A task a FSM cannot do:

determine if a binary string has more 1's than 0's.

Turing machine means to model anything that can be computed with any computational device.

Can compute anything you can compute with a laptop.

Simplicity of computing device \rightarrow Ease of programming.

Finite tape alphabet: \( \Gamma \) set of symbols

Must include "blank": (\( \ast \))

Finite input alphabet: \( \Sigma \) \( \Sigma^* \leq \Gamma \) \( \ast \notin \Sigma \)

Input to the TM is \( x \in \Sigma^* \) string of symbols from \( \Sigma \).

For example, if \( \Sigma = \{0, 1\} \)

\( \Sigma^* \) is the set of all finite binary strings

\[ \Sigma^* = \{ 0, 1, 00, 01, 10, 11, 000, 001, \ldots \} \]

\( \Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots \)
Memory = 1-dimensional tape. Infinite in one direction.
Discrete cells – can hold a single character from a tape alphabet.

Starting configuration of the tape:

TM has a finite set of states: \( q_0, q_1, \ldots, q_m \). = \( S \)

Current configuration has a
- current state
- location of a pointer called a "head"

Each step is dictated by a transition function:

\[ s \left( q_3, 1 \right) = \left( q_4, 0, \text{L/R} \right) \]
(animation).

Three special states in $S$:

$q_0$: Start State.
$q_{acc}$: accept State
$q_{rej}$: reject State.

$TM$ keeps executing steps until it reaches $q_{acc}$ or $q_{rej}$
$\rightarrow$ no "next step" from these states.

$$
\delta: S - \delta_{q_{acc}, q_{rej}}^{2} \times \Gamma \rightarrow S \times \Gamma \times \exists L, R ^{3}
$$
If final state is $\text{acc}$: accept input string
If final state is $\text{ rej}$: reject output string.

TM could "recognize" if there are more 1's than 0's in an input string by accepting strings that do have more 1's than 0's and rejecting ones that don't.

**Decision vs. Search Problems**

Decision problems: output Yes/No

Is $5 \times 3 = 16$?  Is there a path from a to b in G?

Search / Computation

Comput  $5 \times 3$  Find a path from a to b in G.

Input must be a string, just need to have a fixed input format.
Example 1:  \[ S = 3a, b3. \]
\[ \Gamma = 3a, b, \ast 3 \]

Accept \( x \in 2^+ \) if \( x \) has at least two bs.

\[ S = 3q_0, q_1, q_{acc}, q_{rej} \]
\[ L \quad \text{# bs seen so far.} \]

\[
\begin{array}{c|cccc}
S & \downarrow & b & \ast & \\
\hline
q_0 & (q_0, a, R) & (q_1, b, R) & (q_{rej}, \ast, L) \\
q_1 & (q_1, a, R) & (q_{acc}, b, R) & (q_{rej}, \ast, L).
\end{array}
\]

(This task could have been computed by an FSM.)

![Diagram](attachment:image.png)
More complex TM

\[ \Sigma = \{a, b, c\} \]
\[ \Gamma = \{a, b, \#\} \]

Accepts \( x \in \Sigma^* \) if the number of characters in \( x \) is a power of 2.

If \( n \) is a power of 2, can keep dividing \( n \) by 2 until you reach 1 with no remainder.

\[
8 \quad 4 \quad 2 \quad 1
\]

\[
12 \quad 6 \quad 3 \quad \#
\]

Head shuttles back and forth between the two ends of the string.

\[
\text{\# of a's}
\]

\[
\text{\# of a's}
\]

\[
\text{\# of a's}
\]

\[
\text{\# of a's}
\]

\[
\text{\# of a's}
\]

\[
\text{\# of a's}
\]

\[
\text{\# of a's}
\]
Design a TM that accepts a binary string if and only if it has the same # of 0's as 1's.

101100 (accept) →
101101 (reject).

Idea: head shuttles back and forth, blanking out 0/1 pairs.

States:
q0: seen a 0, looking for a 1.
q1: seen a 1, looking for a 0.
nore: shuttering back to the beginning.

Characters: Need a X character to blank out 0's and 1's counted.

Moving Right: Start in q0 (haven't seen anything yet).

See a 0: replace w/ X, Go to q1.
See a 1: replace w/ X, go to q0.

q1: see a 0: keep moving right.
See a 1: replace w/ X → return

q0: see a 1: keep moving right.
See a 0: replace w/ X → return.
Moving left in grid: move left until....?

E moves left and stays.

\[ \begin{array}{ccc}
X & O & X \\
X & O & X \\
\end{array} \]

\[ \begin{array}{ccc}
O & 1 & 0 & * & * & * \\
E & 1 & 0 & * \\
\end{array} \]
\[ S = \{ q_0, q_2, q_{\text{one}}, q_{\text{ret}}, q_{\text{acc}}, q_{\text{err}} \} \]

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>X</th>
<th>*</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_0 )</td>
<td>( q_2, E, R )</td>
<td>( q_{\text{one}}, E, R )</td>
<td>( q_0, X, R )</td>
<td>( q_{\text{acc}}, * )</td>
<td>( _ )</td>
</tr>
<tr>
<td>( q_2 )</td>
<td>( q_2, 0, R )</td>
<td>( q_{\text{ret}}, X, L )</td>
<td>( q_2, X, R )</td>
<td>( q_{\text{err}}, * )</td>
<td>( _ )</td>
</tr>
<tr>
<td>( q_{\text{one}} )</td>
<td>( q_{\text{one}}, X, L )</td>
<td>( q_{\text{one}}, 1, R )</td>
<td>( q_{\text{one}}, X, R )</td>
<td>( q_{\text{one}}, X, )</td>
<td>( _ )</td>
</tr>
<tr>
<td>( q_{\text{ret}} )</td>
<td>( q_{\text{ret}}, 0, L )</td>
<td>( q_{\text{ret}}, 1, L )</td>
<td>( q_{\text{ret}}, X, L )</td>
<td>( _ )</td>
<td>( q_0, E, R )</td>
</tr>
</tbody>
</table>