1. Give a Turing machine $M$ which adds 1 in binary. Assume that the input to $M$ is a nonempty string $w$ representing a number $k$ in binary. After $M$ halts, the tape should consist of a string representing $k + 1$ in binary, followed by blanks.

You should assume the standard input convention for $w$, and stick to the convention that $M$’s tape has a left-most tape square and cannot be extended to the left. You may assume that $M$ has extra symbols in its tape alphabet if you want (but the slickest solution doesn’t need any).

Present your TM either as a state diagram in the style used in the text book, or in the style of the CSC 365 course notes “Turing Machines and Reductions”, page 3 (except it would be helpful to annotate some of the TM instructions). Explain clearly how your TM works. Solutions without clear explanations will not be marked.

Solution:
The algorithm is simple. Move to the right looking for a blank. If the string is all 1s, overwrite the blank with a 0 and move left. If the input string contains a 0, then leave the blank and move left. In both case, continue left overwriting 1s with 0s until a 0 is reached. Overwrite the 0 with a 1. The key observation is that, if the input is all 1s, then the first 0 the TM will reach when moving left will be when it reaches the left end of the tape and it reads the 0 it just wrote. (Here we are using the Sipser convention that when the TM gets a move left command while scanning the left-most symbol, the head does not move.)

More formally, the input and tape alphabets are both $\{0, 1\}$. The set of states, with their intended meaning are

<table>
<thead>
<tr>
<th>state</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>moving right, no 0 found</td>
</tr>
<tr>
<td>$q_1$</td>
<td>moving right, 0 found</td>
</tr>
<tr>
<td>$q_2$</td>
<td>moving left, no 0 found</td>
</tr>
<tr>
<td>$q_3$</td>
<td>halting state</td>
</tr>
</tbody>
</table>

The transition function is as follows:

<table>
<thead>
<tr>
<th>state</th>
<th>symbol</th>
<th>action</th>
<th>state</th>
<th>symbol</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>0</td>
<td>$(q_1, 0, R)$</td>
<td>$q_1$</td>
<td>0</td>
<td>$(q_1, 0, R)$</td>
</tr>
<tr>
<td>$q_0$</td>
<td>1</td>
<td>$(q_0, 1, R)$</td>
<td>$q_1$</td>
<td>1</td>
<td>$(q_1, 1, R)$</td>
</tr>
<tr>
<td>$q_0$</td>
<td>b</td>
<td>$(q_2, 0, L)$</td>
<td>$q_1$</td>
<td>b</td>
<td>$(q_2, b, L)$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>0</td>
<td>$(q_3, 1, L)$</td>
<td>$q_2$</td>
<td>1</td>
<td>$(q_2, 0, L)$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>b</td>
<td>$(q_2, b, L)$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2. Show that SD is closed under union and intersection. That is, show that if \( A \) and \( B \) are each semi-decidable, then so are \( A \cup B \) and \( A \cap B \). NOTE: This problem is solved on page 4 of the Notes “Computability and Noncomputability”. Therefore you should give a different proof, based on enumerators: Given enumerators for \( A \) and \( B \), show how to use them to get enumerators for \( A \cup B \) and \( A \cap B \).

**Solution:**
Let \( M_A \) be an enumerator for \( A \) and let \( M_B \) be an enumerator for \( B \). Thus \( A = E(M_A) \) and \( B = E(M_B) \).

We construct an enumerator \( M_U \) for \( A \cup B \) as follows. \( M_U \) alternately simulates a step of \( M_A \) and a step of \( M_B \). Each time \( M_A \) writes a string \( w \) on its output tape, \( M_U \) writes \( w \) on its output tape. Each time \( M_B \) writes a string \( w \) on its output tape, \( M_U \) writes \( w \) on its output tape. It is easy to see that \( E(M_U) = A \cup B \).

We construct an enumerator \( M_I \) for \( A \cap B \) as follows. \( M_I \) alternately simulates a step of \( M_A \) and a step of \( M_B \). \( M_I \) keeps a list \( L_A \) of all strings that \( M_A \) has output so far, and also a list \( L_B \) of all strings that \( M_B \) has output so far. Each time that \( M_A \) outputs a new string \( w \), \( M_I \) adds \( w \) to the list \( L_A \), and checks to see if \( w \) occurs on the list \( L_B \). If \( w \) occurs on \( L_B \), then \( M_I \) outputs \( w \).

Similarly each time \( M_B \) outputs a string \( w \) \( M_I \) adds \( w \) to the list \( L_B \), and checks to see if \( w \) occurs on \( L_A \). If so, then \( M_I \) outputs \( w \).

It is easy to see that \( E(M_I) = A \cap B \).

3. Recall from page 6 of the Notes “Computability and Noncomputability” that \( \text{DIAG} = \{ \langle M \rangle \mid \langle M \rangle \not\in L(M) \} \). Modify the proof in the Notes showing that \( \text{DIAG} \) is undecidable to show that the set

\[
\text{DD} = \{ \langle M \rangle \mid \langle M \rangle \langle M \rangle \not\in L(M) \}
\]

is undecidable.

**Solution:**
We need to modify the diagonal argument in the Notes and give a reduction.
The new diagonal argument replaces \( \text{DIAG} \) by \( \text{DD}' \), where

\[
\text{DD'} = \{ \langle M \rangle \langle M \rangle : M \text{ is a Turing machine and } \langle M \rangle \langle M \rangle \not\in L(M) \}
\]

Then \( \text{DD'} \) is not decidable, because if \( L(M_0) = \text{DD'} \), then we argue as before that

\[
\langle M_0 \rangle \langle M_0 \rangle \in L(M_0)
\]

is true iff it is false.

Now we will show \( \text{DD'} \leq_m \text{DD} \), so \( \text{DD} \) is also undecidable. Given an input \( \langle M \rangle \langle M \rangle \) to \( \text{DD'} \), where \( M \) is a Turing machine, define

\[
f(\langle M \rangle \langle M \rangle) = \langle M \rangle
\]
and define \( f(w) = w_0 \) if \( w \) does not have the form \( \langle M \rangle \langle M \rangle \) (where \( w_0 \) is, say, the empty string). Then for all strings \( w \),

\[
 w \in DD' \text{ iff } f(w) \in DD
\]
as required.

4. Let

\[
 A = \{ \langle M_1 \rangle, \langle M_2 \rangle, \langle M_3 \rangle, \ldots \}
\]

be a semidecidable set of Turing machine descriptions. Show that there exists a decidable set \( B \) of Turing machine descriptions which has the same set of associated languages \( L(M_i) \) as \( A \).

**Hint:** Show how you can pad the description \( \langle M \rangle \) of a Turing machine \( M \) to get an equivalent Turing machine, but with a much longer description.

**Solution**

We may assume that \( A \) is infinite, since if \( A \) is finite we can simply let \( B = A \). Since \( A \) is semidecidable, \( A = L(M_e) \) for some enumerator \( M_e \). We may assume that \( M_e \) enumerates the elements of \( A \) in the order given in (1).

We define

\[
 B = \{ \langle M'_1 \rangle, \langle M'_2 \rangle, \langle M'_3 \rangle, \ldots \}
\]

successively is such a way that \( L(M_i) = L(M'_i) \) for all \( i \), and

\[
 |\langle M'_i \rangle| < |\langle M'_{i+1} \rangle|, i = 1, 2, \ldots
\]
as follows.

Let \( \langle M'_i \rangle = \langle M_i \rangle \). For \( i \geq 1 \), let \( \langle M'_{i+1} \rangle \) be a padded version of \( \langle M_{i+1} \rangle \) such that \( M_{i+1} \) and \( M'_{i+1} \) are equivalent Turing machines, but (3) holds. This can be done simply by inserting sufficiently many harmless command sequences which cause the tape head to move back and forth many times without changing the contents of the tape.

It suffices to show that \( B \) is decidable. For this, we use that fact that any language for which the strings in the language can be enumerated in increasing order of length is decidable. In fact \( B \) can be enumerated in the increasing order indicated in (2), by modifying the enumerator \( M_e \) which enumerates \( A \) in the order given by (1) by converting \( \langle M_i \rangle \) to \( \langle M'_i \rangle \). (To do this, the revised enumerator can remember the length of \( \langle M'_i \rangle \) before converting \( \langle M_{i+1} \rangle \) to \( \langle M'_{i+1} \rangle \).)