Jason Downing
Email: jason_downing@student.uml.edu
Foundations of Computer Science
Homework #4 - Chapter 3
11/20/2016

****

I was unable to complete a large portion of 3.8, including the state diagrams and the configurations for those state diagrams. To make up for these missing problems I decided to do as much **Extra Credit** as I could. As a result, I did the following **Extra Credit** problems:

1. 3.1 B
2. 3.2 A
3. 3.4
4. 3.5 A
5. 3.5 B
6. 3.5 C
7. 3.5 D
8. 3.6
9. 3.7
10. 3.8 A
11. 3.10
12. 3.11

I figured that I missed roughly 12 problems, so I should do up to 12 problems worth of Extra Credit. I hope that this is enough extra work to counter the missing problems that I was unable to solve before the deadline. I have marked all **Extra Credit** in bold and I have put them at the end of my PDF in a clearly marked "**Extra Credit**" section.

****

3.1 This exercise concerns $TM M_2$, whose description and state diagram appear in Example 3.7. In each of the parts, give the sequence of configurations that M_2 enters when started on the indicated input string.

a. 0.

$q_1 0$ is the starting state.
Running the input 0 on the machine M_2, we get the following configuration:

$q_1 0$
$\sqcup q_2 \sqcup$
$\sqcup \sqcup q_{accept}$
M_2 enters into the q_{accept} state, and as a result the input is accepted.

c. 000.
$q_1 000$ is the starting state.
Running the input 000 on the machine M_2, we get the following configuration:

$q_1 000$
$\sqcup q_2 00$
$\sqcup x q_3 0$
$\sqcup x 0 q_4$
$\sqcup x 0 q_{reject}$
M_2 enters into the q_{reject} state, and as a result the input is rejected.

d. 000000.
$q_1 000000$ is the starting state.
Running the input 000000 on the machine M_2, we get the following config:

$q_1 000000$
$\sqcup q_2 00000$
$\sqcup x q_3 0000$
$\sqcup x 0 q_4 000$
$\sqcup x 0 x q_3 00$
$\sqcup x 0 x 0 q_4 0$
$\sqcup x 0 x 0 x q_3 \sqcup$
$\sqcup x 0 x 0 q_5 x \sqcup$
$\sqcup x 0 x q_5 0 x \sqcup$
$\sqcup x 0 q_5 x 0 x \sqcup$
$\sqcup x q_5 0 x 0 x \sqcup$
$\sqcup q_5 x 0 x 0 x \sqcup$
$q_5 \sqcup x 0 x 0 x \sqcup$
$\sqcup q_2 x 0 x 0 x \sqcup$
$\sqcup x q_2 0 x 0 x \sqcup$
$\sqcup x x q_3 x 0 x \sqcup$
$\sqcup x x x q_3 0 x \sqcup$
$\sqcup x x x 0 q_4 x \sqcup$
$\sqcup x x x 0 x q_4 \sqcup$
$\sqcup x x x 0 x \sqcup q_{reject}$
M_2 enters into the q_{reject} state, and as a result the input is rejected.

Plus 00000000.

$q_1 00000000$ is the starting state.
Running the input 000000 on the machine M_2, we get the following configuration:

$q_1 00000000$
$\sqcup q_2 0000000$
$\sqcup x q_3 000000$
$\sqcup x 0 q_4 00000$
$\sqcup x 0 x q_3 0000$
$\sqcup x 0 x 0 q_4 000$
$\sqcup x 0 x 0 x q_3 00$
$\sqcup x 0 x 0 x 0 q_4 0$
$\sqcup x 0 x 0 x 0 x q_3 \sqcup$
$\sqcup x 0 x 0 x 0 q_5 x \sqcup$
$\sqcup x 0 x 0 x q_5 0 x \sqcup$
$\sqcup x 0 x 0 q_5 x 0 x \sqcup$
$\sqcup x 0 x q_5 0 x 0 x \sqcup$
$\sqcup x 0 q_5 x 0 x 0 x \sqcup$
$\sqcup x q_5 0 x 0 x 0 x \sqcup$
$\sqcup q_5 x 0 x 0 x 0 x \sqcup$
$q_5 \sqcup x 0 x 0 x 0 x \sqcup$
$\sqcup q_2 x 0 x 0 x 0 x \sqcup$
$\sqcup x q_2 0 x 0 x 0 x \sqcup$
$\sqcup x x q_3 x 0 x 0 x \sqcup$
$\sqcup x x x q_3 0 x 0 x \sqcup$
$\sqcup x x x 0 q_4 x 0 x \sqcup$
$\sqcup x x x 0 x q_4 0 x \sqcup$
$\sqcup x x x 0 x x q_3 x \sqcup$
$\sqcup x x x 0 x x x q_3 \sqcup$
$\sqcup x x x 0 x x q_5 x \sqcup$
$\sqcup x x x 0 x q_5 x x \sqcup$
$\sqcup x x x 0 q_5 x x x \sqcup$
$\sqcup x x x q_5 x x x 0 \sqcup$
$\sqcup x x q_5 x x x 0 x \sqcup$
$\sqcup x q_5 x x x 0 x x \sqcup$
$\sqcup q_5 x x x 0 x x x \sqcup$

$q_5 \sqcup xxx0xxx \sqcup$
$\sqcup q_2 xxx0xxx \sqcup$
$\sqcup x q_2 xx0xxx \sqcup$
$\sqcup xx q_2 x0xxx \sqcup$
$\sqcup xxx q_2 0xxx \sqcup$
$\sqcup xxxx q_3 xxx \sqcup$
$\sqcup xxxxx q_3 xx \sqcup$
$\sqcup xxxxxx q_3 x \sqcup$
$\sqcup xxxxxxx q_3 \sqcup$
$\sqcup xxxxxx q_5 x \sqcup$
$\sqcup xxxxx q_5 xx \sqcup$
$\sqcup xxxx q_5 xxx \sqcup$
$\sqcup xxx q_5 xxxx \sqcup$
$\sqcup xx q_5 xxxxx \sqcup$
$\sqcup x q_5 xxxxxx \sqcup$
$\sqcup q_5 xxxxxxx \sqcup$
$q_5 \sqcup xxxxxxx \sqcup$
$\sqcup q_2 xxxxxxx \sqcup$
$\sqcup x q_2 xxxxxx \sqcup$
$\sqcup xx q_2 xxxxx \sqcup$
$\sqcup xxx q_2 xxxx \sqcup$
$\sqcup xxxx q_2 xxx \sqcup$
$\sqcup xxxxx q_2 xx \sqcup$
$\sqcup xxxxxx q_2 x \sqcup$
$\sqcup xxxxxxx q_2 \sqcup$
$\sqcup xxxxxxx q_{accept} \sqcup$
M_2 enters into the q_{accept} state, and as a result the input is accepted.

3.2 This exercise concerns $TM M_1$, whose description and state diagram appear in Example 3.9. In each of the parts, give the sequence of configurations that M_1 enters when started on the indicated input string.

b. 1#1.

Input is 1#1. Starting state is $q_1 1\#1$.
Running 1#1 on the machine M_1 results in the following configuration:

$q_1 1\#1$
$xq_3\#1$
$x\#q_5 1$
$x\#q_6 x$
$q_7 x\#x$
$xq_1\#x$
$x\#q_8 x$
$x\#xq_9 \sqcup$
$x\#x\sqcup q_{accept}$
M_1 enters into the q_{accept} state, and as a result the input is accepted.

c. 1##1

Input is 1##1. Starting state is $q_1 1\#\#1$.
Running 1##1 on the machine M_1 results in the following configuration:

$q_1 1\#\#1$
$xq_3\#\#1$
$x\#q_5\#1$ *(At this point q_5 does not read the #, so it enters the reject state)*
$x\#\#q_{reject}1$
M_1 enters into the q_{reject} state, and as a result the input is rejected.

d. 10#11.

Input is 10#11. Starting state is $q_1$10#11.
Running 10#11 on the machine M_1 results in the following configuration:

$q_1 10\#11$
$xq_3 0\#11$
$x0q_3\#11$
$x0\#q_5 11$
$x0q_6\#x1$
$xq_7 0\#x1$
$q_7 x0\#x1$
$xq_1 0\#x1$
$xxq_2\#x1$
$xx\#q_4 x1$
$xx\#xq_4 1$ *(At this point q_4 does not read the 1, so it enters the reject state)*
$xx\#x1q_{reject}$
M_1 enters into the q_{reject} state, and as a result the input is rejected.

e. 10#10.

Input is 10#10. Starting state is $q_1$10#10.
Running 10#10 on the machine M_1 results in the following configuration:

$q_1 10\#10$
$xq_3 0\#10$
$x0q_3\#10$
$x0\#q_5 10$
$x0q_6\#x0$
$xq_7 0\#x0$
$q_7 x0\#x0$
$xq_1 0\#x0$
$xxq_2\#x0$
$xx\#q_4 x0$
$xx\#xq_4 0$
$xx\#q_6 xx$
$xxq_6\#xx$
$xq_7 x\#xx$
$xxq_1\#xx$
$xx\#q_8 xx$
Now right shift q_8 until all x's have been read
$xx\#xxq_8 \sqcup$
$xx\#xx\sqcup q_{accept}\sqcup$
M_1 enters into the q_{accept} state, and as a result the input is accepted.

Plus: 01100#01100

Input is 01100#01100. Starting state is $q_1$01100#0110.
Running 01100#01100 on the machine M_1 results in the following config:

$q_1 01100\#01100$
$xq_2 1100\#01100$
Right shift (0,1) until we hit a #.
$x1100q_2\#01100$
$x1100\#q_4 01100$
$x1100\#xq_6 1100$
$x1100\#q_6 x1100$
$x1100q_7\#x1100$
Left shift (0,1) until we hit a x.
$xq_7 1100\#x1100$
$x1q_1 100\#x1100$
$x1xq_3 00\#x1100$
Right shift (0,1) until we hit a #.
$x1x00q_3\#x1100$
$x1x00\#q_5 x1100$
$x1x00\#xq_5 1100$
$x1x00\#xxq_6 100$
Left shift (0,1,x) until we hit a #.
$x1x00\#q_6 xx100$
$x1x00q_7\#xx100$
Left shift (0,1) until we hit a x.
$x1xq_7 00\#xx100$
$x1q_1 x00\#xx100$
$xxxq_3 00\#xx100$
Right shift (0,1) until we hit a #.
$xxx00q_3\#xx100$
$xxx00\#q_5 xx100$
Right shift (x) until we hit a 1. $xxx00\#xxq_5 100$
$xxx00\#xq_6 xx00$
$xxx00\#q_6 xxx00$
$xxx00q_7\#xxx00$
Left shift (0,1) until we hit a x.
$xxxq_7 00\#xxx00$

$xxx0q_10\#xxx00$
$xxxx0q_2\#xxx00$
$xxxx0\#q_4xxx00$
$xxxx0\#xq_4xx00$
Right shift (x) until we hit a 0.
$xxxx0\#xxxq_400$
$xxxx0\#xxq_6xx0$
Left shift (0,1,x) until we hit a #.
$xxxx0\#q_6xxxx0$
$xxxx0q_7\#xxxx0$
$xxxxq_70\#xxxx0$
$xxxx0q_1\#xxxx0$
$xxxxx\#q_2xxxx0$
$xxxxx\#xq_4xxx0$
Right shift until we hit a 0.
$xxxxx\#xxxxq_40$
$xxxxx\#xxxq_6xx$
Left shift (0,1,x) until we hit a #.
$xxxxx\#q_6xxxxx$
$xxxxxq_7\#xxxxx$
$xxxxx\#q_1xxxxx$
$xxxxx\#xq_8xxxx$
Right shift all the x's.
$xxxxx\#xxxxxq_8\sqcup$
$xxxxx\#xxxxx\sqcup q_{accept}\sqcup$
M_1 enters into the q_{accept} state, and as a result the input is accepted.

Plus: 01101#01100

Input is 01101#01100. Starting state is $q_1$01101#01100.
Running 01101#01100 on the machine M_1 results in the following config:

$q_1 01101\#01100$
$x q_2 1101\#01100$
Right shift (0,1) until we hit a #
$x1101 q_2 \#01100$
$x1101\# q_4 01100$
$x1101 q_6 \# x1100$
$x110 q_7 1\# x1100$
Left shift (0,1) until we hit a x
$x q_7 1101\# x1100$
$x1 q_1 101\# x1100$
$xx1 q_3 01\# x1100$
Right shift (0,1) until we hit a #
$xx101 q_3 \# x1100$
$xx101\# q_5 x1100$
$xx101\# x q_5 1100$
$xx101\# q_6 xx100$
$xx101 q_7 \# xx100$
Left shift (0,1) until we hit a x
$xx q_7 101\# xx100$
$xx1 q_1 01\# xx100$
$xxx0 q_3 1\# xx100$
$xxx01 q_3 \# xx100$
$xxx01\# q_5 xx100$
Right shift (x) until we hit a 1.
$xxx01\# xx q_5 100$
$xxx01\# x q_6 xx00$
$xxx01\# q_6 xxx00$
$xxx01 q_7 \# xxx00$
Left shift (0,1) until we hit a x
$xxx q_7 01\# xxx00$
$xxx0 q_1 1\# xxx00$
$xxxx1 q_2 \# xxx00$
$xxxx1\# q_4 xxx00$

Right shift (x) until we hit a 0.
$xxxx1\#xxxq_400$
$xxxx1\#xxxxq_60$
Left shift (0, 1, x) until we hit #
$xxxx1\#q_6xxxx0$
$xxxx1q_7\#xxxx0$
$xxxxq_71\#xxxx0$
$xxxx1q_1\#xxxx0$
$xxxxx\#q_3xxxx0$
$xxxxxq_5\#xxxx0$
Right shift (x) until we hit a 1. $xxxxx\#xxxxq_50$
At this point the machine will go into the reject state.
The reason for this is that it cannot read a 1, or an x.
$xxxxx\#xxxx0q_{reject}$
M_1 enters into the q_{reject} state, and as a result the input is rejected.

The reason for this failure is that at q_5, the machine expects a 1 but it gets a 0. There are no paths for 0 at q_5, and as a result the machine will go into the reject state as it is unable to continue.

3.8 Give implementation-level descriptions of Turing machines that decide the following languages over the alphabet $\{0, 1\}$.
b. {w | w contains twice as many 0s as 1s}

For input string w, we would do the following implementation:
1. We first scan the tape and mark the first 0 that has not been marked yet. If we find no unmarked 0's, we then continue to #4.
2. We move onto mark the next unmarked 0. If we do not find any on the tape, we enter the reject state. Otherwise, we move back to the front of the tape.
3. We scan the tape and mark the first 1 which has not been marked yet. If there is no unmarked 1, we enter the reject state.
4. We now move the head back to the front of the tape, and repeat #1.
5. We move the head of the tape back to the front of the tape, and we then scan the tape to see if we can find any unmarked 1's. If there are none, we enter the accept state. Otherwise we enter the reject state.

c. {w | w does not contain twice as many 0s as 1s}

For input string w, we would do the following implementation:
1. We first scan the tape and mark the first 0 which has not yet been marked. If we find no unmarked 0, we go to #4.
2. We continue moving and mark the next unmarked 0. If we do not find any on the tape, then we enter the accept state. Otherwise, we move the head of the tape back to the front and we continue.
3. We scan the tape and mark the first 1 which has not yet been marked. If there are no unmarked 1's, we enter the accept state.
4. We move the head back to the front of the tape, and we repeat #1.
5. We move the head back to the front of the tape, and we scan the tape to see if there are any unmarked 1's. If there are no unmarked 1's, we enter the reject state. Otherwise, we enter the accept state.

****

EXTRA CREDIT SECTION BEGINS HERE

****

3.1 b. 00. (**EXTRA CREDIT**)

$q_1 00$ is the starting state.
Running the input 00 on the machine M_2, we get the following configuration:

$q_1 00$
$\sqcup q_2 0$
$\sqcup x q_3 \sqcup$
$\sqcup q_5 x \sqcup$
$q_5 \sqcup x \sqcup$
$\sqcup q_2 x \sqcup$
$\sqcup x q_2 \sqcup$
$\sqcup x \sqcup q_{accept}$

3.2 a. 11 (**EXTRA CREDIT**)

Input is 11. Starting state is $q_1 11$. Config is:

$q_1 11$
$\sqcup q_3 1$
$\sqcup 1 q_3 \sqcup$
$\sqcup 1 \sqcup q_{reject}$
M_1 enters into the q_{reject} state, and as a result the input is rejected.

The reason for this is because there is no path for the machine M_1 to take at q_3 that involves a 1. It cannot go to q_4 without having a 0, and it cannot go to q_5 without having a $\sqcup$ before the q_3.

3.4 (**EXTRA CREDIT**)
Give a formal definition of an enumerator. Consider it to be a type of two-tape Turing machine that uses its second tape as the printer. Include a definition of the enumerated language.

We can define an enumerator as a 7-tuple: (Q, $\sum$, τ, δ, q_0, q_{print}, q_{accept})

Where Q, $\sum$, and τ are finite sets and:
1. Q is the set of states
2. τ is the work tape alphabet
3. $\sum$ is the output tape alphabet
4. $\delta : Q \times \tau \rightarrow Q \times \tau \times \{L, R\} \times \sum_{\epsilon}$ is the transition function
5. $q_0 \epsilon Q$ is the starting state
6. $q_{print} \epsilon Q$ is the printing state
7. $q_{accept} \epsilon Q$ is the rejecting state, which is where $q_{print} \neq q_{reject}$

The computation of this enumerator E is defined as an ordinary Turing Machine (TM), except for a few points. It has two tapes, a print tape and a work tape, which both begin blank initially. During each step, the TM can write a symbol from $\sum$ onto the output tape, or it can write nothing at all, which is determined by δ. If $\delta(q, a) = (r, b, L, c)$, it means that during state q while reading a, the enumerator E enters into state R, and it writes B back onto the work tape. It then moves the head of the working tape left or right depending on whether L was previously R, and it then writes C onto the output tape. If $C \neq \epsilon$, then the head of the output tape moves to the right.

Whenever we enter the state q_{print}, then the output tape is reset to blank, and the head returns to the left side. The machine will halt when q_{accept} is entered. $L(E) = \{w\epsilon \sum^* |w\}$ will be on the work tape if q_{print} is entered.

3.5 (**EXTRA CREDIT**)
Examine the formal definition of a Turing machine to answer the following questions, and explain your reasoning.

a. Can a Turing machine ever write the blank symbol $\sqcup$ on its tape?

Yes, a turing machine can write the blank symbol $\sqcup$ onto its tape. This is because according to the definition of a Turing Machine, a TM may write any characters in τ onto its tape. τ is the tape alphabet, and according to the definition of a TM, $\sqcup \in \tau$.

b. Can the tape alphabet τ be the same as the input alphabet $\sum$?

No, this is not possible because $\sum$ never contains the blank symbol $\sqcup$. However, τ always contains the blank symbol $\sqcup$, and as a result, they can never be equal.

c. Can a Turing machines head *ever* be in the same location in two successive steps?

Yes, a TM can have its head in the same location during two successive steps. This is because if the TM tries to move its head of the left hand side of the tape, it will remain in the same location as the previous step.

d. Can a Turing machine contain just a single state?

No, a TM cannot contain just a single state. Any TM by definition must contain at least two distinct states, q_{accept} and q_{reject}. As a result, a TM must contain at a minimum two states to be considered a TM.

3.6 (**EXTRA CREDIT**)

In Theorem 3.21, we showed that a language is Turing-recognizable iff some enumerator enumerates it. Why didnt we use the following simpler algorithm for the forward direction of the proof? As before, s_1, s_2, ... is a list of all strings in $\sum^*$.

$E =$"Ignore the input.

1. Repeat the following for $i = 1, 2, 3, ...$
2. Run M on s_1
3. If it accepts, print ouf s_1."

The reason we don't use a simpler algorithm during this proof is that in part 2 of the algorithm (Run M on s_1), if M loops on the input s_i, E will not check any inputs after s_i. As a result, if that were to occur, E could fail to enumerate L(M) as it is required to.

3.7 (**EXTRA CREDIT**)

Explain why the following is not a description of a legitimate Turing machine.
$M_{bad} =$ "On input (p), a polynomial over variables x_1, ..., x_k:

1. Try all possible settings of x_1, ..., x_k to integer values.
2. Evaluate p on all of these settings.
3. If any of these settings evaluates to 0, *accept* or otherwise *reject*.

This is not a description of a legit Turing Machine because the variables listed have an infinite amount of possible settings. A TM would as a result require an infinite amount of time to try all possible settings. This is not allowed in a valid TM as we require that each step in a Turing Machine be completed in a finite amount of steps. Requiring an infinite amount of steps as a result makes it not a legitimate Turing Machine.

3.8 a. $\{$w | w contains an equal number of 0's and 1's. (**EXTRA CREDIT**)

For input string w, we would do the following implementation:
1. Scan the tape, and mark the first 0 which has not been marked yet. Now if there are no unmarked 0's, we go to stage #4. Otherwise, we move the head back to the front of the tape.

2. We now scan the tape, and mark the first 1 which has no been marked yet. If we find no unmarked 1's, we enter the $reject$ state.

3. Now move the head back to the front of the tape, and repeat step #1.

4. Now move the head to the front of the tape. We scan the tape to see if we can find any unmarked 1's that are remaining. If we find no unmarked 1's, we enter the *accept* state. Otherwise, we enter the $reject$ state.

3.10 (**EXTRA CREDIT**)

Say that a write-once Turing machine is a single-tape TM that can alter each tape square at most once (including the input portion of the tape). Show that this variant Turing machine model is equivalent to the ordinary Turing machine model. (Hint: As a first step, consider the case whereby the Turing machine may alter each tape square at most twice. Use lots of tape.)

We can simulate an ordinary Turing Machine (TM) by using a write-twice TM. The write-twice TM can simulate a single step of the original TM by copying its entire tape over to a new section of the tape to the right of the currently used section. The copy procedure would be as follows:

• Copy character, by character, marking a character as it is copied.
• Alter the tape square twice, once to write a character for the first time, and a second time to mark that it has been copied.

The position of the original TM's tape head is marked on the tape, and when it copies the positions around the marked position, the tape contents are updated according to the rules of the TM.

To complete the simulation with a write-once TM, we would operate the same as before, except that we would make sure that each cell of the tape is represented by two cells. The first cell would contain the original TM's tape, and the second would be used to mark each character as it is copied over during the copying procedure. The input would not be presented to the machine as there are two cells per symbol, so as a result the first time that the tape is copied the marks are put directly over the input symbols.

3.11 (**EXTRA CREDIT**)

A ***Turing machine with doubly infinite tape*** is similar to an ordinary Turing machine, but its tape is infinite to the left as well as to the right. The tape is initially filled with blanks except for the portion that contains the input. Computation is defined as usual except that the head never encounters an end to the tape as it moves leftward. Show that this type of Turing machine recognizes the class of Turing-recognizable languages.

A Turing Machine (TM) with ***"doubly infinite tape"*** can easily simulate a normal TM. It will need to mark the left side of the input's end, so that it can prevent the head from moving off the end of the tape.

In order for a normal TM to simulate a ***"doubly infinite tape"*** TM, we will need to show how to simulate the TM with a 2 tape TM. This has already been shown to be equal in power to a normal TM. The first tape of this 2 tape TM will be written with the input string, and the second tape will be left blank initially. We will cut the tape of the doubly infinite tape TM into two separate parts, which will be at the starting position of the input string. The input string portion and the blank spaces to its right will appear on the first tape of the 2 tape TM. The portion to the left of the input string will appear on the second tape, but in the reverse order.

Everything else from here on down I was unable to complete.
I left it here for future reference.

Plus: Draw the state diagram for Turning Machines 3.8b and 3.8c
3.8b

3.8c

For these machines draw the configurations for
3.8b:
010100.

010101.

3.8c:
000111.

000110.

Modify machine M2 to recognize odd number of 0s and draw the state diagram.

State diagram:

Draw the configuration for
000.

0000.

Modify machine M2 to recognize even number of 0s and draw the state diagram.

State diagram:

Draw the configuration for
000.

0000.