Problem 1

Design a TM that decides the language \{0^n1^n|n \geq 0\}

- \(\Sigma = \{0, 1\}\)
- \(\Gamma = \{0, 1, \omega\}\)
- \(Q = \{q_0, q_1, q_2, q_3, q_a, q_r\}\)
- \(q_0 = \) the start state
- \(q_a = \) the accept state
- \(q_r = \) the reject state
- \(\delta\) given by the following picture (where unspecified, assume the head writes the same symbol it reads, and moves to the right).
Problem 3.11

Show that a Turing machine with doubly-infinite tape recognizes the same class of languages as an ordinary Turing machine.

- Clearly one direction is easy. The doubly-infinite tape can simulate an ordinary TM by just not using the portion of it’s tape to the left of the input.

- The other direction follows from the proof that a multitape TM is no more powerful than a single-tape machine. A two-tape Turing machine can simulate a doubly-infinite one by using it’s second tape as if it were the ‘negative’ half of the doubly-infinite tape.

Problem 3.12

Show that Turing machines with left reset recognize the class of Turing-recognizable languages.

- One direction is easy; regular Turing machines can obviously simulate left reset machines by using a method like that used in example 3.6 to find the left-hand end of the tape.

- The harder direction is getting a left reset machine to simulate a regular Turing machine. For a left reset machine to simulate an ordinary machine $M$, the machine can replace all of $M$’s left-moving states with the following algorithm for moving left over some character in the middle of the tape. Suppose the head wants to move left from the spot it is currently on.

  1. Dot the current spot. Reset to the first spot
  2. Dot the first spot. Reset.
  3. Move right until you hit a dot. Move right again.
  4. If the cell you are over already has a dot, it must be the original cell. Erase the dot, reset, and move right until you hit a dot. Now you are one spot to the left of your original spot, and you are done.
  5. If the cell you are over does not have a dot, dot it, reset, move until you hit a dot, erase the dot, reset, go to step 3.

Hand-running this algorithm a couple of times should convince one that it actually works :)

Problem 3.16

Show that a language is decidable if and only if some enumerator enumerates it in lexicographic order.
Suppose a language \( L \) is decidable by a TM \( M \). Let \( s_1, s_2, s_3, \cdots \) be a lexicographic ordering of the strings in \( \Sigma^* \). Then an enumerator \( E \) works as follows:

\[ E = \text{Ignore the input} \]

1. For \( i = 1, 2, 3 \)
2. Run \( M \) on \( s_i \)
3. If \( M \) accepts, print \( s_i \), If \( M \) rejects, move on."

Note that step 2 is guaranteed to terminate, because \( M \) decides (not just recognizes) \( L \).

Suppose a language \( L \) is enumerated in lexicographic order by an enumerator \( E \). If \( L \) is finite, then of course it’s decidable, so we suppose that \( L \) is infinite. A TM \( M \) which decides \( L \) works as follows:

\[ M = \text{On input } w \]

1. Wait for \( E \) to print a string \( s \).
2. If \( s = w \), accept
3. If \( s > w \) (lexicographically), reject
4. If \( s < w \), go back to step 1.

Note that this is guaranteed to halt, since \( E \) never runs out of strings, and there are only a finite number of strings \( \leq w \).

**Problem 3.19**

Is the language containing the single string \( s = \{ 0 \text{ if God does not exist}, 1 \text{ if God exists} \} \) decidable?

- Yes. Any finite language is decidable. It is not the job of the TM to decide whether or not God exists. Once \( s \) is fixed, the language is finite, and hence decidable