Exercise  (Example 8.2 from textbook)

Construct a Turing Machine accepting the language
\[\{0^n 1^m \mid n \geq 1\}\]

Solution

The idea is that the TM $M$ that we construct needs the leftmost 0, turns it into X, and moves right until it reaches a 1, that is turned into Y. Then the head moves left again to the leftmost 0 (on the right to a X), and starts again until all 0's and 1's are turned into X's and Y's respectively.

If the input is not in $0^*1^*$, $M$ will fail to find a move and it won't accept. If $M$ changes the last 0 and the last 1 in the same round, it will go into the final state and accept.

$$Q = \{q_0, q_1, q_2, q_3, q_4\}$$

$$\Sigma = \{0, 1\}$$

$$\Gamma = \{0, 1, X, Y, B\}$$

($B$ denotes blank symbol)

$q_0$: start state

$F = \{q_9\}$

In $q_0$ is the state in which $M$ is when the head precedes the leftmost 0. In state $q_1$, $M$ moves right skipping 0's and 1's until it gets to a 1. In state $q_2$, $M$ moves left while skipping Y's and 0's again, until it gets to a X and goes again in $q_0$. 
Starting from $q_0$, if a $Y$ is read instead of a $0$, 
$M$ goes in $q_3$ and moves right: if a $1$ is found, 
then move one $1$ then $01s$; if a $0$ is read, 
then the initial string is accepted (transition to $q_4$).

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>X</th>
<th>Y</th>
<th>$\mathbf{5}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$(q_1, x, R)$</td>
<td>—</td>
<td>—</td>
<td>$(q_3, Y, R)$</td>
<td>—</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$(q_1, 0, R)$</td>
<td>$(q_2, Y, L)$</td>
<td>—</td>
<td>$(q_1, Y, R)$</td>
<td>—</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$(q_2, 0, L)$</td>
<td>—</td>
<td>$(q_0, X, R)$</td>
<td>$(q_2, Y, L)$</td>
<td>—</td>
</tr>
<tr>
<td>$q_3$</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>$(q_3, Y, R)$</td>
<td>$(q_4, 5, R)$</td>
</tr>
<tr>
<td>$q_4$</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
<td>—</td>
</tr>
</tbody>
</table>

**Exercise**

Show the computation of the TM above when the input string is:

(a) 00

(b) 000111

**Solution**

(a) $q_0 00 \rightarrow X q_2 0 \rightarrow X 0 q_4$

and the TM halts.

(b) $q_0 000111 \rightarrow X q_2 00111 \rightarrow X 0 q_4 0111$

$X 0 0 q_4 111 \rightarrow X 0 q_2 0 Y 11 \rightarrow X q_2 00 Y 11 \rightarrow q_2 X 00 Y 11 \rightarrow$

$X q_2 00 Y 11 \rightarrow X X q_2 0 Y 11 \rightarrow X X 0 q_2 Y 11 \rightarrow X 0 Y q_2 11 \rightarrow$

$X 0 q_2 Y 11 \rightarrow X X q_2 0 Y 11 \rightarrow X q_2 X 0 Y 11 \rightarrow X X 0 q_2 Y 11 \rightarrow$

$X X q_2 Y 1 \rightarrow X X X q_2 Y 1 \rightarrow X X X Y q_2 1 \rightarrow X X X Y q_2 Y \rightarrow$

$X X X q_2 Y Y \rightarrow X X q_2 X Y Y \rightarrow X X X q_0 Y Y \rightarrow X X X q_3 Y Y \rightarrow$

$X X X Y q_3 Y \rightarrow X X X Y Y q_4 5 \rightarrow X X X Y Y q_4 5$
Exercise (8.1.1 from textbook)

Give a reduction from the hello-world problem to the following problem:

given a program $P$ and an input $I$, does the program ever produce any output?

Solution

We modify $P$ by making it print its output on some array $A$, capable of storing 12 characters. When $A$ is full, $P$ checks whether it stores "hello world": if it does, $P$ prints (on the output, not on the array) some character (like @); if not, it does not print anything.

So the modified program prints some output if and only if $P$ prints "hello world": if we are able to determine whether a program produces any output, we can solve the hello-world problem. This ends our reduction.
Exercise (8.2.3 from textbook):

Design a Turing Machine that takes as input a number \( N \) in binary and turns it into \( N+1 \) (in binary); the number \( N \) is preceded by the symbol $, which may be destroyed during the computation. For example, $111$ is turned into $1000$; $1001$ is turned into $1010$.

Solution

The idea is to toggle the rightmost digit, and, from right to left, all consecutive 1's until we get to the first 0 (which is also toggled). If there is no 0 to be toggled, a 1 is added on the left of the first digit (i.e., in place of the $).

We need three states, where only \( q_2 \) is the final state; we briefly describe what the TM does in the different states.

\( q_0 \) : The TM goes right until it reaches 0, after the rightmost digit. When 0 is reached, the TM goes into \( q_2 \).

\( q_1 \) : Go left toggling all 1's and the first 0 (from right); when 0 or $ is reached, the symbol is turned into 1.

\( q_2 \) : Final state; the TM does nothing.

<table>
<thead>
<tr>
<th></th>
<th>$</th>
<th>0</th>
<th>1</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( (q_0, $, R) )</td>
<td>( (q_0, 0, R) )</td>
<td>( (q_0, 1, R) )</td>
<td>( (q_2, $, L) )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( (q_1, 1, L) )</td>
<td>( (q_1, 1, L) )</td>
<td>( (q_2, 0, L) )</td>
<td>---</td>
</tr>
<tr>
<td></td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>
Exercise  (8.22 from textbook)

Design Turing machines accepting the following language:

\[ \{ w \in \{0,1\}^* \mid \text{has an equal number of 0's and 1's} \} \]

solution

The idea is that the head of our TM M moves back and forth on the tape, "deleting" one 0 for each 1; if there are no 0's and 1's in the end, the string is accepted.
When in state \( q_1 \), M has found a 1 and looks for a 0; in state \( q_2 \) is the other way around.
Note that the head never moves left of any \( X \), so that there are never unmatched 0's and 1's on the left of an \( X \).
From initial state \( q_0 \), M picks up a 0 or a 1 and turns it into \( X \). The only final state is \( q_4 \). In state \( q_3 \), M moves head left, looking for the rightmost \( X \).

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>( \delta )</th>
<th>X</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( (q_2, X, R) )</td>
<td>( (q_2, X, R) )</td>
<td>( (q_4, \varepsilon, R) )</td>
<td>(-)</td>
<td>( (q_0, Y, R) )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( (q_3, Y, L) )</td>
<td>( (q_2, 1, R) )</td>
<td>(-)</td>
<td>(-)</td>
<td>( (q_2, Y, R) )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( (q_2, 0, R) )</td>
<td>( (q_2, Y, L) )</td>
<td>(-)</td>
<td>(-)</td>
<td>( (q_2, Y, R) )</td>
</tr>
<tr>
<td>( q_3 )</td>
<td>( (q_3, 0, L) )</td>
<td>( (q_3, 1, L) )</td>
<td>(-)</td>
<td>( (q_0, X, R) )</td>
<td>( (q_3, Y, L) )</td>
</tr>
<tr>
<td>( q_4 )</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
<td>(-)</td>
</tr>
</tbody>
</table>
Exercise (8.1.4 (a) from MHU)

Show that the following problem is undecidable, by giving a reduction from the Hello World problem:

Given a program $P$ and an input $x$,
does $P$ eventually halt when it is given $x$ as input?

(Note: this problem is called the Halting problem)

Solution:

We have to construct a reduction $R$ from the HWP to the HP.

$R$ is a program that:
- takes as input an instance $(Q, y)$ of the HWP, and
- produces as output an instance $(P, x)$ of the HP such that

$$Q(x) = \text{"Hello, World"} \iff P(x) \text{ eventually halts.}$$

(i.e., $\text{HWP}(Q, y) = \text{"yes"} \iff \text{HP}(P, y) = \text{"yes"}$)

Using $R$, we can show that the HP is undecidable.

Indeed, assume the HP is decidable, and let $S_{HP}$ be a solver for the HP, i.e.,

$$\begin{align*}
(P, x) \quad &\rightarrow S_{HP} \quad \rightarrow \text{"yes" if } P(x) \text{ halts} \\
&\quad \quad \quad \rightarrow \text{"no" if } P(x) \text{ does not halt}
\end{align*}$$

We now $R$ and $S_{HP}$ to construct a solver $S_{\text{HWP}}$ for the HWP:

$$\begin{align*}
(Q, y) \quad &\rightarrow R \quad \rightarrow (P, x) \quad \rightarrow S_{HP} \quad \rightarrow \text{"yes" if } P(x) \text{ halts} \\
&\quad \quad \quad \quad \rightarrow \text{"no" if } P(x) \text{ does not halt}
\end{align*}$$

Since $S_{\text{HWP}}$ does not exist, also $S_{HP}$ cannot exist.
We show now how to construct Red by describing what it does:

\[(x, y) \xrightarrow{\text{Red}} (P, x)\]

Red leaves \(y\) unchanged, i.e. \(x = y\).

Red performs on \(Q\) the following modifications:

1) It makes sure that \(Q\) never halts

   (e.g. by inserting \(\text{while (true) \{ \}}\) at the end of
   \(\text{main()}\) and before every \(\text{return;}\) in \(\text{main}()\))

2) It modifies the \(\text{print()}\) method so that it stores the
   printed characters in an array and then checks whether
   the array contains "Hello, World". If yes, \(Q\) halts.

The resulting program is \(P\).

Note that Red, which computes \(P\) from \(Q\) can be written
in Java.