Lecture 8 - Turing machines

Example: DFA for $0^*1^*$

\[
\begin{array}{c}
\text{start} \\
\rightarrow \\
\downarrow \\
q_1 \\
\rightarrow \\
\downarrow \\
q_{accept} \\
\rightarrow \\
\downarrow \\
q_{reject}
\end{array}
\]

TM: * Intuition
* Examples: $0^*1^*$, $\{0^n1^n\}$
* Definitions
  - TM, its computation, its language

TM for $0^*1^*$:

A TM is allowed to move forward and backward through its input, and also overwrite its input.

Omitting a symbol means we leave the symbol unchanged.

$0 \rightarrow 0, R$ is the same as $0 \rightarrow R$.

Main differences from a DFA:
* can go right and left
* can write over input symbols
* have an infinite amount of tape available (to the right of the input).

(If there's a transition missing, by convention, we reject.)
A TM for $On^1$:

**Idea**: Match every 0's with a 1 and cross out pairs. If there's nothing left after matching all pairs, accept.

**Model of TM**:

```
    TM control
       ←→
        ↑
   input  blank space
```

Back to $On^1$:
- Cross out a 0 (also, take care of 0's and 1's)
  - Skip over any 0's and 1's to find a 1 to cross out
    - If none found, reject
    - Hop back to the leftmost 0
    - If none found, skip over 1's to see if any 0's or 1's remain— if none, accept; else reject
    - Cross out the leftmost 0, go to $\text{0}$

**States**:

- $q_{\text{start}}$: just started
  - If see $U$, accept
  - If see 1, reject
  - Else, cross out 0, move right

- $q_1$: just crossed out a 0; skip ahead until find a 1.

- $q_2$: just crossed out a 1, hop back until find $\varnothing$

- $q_3$: looking for leftmost 0.

- $q_4$: no more 0's left, skipping ahead to see if we have any more 0's or 1's
  - If yes, reject; else accept

**Note**: $\varnothing$ and 1 are symbols in the tape alphabet; we can do this without these symbols, but that comes later.
Note: Input alphabet doesn't contain \( \omega \). Also, \( \omega \) takes up a space, while \( \epsilon \) doesn't.

State diagram:

Step 1: 0 0 0 1 1 1 1 ...

\( q_{\text{start}} \)

Step 2: 0 0 0 1 1 1 1 ...

\( q_1 \)

Step 3: 0 0 0 1 1 1 1 ...

\( q_1 \)

etc.

Is there a bug in this TM? We think there isn't.
Formal definition:

A Turing machine is a 7-tuple: \((Q, \Sigma, \Gamma, s, q_0, q_A, q_R)\).

1. Finite set of states
2. \(\Sigma\) : finite alphabet for the input, \(\Sigma \neq \emptyset\)
3. \(\Gamma\) : finite tape alphabet \(\Sigma \subseteq \Gamma\), \(w \in \Gamma\)
4. \(S: \mathcal{Q} \times \Gamma \rightarrow \mathcal{Q} \times \Gamma \times \{L, R\}\)
5. \(q_0, q_A, q_R\) are start, accept, and reject states, respectively.

A configuration \(C\) of a TM \(M\) is a triple \(\langle q, i, u \rangle\),

where \(q \in Q\), \(i\) is an index, and \(u\) is a string over \(\Gamma\).

\((q, i, u)\) means

This is a snapshot before it takes the transition.

Another way to describe a configuration \(C\):

Split up \(u = x_1 y\): \(x\) - first \(i-1\) symbols of \(u\), \(y\) - rest.

Write \(C = x q y\)
(Minimal description of $C$ is one in which the blanks to the right of the last non-blank symbol of $y$ are omitted.)

Consider a configuration $u_{aq}ibv$, where $a, b \in \Gamma, iv$ strings over $\Gamma$. We say $u_{aq}ibv$ yields $u_{aq}acv$ for $c \in \Gamma$ if $\delta(q_i, b) = (q_j, c, L)$. Similarly, $u_{aq}ibv$ yields $uacqv$ if $\delta(q_i, b) = (q_j, c, R)$.

$q_{ibv}$ yields $q_{acv}$ if $\delta(q_i, b) = (q_j, c, L)$

$u_{aq}$ is equivalent to $u_{aq};L$, and yields whatever $u_{aq};L$ yields.

Start configuration: $\exists w$ for $w \in \Sigma^+$

Accept config: $x\text{accept } y$ for any $x, y$

Reject config: $x\text{ reject } y$ for any $x, y$

A TM $M$ accepts $w$ if $\exists$ Configs $C_0, C_1, \ldots, C_k$ of $M$ s.t. $C_0$ is the start config of $M$ on $w$, $C_i$ yields $C_{i+1}$ for $0 \leq i \leq k-1$, and $C_k$ is an accept config.

For rejecting $w$, just replace accept with reject above.

Def. Let $M$ be a TM. Then $L(M) = \{w \mid M \text{ accepts } w\}$

Question: Is $L(M) = \{w \mid M \text{ rejects } w\}$?

Answer: No. TMs can loop forever.
Def: $M$ is a decider if $\forall w$, $M$ either accepts or rejects $w$.

If $M$ is a decider, $L(M) = \{ w \mid M$ rejects $w \}$.

Def: A language $L$ is Turing-recognizable if $L = L(M)$ for some TM $M$.

Def: $L$ is decidable if $L = L(M)$ for some decider $M$.

Church-Turing Thesis: anything that can be programmed can be programmed on a TM.

Example of an undecidable language:

$A_{TM} = \{ \langle M, w \rangle \mid M$ is a description of a TM, $w$ is a description of its input, and $M$ accepts $w \}$

$A_{TM}$ isn't even Turing-recognizable.