Introduction to Automata Theory, Languages, and Computation

Solutions for Chapter 8

Solutions for Section 8.1

Exercise 8.1.1(a)

We need to take a program *P* and modify it so it:

1. Never halts unless we explicitly want it to, and
2. Halts whenever it prints `hello, world.`

For (1), we can add a loop such as `while(1){x=x;}` to the end of `main`, and also at any point where `main` returns. That change catches the normal ways a program can halt, although it doesn't address the problem of a program that halts because some exception such as division by 0 or an attempt to read an unavailable device. Technically, we'd have to replace all of the exception handlers in the run-time environment to cause a loop whenever an exception occurred.

For (2), we modify *P* to record in an array the first 12 characters printed. If we find that they are `hello, world.`, we halt by going to the end of `main` (past the point where the while-loop has been installed).

Solutions for Section 8.2

Exercise 8.2.1(a)

To make the ID's clearer in HTML, we'll use *[q0]* for *q_0*, and similarly for the other states.

[q0]00 |- X[q1]0 |- X0[q1]

The TM halts at the above ID.

Exercise 8.2.2(a)

Here is the transition table for the TM:

state	0	1	B	X	Y

q0	(q2,X,R)	(q1,X,R)	(qf,B,R)	-	(q0,Y,R)
q1	(q3,Y,L)	(q1,1,R)	-	-	(q1,Y,R)
q2	(q2,0,R)	(q3,Y,L)	-	-	(q2,Y,R)
q3	(q3,0,L)	(q3,1,L)	-	(q0,X,R)	(q3,Y,L)
qf	-	-	-	-	-

In explanation, the TM makes repeated excursions back and forth along the tape. The symbols *X* and *Y* are used to replace 0's and 1's that have been cancelled one against another. The difference is that an *X* guarantees that there are no unmatched 0's and 1's to its left (so the head never moves left of an *X*), while a *Y* may have 0's or 1's to its left.

Initially in state *q0*, the TM picks up a 0 or 1, remembering it in its state (*q1* = found a 1; *q2* = found a 0), and cancels what it found with an *X*. As an exception, if the TM sees the blank in state *q0*, then all 0's and 1's have matched, so the input is accepted by going to state *qf*.

In state *q1*, the TM moves right, looking for a 0. If it finds it, the 0 is replaced by *Y*, and the TM enters state *q3* to move left an look for an *X*. Similarly, state *q2* looks for a 1 to match against a 0.

In state *q3*, the TM moves left until it finds the rightmost *X*. At that point, it enters state *q0* again, moving right over *Y*'s until it finds a 0, 1, or blank, and the cycle begins again.

Exercise 8.2.4

These constructions, while they can be carried out using the basic model of a TM are much clearer if we use some of the tricks of Sect. 8.3.

For part (a), given an input *[x,y]* use a second track to simulate the TM for *f* on the input *x*. When the TM halts, compare what it has written with *y*, to see if *y* is indeed *f(x)*. Accept if so.

For part (b), given *x* on the tape, we need to simulate the TM *M* that recognizes the graph of *f*. However, since this TM may not halt on some inputs, we cannot simply try all *[x,i}* to see which value of *i* leads to acceptance by *M*. The reason is that, should we work on some value of *i* for which *M* does not halt, we'll never advance to the correct value of *f(x)*. Rather, we consider, for various combinations of *i* and *j*, whether *M* accepts *[x,i]* in *j* steps. If we consider *(i,j)* pairs in order of their sum (i.e., (0,1), (1,0), (0,2), (1,1), (2,0), (0,3),...) then eventually we shall simulate *M* on *[x,f(x)]* for a sufficient number of steps that *M* reaches acceptance. We need only wait until we consider pairs whose sum is *f(x)* plus however many steps it takes *M* to accept *[x,f(x)]*. In this manner, we can discover what *f(x)* is, write it on the tape of the TM that we have designed to compute *f(x)*, and halt.

Now let us consider what happens if *f* is not defined for some arguments. Part (b) does not change, although the constructed TM will fail to discover *f(x)* and thus will continue searching forever. For part (a), if we are given *[x,y]*, and *f* is not defined on *x*, then the TM for *f* will never halt on *x*. However, there is nothing wrong with that. Since *f(x)* is undefined, surely *y* is not *f(x)*. Thus, we do not want the TM for the graph of *f* to accept *[x,y]* anyway.

Exercise 8.2.5(a)

This TM only moves right on its input. Moreover, it can only move right if it sees alternating 010101... on the input tape. Further, it alternates between states *q_0* and *q_1* and only accepts if it sees a blank in state *q_1*. That in turn occurs if it has just seen 0 and moved right, so the input must end in a 0. That is, the language is that of regular expression **(01)*0**.

Solutions for Section 8.3

Exercise 8.3.3

Here is the subroutine. Note that because of the technical requirements of the subroutine, and the fact that a TM is not allowed to keep its head stationary, when we see a non-0, we must enter state *q3*, move right, and then come back left in state *q4*, which is the ending state for the subroutine.

state	0	1	B
q1	(q2,0,R)	-	-
q2	(q2,0,R)	(q3,1,R)	(q3,B,R)
q3	(q4,0,L)	(q4,1,L)	(q4,B,L)

Now, we can use this subroutine in a TM that starts in state *q0*. If this TM ever sees the blank, it accepts in state *qf*. However, whenever it is in state *q0*, it knows only that it has not seen a 1 immediately to its right. If it is scanning a 0, it must check (in state *q5*) that it does not have a blank immediately to its right; if it does, it accepts. If it sees 0 in state *q5*, it comes back to the previous 0 and calls the subroutine to skip to the next non-0. If it sees 1 in state *q5*, then it has seen 01, and uses state *q6* to check that it doesn't have another 1 to the right.

In addition, the TM in state *q4* (the final state of the subroutine), accepts if it has reached a blank, and if it has reached a 1 enters state *q6* to make sure there is a 0 or blank following. Note that states *q4* and *q5* are really the same, except that in *q4* we are certain we are not scanning a 0. They could be combined into one state. Notice also that the subroutine is not a perfect match for what is needed, and there is some unnecessary jumping back and forth on the tape. Here is the remainder of the transition table.

state	0	1	B
q0	(q5,0,R)	(q5,1,R)	(qf,B,R)
q5	(q1,0,L)	(q6,1,R)	(qf,B,R)
q6	(q0,0,R)	-	(qf,B,R)
q4	-	(q6,1,R)	(qf,B,R)

Solutions for Section 8.4

Exercise 8.4.2(a)

For clarity, we put the state in square brackets below. Notice that in this example, we never branch. *[q_0]01 |- 1[q_0]1 |- 10[q_1] |- 10B[q_2]*

Exercise 8.4.3(a)

We'll use a second tape, on which the guess *x* is stored. Scan the input from left yo right, and at each cell, guess whether to stay in the initial state (which does the scanning) or go to a new state that copies the next 100 symbols onto the second tape. The copying is done by a sequence of 100 state, so exactly 100 symbols can be placed on tape 2.

Once the copying is done, retract the head of tape 2 to the left end of the 100 symbols. Then, continue moving right on tape 1, and at each cell guess either to continue moving right or to guess that the second copy of *x* begins. In the latter case, compare the next 100 symbols on tape 1 with the 100 symbols on tape 2. If they all match, then move right on tape 1 and accept as soon as a blank is seen.

Exercise 8.4.5

For part (a), guess whether to move left or right, entering one of two different states, each responsible for moving in one direction. Each of these states proceeds in its direction, left or right, and if it sees a $ it enters state *p*. Technically, the head has to move off the $, entering another state, and then move back to the $, entering state *p* as it does so.

Part (b), doing the same thing deterministically, is trickier, since we might start off in the wrong direction and travel forever, never seeing the $. Thus, we have to oscillate, using left and right endmarkers *X* and *Y*, respectively, to mark how far we have traveled on a second track. Start moving one cell left and leave the *X*. Then, move two cells right and leave the *Y*. Repeatedly move left to the *X*, move the *X* one more cell left, go right to the *Y*, move it once cell right, and repeat.

Eventually, we shall see the $. At this time, we can move left or right to the other endmarker, erase it, move back to the end where the $ was found, erase the other endmarker, and wind up at the $.

Exercise 8.4.8(a)

There would be 10 tracks. Five of the tracks hold one of the symbols from the tape alphabet, so there are 7^5 ways to select these tracks. The other five tracks hold either *X* or blank, so these tracks can be selected in 2^5 ways. The total number of symbols is thus 7^5 * 2^5 = 537,824.

Exercise 8.4.8(b)

The number of symbols is the same. The five tracks with tape symbols can still be chosen in 5^7 ways. The sixth track has to tell which subset of the five tapes have their head at that position. There are 2^5 possible subsets, and therefore 32 symbols are needed for the 6th track. Again the number of symbols is 7^5 * 2^5.

Solutions for Section 8.5

Exercise 8.5.1(c)

In principle, any language that is recursively enumerable can be recognized by a 2-counter machine, but how do we design a comprehensible answer for a particular case? As the *a*'s are read, count them with both counters. Then, when *b*'s enter, compare them with one counter, and accept if they are the same. Continue accepting as long as *c*'s enter. If the numbers of *a*'s and *b*'s differ, then compare the second counter with the number of *c*'s, and accept if they match.