1. Formal Definition of Turing Machine

Examine the formal definition of a Turing machine to answer the following questions, explaining your reasoning:

1. Can a TM ever write the blank symbol on its tape?
   Answer: Yes. A TM can write any symbol from \( \Gamma \), and since \( \Gamma \) contains the blank symbol, TMs can write blanks.

2. Can the tape alphabet \( \Gamma \) be the same as the input alphabet \( \Sigma \)?
   Answer: No. The blank symbol must not belong to \( \Sigma \) but must belong to \( \Gamma \).

3. Can a TM head ever be in the same location in two successive steps?
   Answer: Yes. We agreed that a TM should never move its head past the left-hand end of its tape, hence if there is a series of attempts to go past the left-hand end, they will all yield the same configuration, with the same location.

4. Can a TM contain just a single state?
   Answer: No. A TM must have at least two states, \( q_{\text{acc}} \) and \( q_{\text{ rej}} \) which must be distinct. However, there is no need for three states, as \( q_0 \), the start state, can be either \( q_{\text{acc}} \) and \( q_{\text{ rej}} \).

2. An "Ill-Defined" Turing Machine

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

\[ M = \text{"The input is a polynomial } p \text{ over variables } x_1, \ldots, x_k. \]

1. Get all possible assignments of \( x_1, \ldots, x_k \) to integer values.
2. Evaluate \( p \) on all of these values.
3. If any of these evaluations is 0, accept; otherwise reject."

Answer: Step 1 requires infinite memory to store all the assignments and step 2 requires infinite processing time to go through all the possible values.

3. Programming Turing Machines

**Item 1:** Define a Turing machine to accept all strings made up of 0s and 1s but the size of which is an even number. That, your TM should accept strings 01, 00, 11, 0100, and so on, but not strings 1, 0, 101, 111, and so on. 

Answer: The following state diagram specifies the TM

\[ S_{\text{acc}} \rightarrow 0 \rightarrow 0, R \]
\[ S_0 \rightarrow 1 \rightarrow 1, R \]
\[ 0 \rightarrow 0, R \]
\[ S_1 \rightarrow 1 \rightarrow 1, R \]
\[ 1 \rightarrow \#, L \]

**Item 2:** Define a Turing machine to do the following: given an input string of 0's, your machine should end up with a string of 0's five times the original size on the tape. For instance, if the machine starts with string 00 it should stop with string 0000000000 (5 times 2 = 10 zeros).

Answer: Assume the input is \#w, where w is the string of 0’s. The TM should follow the steps:

1. Move the head to the second cell (where w begins)
2. If the contents of the cell the tape is reading is 0, write x on the cell
3. Go to the end of the string (that is, to the first blank) and replace four blanks each with an x
4. Go back to the second cell (use "#" to detect when we get there)
5. Move to the right until we find a 0
6. Go to step 1
7. When there are no more zeros, change all x’s to 0’s
8. Stop
4. Implementation-Level Description of Turing Machines

Give implementation-level descriptions of TMs to decide the following languages over the alphabet \{0,1\}:

1. \[\{w \mid w \text{ contains an equal number of 0s and 1s}\}\]

**Answer:** The TM should follow the steps:
1. Search from left-to-right for a 0; when one is found, write x on the cell
2. Go back to the beginning and search for a 1, if one is found, write x on the cell; if none is found, reject.
3. Go back to step 1
4. If there are only x’s then accept; otherwise, reject

2. \[\{w \mid w \text{ contains twice as many 0s as 1s}\}\]

**Answer:** The TM should follow the steps:
1. Search from left-to-right for a 0; when one is found, write x on the cell
2. Go back to the beginning and search for a 1, if one is found, write x on the cell; if none is found, reject.
3. Go back to the beginning and search for another 1, if one is found, write x on the cell; if none is found, reject.
4. Go back to step 1
5. If there are only x’s then accept; otherwise, reject

5. Proofs!

Show that decidable languages are closed under the operations of union and intersection.

**Answer:** Let there be two decidable languages \(L_1\) and \(L_2\). Since they are decidable, then each has a decider \(M_1\) and \(M_2\), respectively, such that \(L(M_1) = L_1\) and \(L(M_2) = L_2\). We can prove that the union of these languages \(L' = L_1 \cup L_2\) is also decidable by showing how we can build a decider \(M'\) for \(L'\) using the deciders for \(L_1\) and \(L_2\).

We shall build our decider for \(L'\) using a two-tape Turing machine which accepts any string in \(L_1 \cup L_2\):

\[M' = \text{“On input string } w,\]
1. Copy input string \(w\) to tape 2, so that both tapes now contain \(w\).
2. Run \(M_1\) on tape 1; if \(M_1\) accepts \(w\), then accept; if \(M_1\) rejects \(w\) then go to step 3.
3. Run \(M_2\) on tape 2; if \(M_2\) accepts \(w\), then accept; if \(M_2\) rejects \(w\) then reject.”

Since both \(M_1\) and \(M_2\) are deciders (i.e., they always halt) then \(M'\) is also a decider – it will never loop. Moreover, we have seen in our lectures that every multitape TM has an equivalent single tape TM, so we can manipulate the definition above and make it a single tape TM.

The intersection is done similarly, but the machine is defined as follows:

\[M'' = \text{“On input string } w,\]
1. Copy input string \(w\) to tape 2, so that both tapes now contain \(w\).
2. Run \(M_1\) on tape 1; if \(M_1\) accepts \(w\), then go to step 3; if \(M_1\) rejects \(w\) then reject.
3. Run \(M_2\) on tape 2; if \(M_2\) accepts \(w\), then accept; if \(M_2\) rejects \(w\) then reject.”