Using Introduction to the Theory of Computation, Chapter 7
Outline

formal languages

FSAs

notes

annotations
some definitions

alphabet: finite, non-empty set of symbols, e.g. \{a, b\} or \{0, 1, -1\}. Conventionally denoted \(\Sigma\).

string: finite (including empty) sequence of symbols over an alphabet: \(abba\) is a string over \{a, b\}.
Convention: \(\varepsilon\) is the empty string, never an allowed symbol, \(\Sigma^*\) is set of all strings over \(\Sigma\).

language: Subset of \(\Sigma^*\) for some alphabet \(\Sigma\). Possibly empty, possibly infinite subset. E.g. \{\}, \{aa, aaa, aaaa, ...\}, \(L = \Sigma_0, 13^*\)

N.B.: \(\{\} \neq \{\varepsilon\}\). \(|\varepsilon\varepsilon^3\| = 1\)

\(|\varepsilon| = 0\)
Many problems can be reduced to languages: logical formulas, identifiers for compilation, natural language processing. Key question is recognition:

Given language $L$ and string $s$, is $s \in L$?

Languages may be described either by descriptive generators (for example, regular expressions) or procedurally (e.g. finite state automata)
more notation

string length: denoted $|s|$, is the number of symbols in $s$, e.g. $|bba| = 3$.

$s = t$: if and only if $|s| = |t|$, and $s_i = t_i$ for $1 \leq i \leq |s|$.

$s^R$: reversal of $s$ is obtained by reversing symbols of $s$, e.g. $1011^R = 1101$.

$st$ or $s \circ t$: concatenation of $s$ and $t$ — all characters of $s$ followed by all those of $t$, e.g. $bba \circ bb = bbabb$.

$s^k$: denotes $s$ concatenated with itself $k$ times. E.g., $ab^3 = ababab$, $101^0 = \varepsilon$.

$\Sigma^n$: all strings of length $n$ over $\Sigma$, $\Sigma^*$ denotes all strings over $\Sigma$. 
language operations

\[ L \]: Complement of \( L \), i.e. \( \Sigma^* - L \). If \( L \) is language of strings over \( \{0, 1\} \) that start with 0, then \( \overline{L} \) is the language of strings that begin with 1 plus the empty string.

\[ L \cup L' \]: union

\[ L \cap L' \]: intersection

\[ L - L' \]: difference

\[ \text{Rev}(L) \]: \( \{s^R : s \in L\} \)

concatenation: \( LL' \) or \( L \cdot L' = \{rt \mid r \in L, t \in L'\} \). Special cases

\( L\{\epsilon\} = L = \{\epsilon\}L \), and \( L\{\} = \{\} = \{\}L \).

\( L_1, L_2 \) always equal \( L_2 L_1 \)
more language operations

exponentiation: $L^k$ is concatenation of $L$ $k$ times. Special case, $L^0 = \{\varepsilon\}$, including $L = \{\}$!

Kleene star: $L^* = L^0 \cup L^1 \cup L^2 \cup \ldots$. 

\[ 0^0 = 1 \]
states needed to classify a string

what state is a stingy vending machine in based on coins?
accepts only nickles (a), dimes (b), and quarters (c),
no change given, and everything costs 30 cents
useful toy (you’ll need JRE)

\[ \sum = \{n, d, q\}^3 \]

How much was paid to a very lame vending machine?
accepted strings \( n \rightarrow \) send \( \text{machine} \) to \( \geq 30 \)

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>0</th>
<th>5</th>
<th>10</th>
<th>15</th>
<th>20</th>
<th>25</th>
<th>( \geq 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>( \geq 30 )</td>
<td>( \geq 30 )</td>
</tr>
<tr>
<td>d</td>
<td>10</td>
<td>15</td>
<td>20</td>
<td>25</td>
<td>( \geq 30 )</td>
<td>( \geq 30 )</td>
<td>( \geq 30 )</td>
</tr>
<tr>
<td>q</td>
<td>25</td>
<td>( \geq 30 )</td>
<td>( \geq 30 )</td>
<td>( \geq 30 )</td>
<td>( \geq 30 )</td>
<td>( \geq 30 )</td>
<td>( \geq 30 )</td>
</tr>
</tbody>
</table>
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:

\[
\delta^*(q, s) = \begin{cases} 
q & \text{if } s = \epsilon \\
\delta(\delta^*(q, s'), a) & \text{if } s' \in \Sigma^*, a \in \Sigma, s = s' a
\end{cases}
\]

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 as

\[ \leq \exists a, b^* \]

\[
S \rightarrow E \rightarrow 0d \rightarrow b
\]

Conventions:
leave out transitions that never lead to on accept state.

Formal proof requires inductive proof of invariant:

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

use induction on \(|s|\)
(Simple induction)
you should write one up

\[
\text{if } \Sigma \text{ is an alphabet, then define } \Sigma^* \text{ as follows,}
\]
\[
\begin{align*}
1 & \quad \varepsilon \in \Sigma^* \\
2 & \quad \text{if } x \in \Sigma^* \text{ and } c \in \Sigma, \text{ then } xc \in \Sigma^*
\end{align*}
\]
Devise a machine that accepts $L_F$
more odd/even

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

Devise a machine for $L$.

\[ \Sigma = \{a, b\} \]

Possible states:
- $EE$ - even $a$'s, even length
- $OE$ - odd $a$'s, even length
- $OO$ - odd $a$'s, odd length

State invariant:
\[ S^*(EE, S) = \begin{cases} EE & \text{even } a, \text{ even length} \\ OE & \text{odd } a, \text{ even length} \\ OO & \end{cases} \]
notes
annotated slides