Ryan Darras, CS 5070 - HW08 3.(1,2,6,7,8)

Pardon the font change. I used curier new so everything would be aligned and easier to read.

Problem 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.
b) 00.
c) 000.
d) 000000.

Problem 3.1a Answer

q_1 0
_ q_2 _
_ _q_{accept}

Problem 3.1b Answer

q_1 0 0
_ q_2 0
_ x q_3 _
_ q_5 x _

Q_5 _ x _
_ q_2 x _
_ x q_2 _
_ x _ q_{accept}

Problem 3.1c Answer

q_1 0 0 0
_ q_2 0 0
_ x q_3 0
_ x 0 q_4 _
_ x 0 _ q_{reject}

Problem 3.1d Answer

q_1 0 0 0 0 0 0
_ q_2 0 0 0 0 0
_ x q_3 0 0 0 0
_ x 0 q_4 0 0 0
_ x 0 x q_3 0 0
_ x 0 x 0 q_4 _
_ x 0 x 0 x q_3 _
_ x 0 x 0 q_5 x _
_ x 0 x q_5 0 x _
_ x 0 q_5 x 0 x _

_ x q_5 0 x 0 x _
_ q_5 x 0 x 0 x _
q_5 _ x 0 x 0 x _
_ q_2 x 0 x 0 x _
_ x q_2 0 x 0 x _
_ x x q_3 x 0 x _
_ x x x q_3 0 x _
_ x x x 0 q_4 x _
_ x x x 0 x q_4 _
_ x x x 0 x _ q_{reject}

Problem 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.

a) 11.
b) 1#1.
c) 1##1.
d) 10#11.
e) 10#10

Note: I'm not quite sure how to handle rejects that are because of no possible transitions, but I'm assuming that we need to just have a transition to the reject state that is just ((all non-listed inputs) -> R)

Problem 3.2a Answer

q_1 1 1
X q_3 1
X 1 q_3 _
X 1 _ q_{reject}

Problem 3.2b Answer

q_1 1 # 1
x q_3 # 1
x # q_5 1
x q_6 # x
q_7 x # x
x q_1 # x
x # q_8 x
x # x q_8 _
x # x _ q_{accept}

Problem 3.2c Answer

q_1 1 # # 1
x q_3 # # 1
x # q_5 # 1
x # # q_{reject} 1

Problem 3.2d Answer

q_1 1 0 # 1 1
x q_3 0 # 1 1
x 0 q_3 # 1 1
x 0 # q_5 1 1
x 0 q_6 # x 1
x q_7 0 # x 1
q_7 x 0 # x 1
x q_1 0 # x 1
x x q_2 # x 1
x x # q_4 x 1
x x # x q_4 1
x x # x 1 q_{reject}

Problem 3.2e Answer

q_1 1 0 # 1 0
x q_3 0 # 1 0
x 0 q_3 # 1 0
x 0 # q_5 1 0
x 0 q_6 # x 0
x q_7 0 # x 0
q_7 x 0 # x 0
x q_1 0 # x 0
x x q_2 # x 0
x x # q_4 x 0

x x # x q_4 0
x x # q_6 x x
x x q_6 # x x
x q_7 x # x x
x x q_1 # x x
x x # q_8 x x
x x # x q_8 x
x x # x x q_8 _
x x # x x _ q_{accept}

Problem 3.6

In Theorem 3.21, we showed that a language is Turing-recognizable iff some enumerator enumerates it. Why didn't 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 Σ*.
E = "Ignore the input.

1. Repeat the following for i =1 ,2,3,....
2. Run M on s_i.
3. If it accepts, print out s_i."

Problem 3.6 Answer

This is like a "chicken has to come before the egg" problem where the chicken is the enumerator and the egg is the machine. If we tried to use the machine to create the enumerator, then we might get into a diabolical situation where the machine loops forever on some s_i. This means that we won't ever generate the enumerator, in which case we cannot use it to prove that the language is turing-recognizable. By having the enumerator first, we can just run the enumerator and compare its results to our input string.

Problem 3.7

Explain why the following is not a description of a legitimate Turing machine.

M_{bad} = "On input $\langle p \rangle$, 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; otherwise,reject."

Problem 3.7 Answer

First off, this is a very broad description that could probably be clarified in a more step by step kind of description. Second, step 3 states that if any settings evaluate to 0, we accept, otherwise we reject. If the input polynomial has a solution, fine… We get to the accept state and accept, but if the input polynomial doesn't have a solution the machine will never be able to tell based on the description provided, so it will never be able to go to the reject state.

Problem 3.8

Give implementation-level descriptions of Turing machines that decide the following languages over the alphabet {0,1}.

a) { w | w contains an equal number of 0s and 1s }
b) { w | w contains twice as many 0s as 1s }
c) { w | w does not contain twice as many 0s as 1s }

Note: For B and C, we are assuming that if we have no 0s and 1s, then it is NOT considered twice as many. Technically 0 = 2 * 0, so I'm just putting this here for clarification. To be considered twice as many, you must have values above 0.

Problem 3.8a Answer

On input string w:

1) Scan the tape and mark the first 0 that has not been marked. If no unmarked 0 is found, go to step 4. Otherwise, move the head back to the front of the tape and proceed to step 2.
2) Scan the tape and mark the first 1 that has not been marked and move to step 3. If no unmarked 1 is found, reject.
3) Move the head back to the front of the tape and go to step 1.
4) Move the head back to the front of the tape. Scan the tape to see if any unmarked 1s remain. If none are found, accept; otherwise, reject.

Problem 3.8b Answer

On input string w:

1) Scan the tape and mark the first 2 0s and the first 1 with different marks. If 2 0s or 1 1 is not found, reject. Otherwise, step back through the tape and switch the marked indices back to their respective value.
2) Scan the tape and mark the first two 0s that have not been marked. If no unmarked 0 is found, go to step 5, and if only 1 unmarked 0 is found, reject. Otherwise, move the head back to the front of the tape and proceed to step 2.
3) Scan the tape and mark the first 1 that has not been marked and move to step 4. If no unmarked 1 is found, reject.
4) Move the head back to the front of the tape and go to step 2.
5) Move the head back to the front of the tape. Scan the tape to see if any unmarked 1s remain. If none are found, accept; otherwise, reject.

Problem 3.8c Answer - Note: I just switched accept to reject and vica versa.

On input string w:

1) Scan the tape and mark the first 2 0s and the first 1 with different marks. If 2 0s or 1 1 is not found, accept. Otherwise, step back through the tape and switch the marked indices back to their respective value.
2) Scan the tape and mark the first two 0s that have not been marked. If no unmarked 0 is found, go to step 5, and if only 1 unmarked 0 is found, accept. Otherwise, move the head back to the front of the tape and proceed to step 2.
3) Scan the tape and mark the first 1 that has not been marked and move to step 4. If no unmarked 1 is found, accept.
4) Move the head back to the front of the tape and go to step 2.
5) Move the head back to the front of the tape. Scan the tape to see if any unmarked 1s remain. If none are found, reject; otherwise, accept.