involving some algorithm. But what could that algorithm be, given that the brothers seem to possess no concept of division?

One attempt at an explanation would involve positing, in the case of the twins, some unusual, highly specific neural representation of number that would be especially advantageous for discerning primality—and apparently not much else. Then the brothers' activity could be regarded as "symbolic" manipulation of these representations in the usual way. This view is not unappealing. The only real alternative, it seems, is to hold that the brothers are engaged in some activity that is "half perception" (intuition) and "half computation." But this would, in effect, amount to saying that, given our current battery of concepts, we have absolutely no hope of explaining what it is that the brothers are doing and how they are doing it.

1.1.5 (a) Suppose that $f(n)$ is some unary, computable, number-theoretic function and that $f(3)=7$, say. Then f may be regarded as transforming string *111*, or the like, into string *1111111*. In other words, function computation may be construed as a special case of transduction.

As for language recognition, recall that recognition of language L is describable as computation of characteristic function $\chi_{Codes(L)}$ of the set $Codes(L)$ of codes of words in L. But we just saw that computation of such a function is, in turn, describable as an instance of transduction.

(b) Suppose that some computation involves transforming an arbitrary input word w over some alphabet into its reversal w^R, say. Assigning symbol codes, we may redescribe this in terms of function computation by considering the unary number-theoretic function f defined by

$$f(\ulcorner w \urcorner) =_{\text{def.}} \ulcorner w^R \urcorner$$

where we are writing $\ulcorner w \urcorner$ and $\ulcorner w^R \urcorner$ for the induced encodings of w and w^R, respectively.

As for the language recognition paradigm, note that any computation transforming arbitrary input w into w^R is describable as recognition of that language consisting of all and only those strings of the form

$$w\#w^R$$

at least under certain reasonable assumptions.

Solutions to Exercises for § 1.2

1.2.1 [hwk] (a) If started scanning a blank on a completely blank tape, then M_0 writes exactly a b followed by an a and halts reading the b.

If started scanning a blank on a completely blank tape, then M_1 writes exactly three *1*s and halts scanning the leftmost of these *1*s.

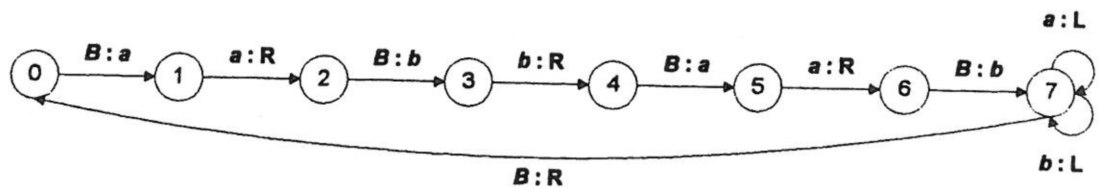
If started scanning a blank on a completely blank tape, then M_2 writes exactly three *1*s and halts scanning the leftmost of these *1*s. (In other words, its behavior is in a certain sense equivalent to that of machine M_1.)

If started scanning a blank on a completely blank tape, then M_3 writes a *1* to the right and a *1* to the left, then another *1* to the right and another *1* to the left—back and forth—without ever halting.

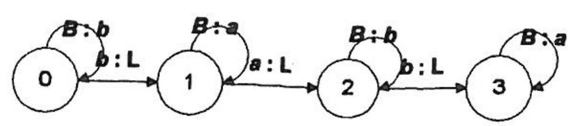
If started scanning a blank on a completely blank tape, then M_4 writes the string *ab* off to the right and then off to the left without ever halting.

(b) $\delta_{M_1}(q_0,B)=(1,q_1)$
$\delta_{M_1}(q_1,1)=(\mathrm{R},q_2)$
$\delta_{M_1}(q_2,B)=(1,q_3)$
$\delta_{M_1}(q_3,1)=(\mathrm{R},q_4)$
$\delta_{M_1}(q_4,B)=(1,q_5)$
$\delta_{M_1}(q_5,1)=(\mathrm{L},q_5)$
$\delta_{M_1}(q_5,B)=(\mathrm{R},q_6)$

1.2.2 (a) Here is one machine that works:



But a machine with fewer states is possible:

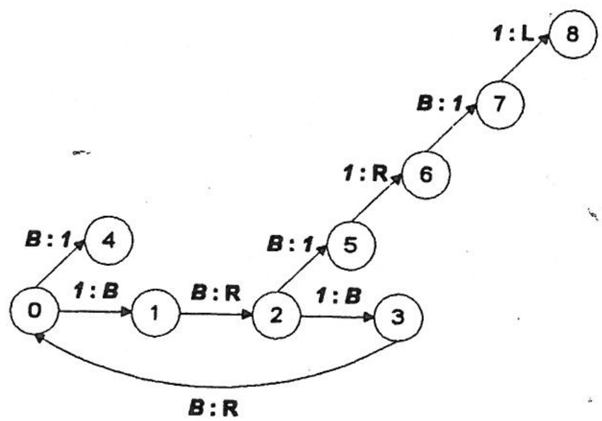


(b) A minimum of *length*(*ababbaab*)=8 states is required.

Solutions to Exercises for § 1.3

1.3.1 [hwk] (a) The halting configuration is $B{*}{*}{*}{*}a{*}q_3B$. This is in accordance with the specification: we do have equality initially and hence M does not halt in configuration Bq_81B.
(b) The final configuration is $B{*}{*}{*}bbq_4B$.

1.3.2.

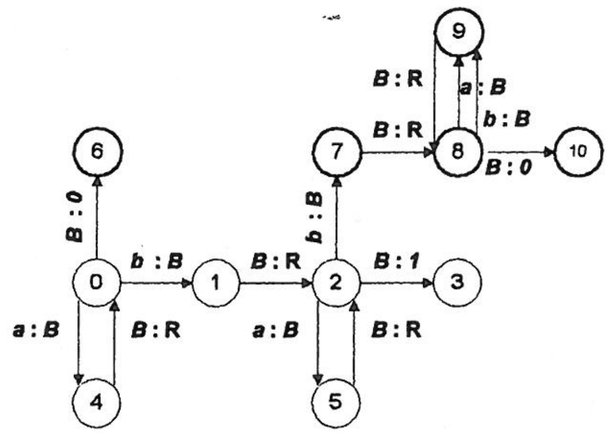


1.3.3. A solution to this exercise can be found by clicking twice on the icon labeled **Convert-to-Unary** within t **Turing** group. Our Turing machine M reads and erases the input string $B(n)$, one digit at a time starting from t left. M constructs the unary representation of $B(n)$ to the right of $B(n)$ and separated from $B(n)$ by one or mo blanks. Each of (1) and (2) of the hint suggest an easy modification of the Copying Machine of Figure 1.3.3: . reads the current unary representation and writes exactly two *1*s for each *1* read.

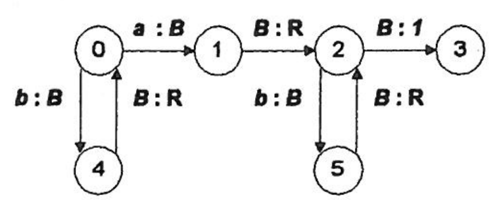
Solutions to Exercises for § 1.4

1.4.1 hwk (a) Word b is the shortest accepted word.
(b) Word aba is accepted but word $abab$ is not.
(c) The language accepted is $\{a^n b a^m | n,m \geq 0\}$.

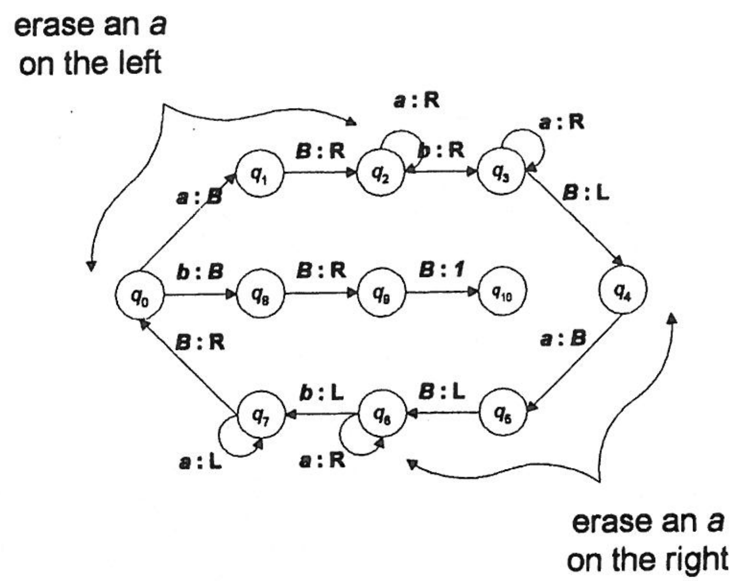
1.4.2. hwk



1.4.3 (a) Starting on the left, the Turing machine whose state diagram appears below deletes 0 or more bs, then a single a, then 0 or more bs, after which it writes a single 1 and halts.



(c) The Turing machine whose state diagram appears below deletes a single a on the far left and then a single a on the far right before returning to the left. This cycle is repeated, if possible, until there are no more as on the left, at which point Turing machine deletes a single b, writes a 1, and then halts.



1.4.4 (a) Compare the solution to Exercise 1.4.3(c). Turing machine M deletes a single a on the far left and then a two as on the far right before returning to the left. This cycle is repeated, if possible, until there are no more as on the left, at which point M deletes a single b, writes a 1, and then halts.

(b) Compare the solutions to Exercises 1.4.3(c) and (a). Turing machine M deletes a single a on the far left and then a pair ab on the far right before returning to the left. (By deletion of a pair ab we, of course, mean deleting a single a, then moving to the right one square and deleting a single b.) This cycle is repeated, if possible, until there are no more as on the left, at which point M deletes a single b, writes a 1, and then halts.

1.4.7 (a) Suppose that Turing machine M recognizes L. By replacing every occurrence of 1 by 0 and vice versa within the arc labels of the state diagram of M, one obtains the state diagram of a new deterministic machine M' that recognizes not L but, rather, L^c. So L^c is also Turing-recognizable. Essentially the same argument can be used to show that if L^c is Turing-recognizable, then so is L.

(b) Suppose that Turing machine M recognizes L. Then M itself also accepts L, so that L is Turing-acceptable. By (a), L^c is Turing-recognizable since L is. So L^c is also Turing-acceptable.

Solutions to Exercises for § 1.5

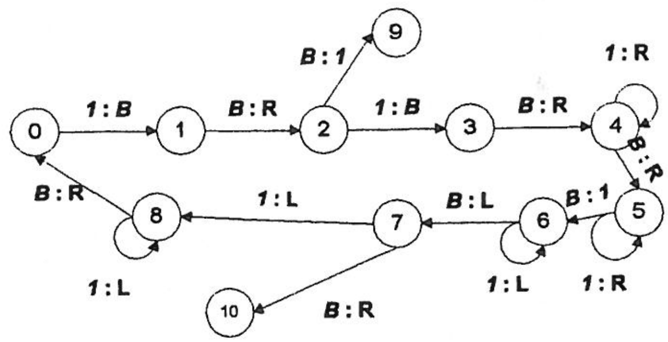
1.5.1 [bwk] (a) $f(n)=3n$
(b) $f(n)=\log_2 n$

1.5.2 The solution to Exercise 1.3.2 fails to recognize that one of the 1s on the tape initially is representational. We remove it by adding two new states, $q_{0'}$ and $q_{0''}$, together with two instructions

$$\delta(q_{0'},1)=(B,q_{0''})$$
$$\delta(q_{0''},B)=(R,q_0)$$

$q_{0'}$, rather than q_0, is the start state of the new machine.

1.5.3 (d)



1.5.5 (a) If M writes the required six *1*s from right to left, then only six states are necessary. (Seven states are required if the *1*s are written from left to right since the read/write head must halt scanning the leftmost *1*.)

(b) Minimum: $m+1$ states

Solutions to Exercises for § 1.6

1.6.2(e) Very roughly, M might do the following. In workspace off to the right, M produces in succession the powers of 2 beginning with $2^0=1$ and continuing 2, 4, 8, 16, 32, ... Producing each member of this sequence after the first is a matter of doubling its predecessor in the sequence. Each power of 2 produced is used to erase an equal number of *1*s from (what remains of) the initially given representation of argument n. With each doubling on the right, M increments a "counter" on the left. This process of doubling-erasing-incrementing continues until M "recognizes" that the number of *1*s remaining is fewer than the current power of 2. At this point, the counter on the right is, in essence, a representation of $\lfloor \log_2 n \rfloor$.

Solutions to Exercises for § 1.7

1.7.1 (a) $time_M(n)$ is $O(n^2)$.
(b) $space_M(n)$ is $O(n)$.

1.7.2. Suppose that M is started scanning word $w=aaba$. Thus the initial tape configuration is

$$\mathbf{B}aaba$$

After entering state q_1, M's reading head is now positioned at the end of w:

$$aabq_1a$$

Since an a is currently being read, the machine proceeds through state q_2 to state q_8 and back to state q_1 and, in the process, replaces this a with an asterisk and writes an a off to the right before resetting: $aaq_1b{*}Ba$. Three more cycles around the loop from q_1 and back produce the configurations $aq_1a{*}{*}Bab$, $q_1a{*}{*}{*}Baba$, and $q_1B{*}{*}{*}{*}Babaa$, respectively. Now the machine has only to erase the four asterisks (states q_9, q_{10}, and q_{11}) and position its read/write head over the leftmost symbol of w^R.

We give a time analysis of the reversal algorithm here. Again, we let n be the length of the input word w. We note the following:

(1) Arriving in state q_1 for the first time requires $n+1$ steps.
(2) The loop from state q_1 through q_8 and back causes one character of w to be replaced by asterisk and then to be copied on the right. Thus the machine traverses the loop from state and back—either via state q_2 or state q_3—exactly n times. Moreover, while the number of st involved in traversing this loop will vary depending upon the position of the character currer under consideration, we can easily give an upper bound on this number. First, notice that number of steps increases as the processed character moves to the front of w. Thus the grea number of steps will occur when the character being copied is the very first character of w. If character being copied is an a, then the number of steps around the loop is summarized in following table.

Number of steps	Action
2	replace a with * and then move one square to the right (states q_1 to q_4)
n-1	move over the n-1 asterisks written previously (q_4)
1	move past the separation blank (q_4 to q_6)
n-1	move over the n-1 characters of w copied previously (q_6)
1	write an a (q_6 to q_8)
n	move to the left over w^R (q_8)
1	move past separation blank (q_8 to q_1)
n	move to the left over n asterisks
4n+3	**Total**

(3) Removing each asterisk (state q_9 and back) requires two steps and this must be done n ti Thus the number of steps required after leaving state q_1 for the last time is $1+2n+1$.

This gives $time_M(n) = (n+1) + n\cdot(4n+3) + (1+2n+1) = 4n^2 + 6n + 3$. Apparently $time_M(n) \leq 5n$ sufficiently large n. Hence $time_M(n)$ is $O(n^2)$.

(b) $space_M(n)$ is $O(n)$.

1.7.3. Suppose that M computes in time $O(f(n))$ with $f(n)\geq 1$. Even if M's every move consisted of a move rig say, it could visit no more than $O(f(n))$ tape squares in $O(f(n))$ time. It follows that M requires at most $O(f(n))$ space. We need the assumption that $f(n)\geq 1$ since $space_M(n)\geq 1$ generally, as was pointed out in the text (see th discussion following Definition 1.9).

1.7.4. Assume that M starts scanning the leftmost *1* in the representation of some natural number n. Th executes exactly two steps no matter what n happens to be. Hence the number of computation steps exe depends in no way upon n, which is just what one expresses by saying that M computes in time $O(1)$.

Solutions to Selected Exercises for § 1.8

1.8.1. hwk

Turing machine M_1 accepts $L=\{\varepsilon, ab, aba, abb\}$.
Turing machine M_2 accepts the language $\{(ab)^n b \mid n\geq 0\}$.
Turing machine M_3 accepts the language $\{(ab)^n \mid n\geq 1\}$.
Turing machine M_4 accepts $L=\{aw \mid w\in\Sigma^*\}$.
Turing machine M_5 accepts $L=\{w : |w| \text{ is even}\}$.
Turing machine M_6 accepts $L=\{wb \mid w\in\Sigma^*\}$.

1.8.2. (d) Turing machine M_1 computes the unary constant-1 function C_1^1. Thus, $f(n)=C_1^1(n)$.

(e) Turing machine M_2 computes the number-theoretic function defined by

$$f(n) = \begin{cases} 0 & \text{if } n=0 \\ n-1 & \text{otherwise} \end{cases}$$

This function is frequently referred to as *pred*(n).

1.8.3. One machine that works is defined completely by writing

$$\delta(q_0,B)=(1,q_1)$$

This is one of many possible answers. The important thing is that, if the machine starts scanning a blank (which means that the "input" word is the empty word), then it writes a *1* and halts. On the other hand, consider a machine that reads a single character—either an *a* or a *b*—overwrites it with a *1* and then halts. Such a machine accepts the two words *a* and *b* and, hence, is not a solution.

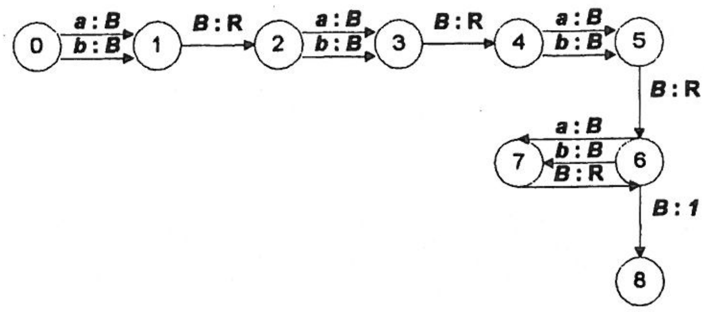
1.8.5. hwk The machine should replace a single *a*, if there is one, by an asterisk ("*"), say, and then move to the right and replace three *b*s, and then move to the right and replace two *c*s. Now it should move left, passing over all tape symbols until it finds the beginning of the string. It should repeat this process until (i) having replaced two *c*s, there are only asterisks on the tape, in which case they should all be erased and a single *1* written; (ii) having replaced an *a* the machine can find no *b*s or *c*s (or at least an insufficient number of them), in which case it can merely halt; or (iii) having replaced two *c*s, there remain no *a*s, but the tape is not simply a string of asterisks either. In the latter case, the machine can merely halt.

1.8.6. hwk

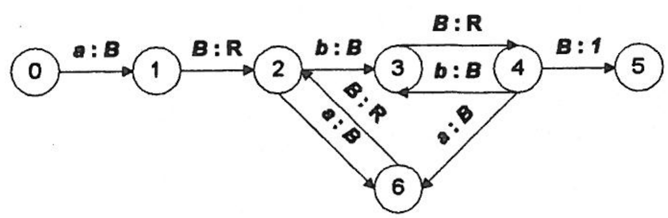
(a) 59
(b) 107

1.8.7 (a) number-theoretic
(b) language
ε
(c) *aba* and *ababa*
(d) $O(n^2)$

1.8.8. hwk



1.8.9. hwk



1.8.10. Suppose that f and g are two unary, total number-theoretic functions that are computed by single-tape Turing machines M_f and M_g, in $O(p_f(n))$ steps and in $O(p_g(n))$ steps, respectively, where p_f and p_g are polynomials in n. We describe a new single-tape Turing machine M^* that computes function $f \circ g$ in polynomial time. First, M^* simulates M_g on input n. By assumption, this simulation requires $O(p_g(n))$ computation steps. The result is value $g(n)$, which must itself be $O(p_g(n))$. (Why?) M^* next simulates M_f on $g(n)$ so as to obtain $f(g(n))=f \circ g(n)$ in $O(p_f(p_g(n)))$ steps. But $p_f(p_g(n))$ is a polynomial in n since both $p_f(n)$ and $p_g(n)$ are. So M^* computes $f \circ g$ in $O(p_g(n))$ steps plus $O(p_f(p_g(n)))$ steps. Hence $f \circ g$ is polynomial-time Turing-computable.

Note that we have used the fact that the coefficient of the highest-degree term of p_f must be positive, so that p_f itself will be monotone increasing. Further, by Exercise 0.5.13, we may assume that

$$g(n) \leq p_g(n)$$

for sufficiently large n. Similarly, we may assume that M^*'s simulation of M_f on $g(n)$ takes no more than $p_f(g(n))$ steps. It then follows that, for sufficiently large n, we also have

$$\begin{aligned} time_{M^*}(n) &\leq p_g(n) + p_f(g(n)) \\ &\leq p_g(n) + p_f(p_g(n)) \end{aligned}$$

Solutions to Selected Exercises for Chapter 2

Solutions to Exercises for § 2.1

2.1.4. Click on icon **Palindromes** within the Turing group. That this machine M computes in space $O(n)$ is obvious: only one tape square to the left and one tape square to the right of the input word w is ever scanned. Where $|w|=n$, it follows that $n+2$, which is $O(n)$, tape squares are visited. As for time, M traverses the cycle from state q_2 to state q_{11} and back $\lfloor n/2 \rfloor$ or $O(n)$ times, worst case. Each such traversal involves no more than about $2n$, which is $O(n)$, steps. Consequently, the total number of steps for all complete traversals is $O(n^2)$. The handling of a possible middle symbol, in the case of an odd-length palindrome, and the ultimate writing of an accepting *1*