1. Construct a Turing machine that accepts the complement of the language $L = L(aaa^*b^*)$. Assume that $\Sigma\{a,b\}$.

**Answer.**

A TM $M$ that accepts $L$ can be constructed as follows.

2. What language is accepted by the Turing machine whose transaction graph is in the figure below?

**Answer.**

$L = ab^* + bb^*a$.
3. Construct Turing machine that will accept the language $L = \{ww : w \in \{a,b\}^+\}$.

Answer.

The hard part here is that we don’t know where the middle of the string is. So we don’t know where the boundary between the first occurrence of $w$ ends and the second begins. We can break this problem into three subroutines, which will be executed in order:

(a) Find the middle and mark it. If there’s a lone character in the middle (i.e., the length of the input string isn’t even), then reject immediately.

(b) Bounce back and forth between the beginning of the first $w$ and the beginning of the second, marking off characters if they match and rejecting if they don’t.

(c) If we get to the end of the $w$’s and everything has matched, accept.

A TM $M$ that accepts $L$ can be constructed as follows.

4. Design a Turing machine to compute the function $f(x, y) = x + 2y$ for $x$ and $y$ positive integers represented in unary.

Answer.

A TM $M$ that accepts $L$ can be constructed as follows:

Idea

- $x$ and $y$ are unary-represented
- Use string copier to copy $y$ as $2y$
- Compute $x + 2y$
5. Using adders, subtracters, comparers, copiers, or multipliers, draw block diagrams for Turing machines that compute the function $f(n) = 2^n$. for all positive integers $n$.

Answer.

![Block Diagram](image)

6. Provide a 'high-level' description for Turing machines that accept the language $L = \{a^n b^{m^2} : n = m^2, m \geq 1\}$ on \{a,b\}. For each problem, define a set of appropriate macroinstructions that you feel are reasonably easy to implement. Then use them for the solution.

Answer.

The idea is to remove $n$ bs as we see an a on the tape.

(a) Make sure the input string is in $L(aa^*bb^*)$. (This can be done by checking the input string. We are expecting to see some as follow by some bs. If there is an a after bs, reject.)

(b) Mark the first unmarked a. If there is no more as, go to (e).

(c) Scan the tape and remove $n$ bs. A possible solution is by zigzagging. That is, mark an a and delete a b. Repeat $n$ times by giving as $n$ different marks. If there are not enough bs to remove ($m < n^2$), then halt and reject.

(d) Go back to the most-left side of the head and repeat from (b).

(e) If the first symbol after marked as is b, halt and reject ($m > n^2$). If the first symbol is blank, then halt and accept.
7. Suppose we make the requirement that a Turing machine can halt only in a final state, that is, we ask that \( \delta(q, a) \) be defined for all pairs \( (q, a) \) with \( a \in \Gamma \) and \( q \in F \). Does this restrict the power of the Turing machine?

**Answer.**

The following shows that the halt-in-final Turing machine and standard Turing machine are equivalent.

- Since a halt-in-final Turing machine is clearly an extension of the standard Turing machine, it is obvious that any standard Turing machine can be simulated by some halt-in-final Turing machine.
- A standard Turing machine \( \hat{M} \) can simulate the computation of a halt-in-final Turing machine by using the following arrangement.
  - Create a new trap-state \( q_{\text{trap}} \) with transitions to itself for all symbol \( a \in \Gamma \), i.e., \( \delta(q_{\text{trap}}, a) = (q_{\text{trap}}, a, L \text{ or } R) \).
  - For each non-final state \( q \), we define a new transition that bring each unused symbol \( a \), which causes halt in state \( q \), to the trap-state \( q_{\text{trap}} \).

Therefore, the modification in the halt-in-final Turing machine does not restrict the power of the Turing machine. \( \square \)

8. Write program for nondeterministic Turing machine that accepts the language \( L = \{ww^Rw : w \in \{a, b\}^+\} \). In each case, explain if and how nondeterminism simplifies the task.

**Answer.**

In the beginning, non-deterministic steps should be used to separate input \( w \) into three segments.

Input: \( w \), surrounded by blanks.

Assume that \( w \) starts with \( a \).

Beginning:

(a) Mark \( a \) as \( X \) and enter \( q_a \). (\( X \) is for first segment \( w_1 \), \( q_a \) will move forward to mark the end of the second segment non-deterministically.)

(b) \( q_a \) moves forward, mark an \( a \) as \( Y \) and enters \( q_{a1} \). (non-deterministically determine the end of the second segment \( w_2 \), that is, \( \delta(q_a, a) = \{(q_a, a, R), (q_{a1}, Y, R)\} \)

(c) \( \delta(q_{a1}, a) = (Z, L) \) ( \( Z \) is for the third segment \( w_3 \)).

For example, \( w = aabbaaaab \), after the above steps: \( w = aabbaaaab \rightarrow XabbaYZab \).

Remarks:

1) \( q_a \) may find a wrong end for the second segment such that \( aabbaaaab \rightarrow XabbYZaab \). This does not matter since TM will not enter a final state in the end if \( XabbYZaab \) is taken.

The rest are deterministic steps:
(d) Find the first symbol $\sigma$ after $X$ and mark it as $X$
(e) Move forward to find the same $\sigma$ right before $Y$ and mark it as $Y$
(f) Move forward to find the same $\sigma$ right after $Z$ and mark it as $Z$.

Remarks:

1) If TM cannot find $\sigma$ in step (d), move forward to find $\Box$ by passing $Y$ and $Z$.
   If encountering symbols other than $Y$ and $Z$, halt and enter a non-final state. If successfully reaching blank, halt and enter a final state.
2) If any of steps (e) and (f) cannot be satisfied, halt and enter a non-final state.

9. Give the encoding, using the suggested method, for the Turing machine with

$$
\delta(q_1,a_1) = (q_1,a_1,R), \\
\delta(q_1,a_2) = (q_3,a_1,L), \\
\delta(q_3,a_1) = (q_2,a_2,L).
$$

**Answer.**

\[
\begin{array}{cccc}
101010111 & 0010110110101 & 001110101101101 \\
\text{First transition} & \text{Second transition} & \text{Third transition}
\end{array}
\]

10. Sketch a Turing machine program that enumerates the set $\{0,1\}^+$ in proper order.

**Answer.**

**Idea:**
- Length 1: 0, 1
- Length 2: 00, 01, 10, 11
- Length 3: 000, 001, 010, 011, 100, 101, 110, 111
- ...

**Sketch:**

(a) Initial string: 0.
(b) Copy the last string.
(c) Find the rightmost 0, change it to 1 and then change all the 1’s, on the right of the found 0, to 0’s. If there is no any 0 in the string change all 1’s to 0’s and add a 0 on the leftmost of the string.
(d) Repeat the step (a).