Using Introduction to the Theory of Computation, Chapter 7
Outline

FSAs (finite state automata)

notes
turnstile finite-state machine

what are the rules for turnstiles?
float machine

which strings are floats in Python?
states needed to classify a string
what state is a stingy vending machine in, based on coins?
accepts only nickles, dimes, and quarters,
no change given, and everything costs 30 cents...
here’s a useful toy

<table>
<thead>
<tr>
<th>δ</th>
<th>0</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>25</th>
<th>≥ 30</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>5</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>25</td>
<td>≥ 30</td>
<td>≥ 30</td>
</tr>
<tr>
<td>d</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>25</td>
<td>≥ 30</td>
<td>≥ 30</td>
<td>≥ 30</td>
</tr>
<tr>
<td>q</td>
<td>25</td>
<td>≥ 30</td>
<td>≥ 30</td>
<td>≥ 30</td>
<td>≥ 30</td>
<td>≥ 30</td>
<td>≥ 30</td>
</tr>
</tbody>
</table>
integer multiples of 3
build an automaton with formalities...

quintuple: \((Q, \Sigma, q_0, F, \delta)\)

- \(Q\) is set of states, \(\Sigma\) is finite, non-empty alphabet, \(q_0\) is start state
- \(F\) is set of accepting states, and \(\delta: Q \times \Sigma \mapsto Q\) is transition function

We can extend \(\delta: Q \times \Sigma \mapsto Q\) to a transition function that tells us what state a string \(s\) takes the automaton to:

\[
\delta^*: Q \times \Sigma^* \mapsto Q \quad \delta^*(q, s) = \begin{cases} 
q & \text{if } s = \varepsilon \\
\delta(\delta^*(q, s'), a) & \text{if } s' \in \Sigma^*, \ a \in \Sigma, \ s = s'a
\end{cases}
\]

String \(s\) is accepted if and only if \(\delta^*(q_0, s) \in F\), it is rejected otherwise.
example — an odd machine
devise a machine that accepts strings over \( \{a, b\} \) with an odd number of \( a \)s

Formal proof requires inductive proof of invariant:

\[
\delta^*(E, s) = \begin{cases} 
E & \text{if } s \text{ has even number of } a \text{s} \\
O & \text{if } s \text{ has odd number of } a \text{s}
\end{cases}
\]
more odd/even: intersection

$L$ is the language of binary strings with an odd number of $a$s, and at least one $b$

Devise a machine for $L$ using product construction
more odd/even: union

$L$ is the language of binary strings with an odd number of $a$s, or at least one $b$

Devise a machine that accepts $L$ using product construction
**more odd/even**

$L$ is the language of binary strings with an odd number of $a$s, but even length.

Devise a machine for $L$ using product construction.
notes