Start Chapter 3 - Turing machines

§3.1 Turing machines as algorithms

- Formally: $(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})$

  plus blank symbol $\Lambda$

  $\Sigma$ = input alphabet $\subset \Gamma$ = tape alphabet
  (includes also $\Lambda$)

  $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$

- Examples to recognize
  - $\{ w \in \{a,b\}^* | \#a's \text{ in } w = \#b's \}$
  - $\{0^n1^n\}$

- TM descriptions
  - High level
  - Implementation level
  - Formal description (give $\delta$)
§ 3.2: multitape & non-deterministic Turing machines

§ 3.3: descriptions of \{ graphs, Boolean formulas, etc. \}

Midterm on Nov 5, will cover material up to October 22 (end of next week)

Details to follow...
Turing Machines:

Part 1: 

Input/Work Tape

Σ alphabet for input, larger alphabet \( \Gamma = \Sigma \cup \{ \# \} \cup \{ \text{additional block symbols} \} \)

Rules for input: \( \Sigma = \{ 0, 1 \} \)

Interested in \( L = \{ 0^n 1^n \mid n \in \mathbb{N} \} \)

Input \( w \in \{ 0, 1 \}^* \), \( w = \sigma_1 \ldots \sigma_k \)

In addition:

- \( Q \) set of states, finite set, intuitively \( Q \) is "the program"
- \( q_0 \) initial state
- \( q_A \) final state

\( \triangledown \) tape head

tells you which state
Formally: a Turing machine: 7-tuple

\[(Q, \Sigma, \Gamma, \delta, q_0, q_{acc}, q_{rej})\]

- \(Q\): states
- \(\Sigma\): input alphabet
- \(\Gamma\): tape alphabet
- \(\delta\): transition function
- \(q_0\): initial state
- \(q_{acc}\): accept state
- \(q_{rej}\): reject state

Example: Want to recognize: \(a(b,a)^*\ b\)

(this is a regular language)

Turing machine approach: input

<table>
<thead>
<tr>
<th>State</th>
<th>Tape Cell</th>
<th>Move</th>
<th>Right or Left</th>
</tr>
</thead>
<tbody>
<tr>
<td>(q_0)</td>
<td>a b b a a</td>
<td>- - - - - -</td>
<td>(\delta)</td>
</tr>
</tbody>
</table>

transition function: \(\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R\}\)

- State
- Tape cell
- New state
- Overwrite on tape cell
To recognize $L = a(a,b)^* b$:

1. Make sure input begins "a" (otherwise reject).
2. Move to the end of tape.
3. Make sure last letter is a "b" (then accept, otherwise reject).

Formally: $(w 	ext{ or } b) ightarrow (a, R)$

Diagram:

- $(a, a) ightarrow$ (state 1)
- $b ightarrow (b, R)$
- $(w) ightarrow (L, L)$
- $b ightarrow$ do anything
- $a ightarrow$ any
- $(a, R) ightarrow$ (state 2)
- $(a, L) ightarrow$ (state 3)
- $(b, R) ightarrow$ (state 4)

ε-steps!

Computation steps!
Describe with table

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>etc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>etc.</td>
<td>etc.</td>
<td>etc.</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$q_{acc}$, R</td>
<td>$q_{acc}$, R</td>
<td>$q_{just}$, L</td>
</tr>
<tr>
<td>$q_{just}$, R</td>
<td>$q_{just}$, L</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$q_{acc}$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$q_{rej}$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$Q \times \Gamma \times \{L, R\}$

Now, we want to *convince ourselves* that Turing machines can solve any decision problem $\Sigma^* \rightarrow \{\text{accept}, \text{reject}\}$ that you could write in Python, C++, Javascript, etc...
Describe Turing machines:

- Formal: describe \( \delta \) (and explain algorithm)
- Middle: Implementation
- High-level

Take \( \{0^n1^r \mid n \in \mathbb{N}\} = \{01, 0011, 011, \ldots\} \)

give Turing machine algorithm: \( \Sigma = \{0, 1\} \)

Given

\[
\begin{array}{cccccccc}
0 & 0 & 1 & 1 & 1 & w & w & w \\
\end{array}
\]

Start in \( q_0 \)

Input 00111 (should reject)

\( \Gamma = \{0, 1, \#\} \), any other finite number of symbols

\( e.g. \{c, 1, 2, \#\} \) OK (\( \Gamma \) finite)

\( \{c, 1, 2, 3, \ldots, \#\} \) (\( Q \) finite)
Compare Turing machines vs. C++ programs:

- Description:
  - Turing machines: Simple
  - C++ programs: Much more complicated

- Computation:
  - Real-world: Unrealistically slow
  - Better idea

Idea to recognize \{ 01, 0011, 000111, \ldots \}:

1. Initially
   - \( q_0 \)

2. \( q_0 \) moves right
   - Move one left
     - Write an \( X \) (or \( Y \), \( u \), or \( w \))
6) write back to first \( x \), then sort of start again

\[ \text{Break 10:20 am, back in 5 min} \]

State diagram: \( L = \{0^1, c011\} \ldots \), use textbook shorthand

\[ \begin{array}{c}
0 \rightarrow c, R \\
\rightarrow \\
q_0 \rightarrow q_1 \\
\rightarrow \\
q_1 \rightarrow q_2 \\
\rightarrow \]

\[ \begin{align*}
\text{Shortened} & : 0 \rightarrow c, R & \text{Shortened} & : 0 \rightarrow R \\
\text{Shortened} & : 1 \rightarrow c, R & \text{Just} & : 1 \rightarrow c, R \\
\text{Shortened} & : 0,1 \rightarrow R & \text{Shortened} & : 0,1 \rightarrow R
\end{align*} \]
shorthand: if \((q, r) \in Q \times \Gamma\)

is never encountered

\[ q \rightarrow r \quad \text{it's OK to leave it out} \]

So far:

\[ L = \{ 0^1, 001^1 \alpha \ldots \} \], use textbook shorthand

---

\[ q_0 \]

\[ 1, 1 \rightarrow R \]

\[ 0 \rightarrow 0 \]

\[ 0, 1 \rightarrow R \]

\[ \text{move to right end} \]

\[ \text{at rightmost symbol} \]

\[ q_{\text{accept}} \]

\[ q_{\text{reject}} \]

\[ q_1 \]

\[ x_0 \text{...} x_1 \text{...} x_n \]

\[ x_2 x_0 \text{...} x_n \]

\[ \text{might see} \]

\[ x_2 x_0 \text{...} x_n \]
next time! look a bit more
then introduce 2-type machines

Individual Q2! 51 states, not 50 states