Finite Automata

Part Three
Recap from Last Time
DFAs

- A **DFA** is a
  - **D**eterministic
  - **F**inite
  - **A**utomaton
- Every state must have exactly one transition defined for each symbol in the alphabet.
If \( D \) is a DFA, the **language of \( D \)**, denoted \( \mathcal{L}(D) \), is \( \{ w \in \Sigma^* \mid D \text{ accepts } w \} \).
NFAs

- An **NFA** is a
  - **N**ondeterministic
  - **F**inite
  - **A**utomaton
- NFAs have no restrictions on how many transitions are allowed per state.
- They can also use ε-transitions.
- An NFA accepts a string \( w \) if there is some sequence of choices that leads to an accepting state.
Massive Parallelism

- An NFA can be thought of as a DFA that can be in many states at once.
- At each point in time, when the NFA needs to follow a transition, it tries all the options at the same time.
- The NFA accepts if any of the states that are active at the end are accepting states. It rejects otherwise.
New Stuff!
Designing NFAs
Designing NFAs

- **Embrace the nondeterminism!**
- Good model: *Guess-and-check*:
  - Is there some information that you'd really like to have? Have the machine *nondeterministically guess* that information.
  - Then, have the machine *deterministically check* that the choice was correct.
- The *guess* phase corresponds to trying lots of different options.
- The *check* phase corresponds to filtering out bad guesses or wrong options.
Guess-and-Check

\[ L = \{ w \in \{0, 1\}^* \mid w \text{ ends in 010 or 101} \} \]
Guess-and-Check

\[ L = \{ \ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \} \]
Guess-and-Check

\[ L = \{ \ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \} \]
Guess-and-Check

\[ L = \{ w \in \{0, 1\}^* \mid w \text{ ends in 010 or 101} \} \]
Guess-and-Check

\( L = \{ \ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \ \} \)
Guess-and-Check

\[ L = \{ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \} \]
Guess-and-Check

$L = \{ \ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \ \}$
Guess-and-Check

\[ L = \{ \; w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \; \} \]
Guess-and-Check

\[ L = \{ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \} \]
Guess-and-Check

\[ L = \{ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \} \]
Guess-and-Check

\[ L = \{ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \} \]
Guess-and-Check

\[ L = \{ w \in \{0, 1\}^* \mid w \text{ ends in 010 or 101} \} \]
Guess-and-Check

$L = \{ \ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \ \}$
Guess-and-Check

\[ L = \{ w \in \{a, b, c\}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w \} \]
Guess-and-Check

$L = \{ w \in \{a, b, c\}^* | \text{at least one of } a, b, \text{ or } c \text{ is not in } w \}$
Guess-and-Check

\[ L = \{ w \in \{a, b, c\}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w \} \]

Nondeterministically guess which character is missing.

Deterministically check whether that character is indeed missing.
Guess-and-Check

\[ L = \{ w \in \{a, b, c\}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w \} \]
Guess-and-Check

$L = \{ w \in \{a, b, c\}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w \}$
Guess-and-Check

$L = \{ w \in \{a, b, c\}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w \}$
Guess-and-Check

$L = \{ w \in \{a, b, c\}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w \}$

\[
\begin{array}{c}
\text{start} \\
\varepsilon \\
\varepsilon \\
\varepsilon \\
\end{array}
\]

\[
\begin{array}{c}
\varepsilon \\
a, c \\
b, c \\
\end{array}
\]

\[
\begin{array}{c}
a \ c \ c \ a \ c \ c \ c
\end{array}
\]
Guess-and-Check

$L = \{ \ w \in \{a, b, c\}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w \ \}$
Guess-and-Check

\[ L = \{ \ w \in \{a, b, c\}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w \ \} \]
Guess-and-Check

$L = \{ w \in \{a, b, c\}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w \}$
$L = \{ \ w \in \{a, b, c\}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w \ \}$
Implementing DFAs and NFAs
Tabular DFAs

- States: $q_0, q_1, q_2, q_3$
- Alphabet: $0, 1$
- Transitions:
  - From $q_0$: $0 \rightarrow q_1$, $1 \rightarrow q_0$
  - From $q_1$: $0 \rightarrow q_2$, $1 \rightarrow q_2$
  - From $q_2$: $0 \rightarrow q_2$, $\epsilon \rightarrow q_3$
  - From $q_3$: $\epsilon \rightarrow q_0$

- Initial state: $q_0$
- Accepting states: $q_3$
Tabular DFAs

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>$q_1$</td>
<td>$q_0$</td>
</tr>
<tr>
<td>$q_1$</td>
<td>$q_3$</td>
<td>$q_2$</td>
</tr>
<tr>
<td>$q_2$</td>
<td>$q_3$</td>
<td>$q_0$</td>
</tr>
<tr>
<td>$q_3$</td>
<td>$q_3$</td>
<td>$q_3$</td>
</tr>
</tbody>
</table>
Tabular DFAs

\[
\begin{array}{c|cc}
\star q_0 & q_1 & q_0 \\
q_1 & q_3 & q_2 \\
q_2 & q_3 & q_0 \\
\star q_3 & q_3 & q_3 \\
\end{array}
\]
Tabular DFAs

These stars indicate accepting states.
Tabular DFAs

Since this is the first row, it's the start state.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>*q₀</td>
<td>q₁</td>
<td>q₀</td>
</tr>
<tr>
<td>q₁</td>
<td>q₃</td>
<td>q₂</td>
</tr>
<tr>
<td>q₂</td>
<td>q₃</td>
<td>q₀</td>
</tr>
<tr>
<td>*q₃</td>
<td>q₃</td>
<td>q₃</td>
</tr>
</tbody>
</table>
Tabular DFAs

Question to ponder: Why isn’t there a column here for $\Sigma$?
int kTransitionTable[kNumStates][kNumSymbols] = {
    {0, 0, 1, 3, 7, 1, ...},
    ...,
};

bool kAcceptTable[kNumStates] = {
    false,
    true,
    true,
    true,
    ...
};

bool SimulateDFA(string input) {
    int state = 0;
    for (char ch: input) {
        state = kTransitionTable[state][ch];
    }
    return kAcceptTable[state];
}
Thought Experiment:
How would you simulate an NFA in software?
The diagram depicts a finite automaton with the following states and transitions:

- **Start State**: $q_0$
- **States**: $q_0, q_1, q_2, q_3$
- **Transitions**:
  - $(q_0, a) \rightarrow q_1$
  - $(q_1, b) \rightarrow q_2$
  - $(q_2, a) \rightarrow q_3$
  - $(q_3, \Sigma) \rightarrow q_0$

The input string $abaaba$ is shown, indicating the path the automaton follows through the states.
\begin{align*}
\Sigma & \to \mathbf{a} \\
\mathbf{a} & \to \mathbf{q}_1 \\
\mathbf{b} & \to \mathbf{q}_2 \\
\mathbf{a} & \to \mathbf{q}_3
\end{align*}
\[
\begin{align*}
q_3 \rightarrow q_3 \\
q_2 \rightarrow q_1 \\
q_1 \rightarrow q_0 \\
q_0 \quad \text{(start)}
\end{align*}
\]
\[
\begin{array}{c}
\Sigma \\
\end{array}
\]

\[
\begin{array}{ccc}
\text{start} & \text{q}_0 & \text{q}_1 & \text{q}_2 & \text{q}_3 \\
\text{a} & \text{b} & \text{a} \\
\{q_0\} & \{q_0, q_1\} & \\
\end{array}
\]
<table>
<thead>
<tr>
<th>State</th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${q_0}$</td>
<td>${q_0, q_1}$</td>
<td>${q_0}$</td>
</tr>
<tr>
<td>$q_1$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$q_2$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$q_3$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
\[ \begin{array}{c|cc}
\{ q_0 \} & a & b \\
\{ q_0, q_1 \} & \{ q_0 \} & \\
\{ q_0, q_1 \} & & \\
\{ q_0, q_1 \} & & \\
\{ q_0, q_1 \} & & \\
\{ q_0, q_1 \} & & \\
\{ q_0, q_1 \} & & \\
\{ q_0, q_1 \} & & \\
\end{array} \]
\[
\begin{align*}
\Sigma & \quad \rightarrow \\
q_0 & \quad a \rightarrow q_1 \\
q_1 & \quad b \rightarrow q_2 \\
q_2 & \quad a \rightarrow q_3
\end{align*}
\]

<table>
<thead>
<tr>
<th>State</th>
<th>(a)</th>
<th>(b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>{q_0}</td>
<td>{q_0, q_1}</td>
<td>{q_0}</td>
</tr>
<tr>
<td>{q_0, q_1}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{q_0, q_1}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
$\begin{array}{c}
\Sigma \\
\text{start} \\
q_0 \\
\text{a} \\
q_1 \\
\text{b} \\
q_2 \\
\text{a} \\
q_3 \\
\end{array}$

$\begin{array}{c|c|c}
& a & b \\
\hline
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & & \\
\{q_0, q_1\} & & \\
\end{array}$
\begin{array}{c|c|c}
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & & \\
& & \\
& & \\
\end{array}
\[
\begin{array}{c}
\Sigma \\
\text{start}
\end{array}
\]

\[
\begin{array}{cccc}
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & & & \\
& & & \\
& & & \\
& & & \\
& & & \\
\end{array}
\]
\[
\begin{array}{c|cc}
& a & b \\
\hline
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & & \\
\end{array}
\]
<table>
<thead>
<tr>
<th>State Set</th>
<th>Input a</th>
<th>Input b</th>
</tr>
</thead>
<tbody>
<tr>
<td>{q_0}</td>
<td>{q_0, q_1}</td>
<td>{q_0}</td>
</tr>
<tr>
<td>{q_0, q_1}</td>
<td>{q_0, q_1}</td>
<td></td>
</tr>
<tr>
<td>{q_0, q_1}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{q_0, q_1}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{q_0, q_1}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{q_0, q_1}</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{q_0, q_1}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram:

- Start state: \(q_0\)
- Transitions:
  - \(q_0\) \(\rightarrow\) \(q_1\) on input \(a\)
  - \(q_1\) \(\rightarrow\) \(q_2\) on input \(b\)
  - \(q_2\) \(\rightarrow\) \(q_3\) on input \(a\)
  - \(q_3\) is a final state

Input alphabet: \(\Sigma\)
<table>
<thead>
<tr>
<th>State</th>
<th>( a )</th>
<th>( b )</th>
</tr>
</thead>
<tbody>
<tr>
<td>{q_0}</td>
<td>{q_0, q_1}</td>
<td>{q_0}</td>
</tr>
<tr>
<td>{q_0, q_1}</td>
<td>{q_0, q_1}</td>
<td></td>
</tr>
<tr>
<td>{q_0, q_1}</td>
<td>{q_0, q_1}</td>
<td></td>
</tr>
<tr>
<td>{q_0, q_1}</td>
<td>{q_0, q_1}</td>
<td></td>
</tr>
</tbody>
</table>

Diagram: A finite automaton with states \( q_0, q_1, q_2, q_3 \) and transitions labeled with symbols 'a' and 'b'. The initial state is \( q_0 \) and the transition symbol is Σ.
\[ \begin{array}{c|cc}
\Sigma & a & b \\
\hline
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & \{q_0, q_1\} & \\
\end{array} \]
\[
\begin{array}{|c|c|c|}
\hline
 & a & b \\
\hline
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & \{q_0, q_1\} & \{q_0, q_2\} \\
\hline
\end{array}
\]
\begin{align*}
\text{start} & \quad q_0 \quad \xrightarrow{a} \quad q_1 \quad \xrightarrow{b} \quad q_2 \quad \xrightarrow{a} \quad q_3 \\
\{q_0\} & \quad \{q_0, q_1\} \quad \{q_0\} \\
\{q_0, q_1\} & \quad \{q_0, q_1\} \quad \{q_0, q_2\}
\end{align*}
The given automaton starts at state $q_0$ and transitions as follows:

- From $q_0$ on input $a$, the automaton moves to state $q_1$.
- From $q_1$ on input $b$, the automaton moves to state $q_2$.
- From $q_2$ on input $a$, the automaton moves to state $q_3$.

The table below summarizes the transitions:

<table>
<thead>
<tr>
<th>Current State</th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${q_0}$</td>
<td>${q_0, q_1}$</td>
<td>${q_0}$</td>
</tr>
<tr>
<td>${q_0, q_1}$</td>
<td>${q_0, q_1}$</td>
<td>${q_0, q_2}$</td>
</tr>
<tr>
<td>${q_0, q_2}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
\begin{table}
\begin{tabular}{|c|c|c|}
\hline
 & $a$ & $b$ \\
\hline
$\{q_0\}$ & $\{q_0, q_1\}$ & $\{q_0\}$ \\
$\{q_0, q_1\}$ & $\{q_0, q_1\}$ & $\{q_0, q_2\}$ \\
$\{q_0, q_2\}$ & & \\
\hline
\end{tabular}
\end{table}
\[
\begin{array}{c|c|c}
    \text{state} & a & b \\
    \hline
    \{ q_0 \} & \{ q_0, q_1 \} & \{ q_0 \} \\
    \{ q_0, q_1 \} & \{ q_0, q_1 \} & \{ q_0, q_2 \} \\
    \{ q_0, q_2 \} & & \\
\end{array}
\]
\[
\begin{array}{c}
\begin{array}{c}
\text{start} \quad \Sigma \quad \Rightarrow \\
q_0 \quad a \quad q_1 \quad b \quad q_2 \quad a \quad q_3
\end{array}
\end{array}
\]

\[
\begin{array}{|c|c|c|}
\hline
& a & b \\
\hline
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & \{q_0, q_1\} & \{q_0, q_2\} \\
\{q_0, q_2\} & & \\
\hline
\end{array}
\]
\[ \Sigma \]

Start

\[ q_0 \]

\[ a \] \[ q_1 \]

\[ b \] \[ q_2 \]

\[ a \] \[ q_3 \]

\[
\begin{array}{c|cc}
\{ q_0 \} & \{ q_0, q_1 \} & \{ q_0 \} \\
\{ q_0, q_1 \} & \{ q_0, q_1 \} & \{ q_0, q_2 \} \\
\{ q_0, q_2 \} & & \\
\end{array}
\]
\[ \begin{array}{c|cc}
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & \{q_0, q_1\} & \{q_0, q_2\} \\
\{q_0, q_2\} & \{q_0, q_1, q_3\} & \\
\end{array} \]
\[
\begin{array}{c|c|c}
\text{State} & \text{Input} & \text{Output} \\
\hline
\{q_0\} & a & \{q_0, q_1\} \\
\{q_0, q_1\} & a & \{q_0, q_2\} \\
\{q_0, q_2\} & b & \{q_0, q_1, q_3\} \\
\end{array}
\]
<table>
<thead>
<tr>
<th>State</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>{q_0}</td>
<td>{q_0, q_1}</td>
<td>{q_0}</td>
</tr>
<tr>
<td>{q_0, q_1}</td>
<td>{q_0, q_1}</td>
<td>{q_0, q_2}</td>
</tr>
<tr>
<td>{q_0, q_2}</td>
<td>{q_0, q_1, q_3}</td>
<td></td>
</tr>
</tbody>
</table>

The diagram shows a transition graph with states \(q_0\), \(q_1\), \(q_2\), and \(q_3\). The transitions are labeled with symbols \(a\) and \(b\), and there is a self-loop on \(q_0\) labeled with \(\Sigma\).
A deterministic finite automaton (DFA) with states $q_0, q_1, q_2, q_3$ and alphabet $\Sigma = \{a, b\}$:

- Start state: $q_0$
- Transitions:
  - $q_0 \xrightarrow{a} q_1$
  - $q_1 \xrightarrow{b} q_2$
  - $q_2 \xrightarrow{a} q_3$
  - $q_3$ is a sink state

The transition table is:

<table>
<thead>
<tr>
<th>Current State</th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${q_0}$</td>
<td>${q_0, q_1}$</td>
<td>${q_0}$</td>
</tr>
<tr>
<td>${q_0, q_1}$</td>
<td>${q_0, q_1}$</td>
<td>${q_0, q_2}$</td>
</tr>
<tr>
<td>${q_0, q_2}$</td>
<td>${q_0, q_1, q_3}$</td>
<td></td>
</tr>
<tr>
<td>States</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
<td>------------</td>
</tr>
<tr>
<td>${q_0}$</td>
<td>${q_0, q_1}$</td>
<td>${q_0}$</td>
</tr>
<tr>
<td>${q_0, q_1}$</td>
<td>${q_0, q_1}$</td>
<td>${q_0, q_2}$</td>
</tr>
<tr>
<td>${q_0, q_2}$</td>
<td>${q_0, q_1, q_3}$</td>
<td>-</td>
</tr>
</tbody>
</table>

The diagram shows a finite automaton with states $q_0$, $q_1$, $q_2$, and $q_3$. The transitions are labeled with symbols $a$ and $b$. The start state is $q_0$.
<table>
<thead>
<tr>
<th>State</th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${q_0}$</td>
<td>${q_0, q_1}$</td>
<td>${q_0}$</td>
</tr>
<tr>
<td>${q_0, q_1}$</td>
<td>${q_0, q_1}$</td>
<td>${q_0, q_2}$</td>
</tr>
<tr>
<td>${q_0, q_2}$</td>
<td>${q_0, q_1, q_3}$</td>
<td>${q_0}$</td>
</tr>
</tbody>
</table>

The diagram shows a transition system with states $q_0$, $q_1$, $q_2$, and $q_3$. The alphabet $\Sigma$ includes both $a$ and $b$.

- From $q_0$, on input $a$, move to $q_1$.
- From $q_1$, on input $b$, move to $q_2$.
- From $q_2$, on input $a$, move to $q_3$ (a loop).

The table provides the next states for inputs $a$ and $b$ for each state.
\[
\begin{align*}
\Sigma & \quad \{ q_0 \} & \{ q_0, q_1 \} & \{ q_0 \} \\
\{ q_0, q_1 \} & \{ q_0, q_1 \} & \{ q_0, q_2 \} \\
\{ q_0, q_2 \} & \{ q_0, q_1, q_3 \} & \{ q_0 \}
\end{align*}
\]
\[
\begin{array}{c|cc|c}
\text{State} & \text{a} & \text{b} \\
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & \{q_0, q_1\} & \{q_0, q_2\} \\
\{q_0, q_2\} & \{q_0, q_1, q_3\} & \{q_0\} \\
\{q_0, q_1, q_3\} & & \\
\end{array}
\]
\[
\begin{array}{c|c|c}
\text{state} & \text{a} & \text{b} \\
\hline
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & \{q_0, q_1\} & \{q_0, q_2\} \\
\{q_0, q_2\} & \{q_0, q_1, q_3\} & \{q_0\} \\
\{q_0, q_1, q_3\} & & \\
\end{array}
\]
\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]

\[ q \]
\begin{itemize}
\item \begin{array}{c}
q_0 \\
\rightarrow \quad a \\
\rightarrow \quad b \\
\rightarrow \quad \Sigma \\
\rightarrow \quad \text{start}
\end{array}
\end{itemize}

\begin{array}{c|c|c}
 & a & b \\
\hline
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & \{q_0, q_1\} & \{q_0, q_2\} \\
\{q_0, q_2\} & \{q_0, q_1, q_3\} & \{q_0\} \\
\{q_0, q_1, q_3\} & & \\
\hline
\end{array}
\begin{tabular}{|c|c|c|}
\hline
& \(a\) & \(b\) \\
\hline
\{\(q_0\}\} & \{\(q_0, q_1\}\} & \{\(q_0\}\} \\
\{\(q_0, q_1\}\} & \{\(q_0, q_1\}\} & \{\(q_0, q_2\}\} \\
\{\(q_0, q_2\}\} & \{\(q_0, q_1, q_3\}\} & \{\(q_0\}\} \\
\{\(q_0, q_1, q_3\}\} & & \\
\hline
\end{tabular}
<table>
<thead>
<tr>
<th>State</th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${q_0}$</td>
<td>${q_0, q_1}$</td>
<td>${q_0}$</td>
</tr>
<tr>
<td>${q_0, q_1}$</td>
<td>${q_0, q_1}$</td>
<td>${q_0, q_2}$</td>
</tr>
<tr>
<td>${q_0, q_2}$</td>
<td>${q_0, q_1, q_3}$</td>
<td>${q_0}$</td>
</tr>
<tr>
<td>${q_0, q_1, q_3}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
\[
\begin{align*}
\Sigma & \rightarrow \{q_0, q_1\} \\
\{q_0\} & \rightarrow \{q_0, q_1\} \quad \{q_0\} \\
\{q_0, q_1\} & \rightarrow \{q_0, q_1\} \quad \{q_0, q_2\} \\
\{q_0, q_2\} & \rightarrow \{q_0, q_1, q_3\} \quad \{q_0\} \\
\{q_0, q_1, q_3\} & \rightarrow \{q_0, q_1\}
\end{align*}
\]
The given automaton has the following transitions:

- From $q_0$ on input $a$, the automaton moves to $q_1$.
- From $q_1$ on input $b$, the automaton moves to $q_2$.
- From $q_2$ on input $a$, the automaton moves to $q_3$.
- From $q_3$, the automaton loops back to $q_0$ on any input.

The table below summarizes the transitions for inputs $a$ and $b$:

<table>
<thead>
<tr>
<th>Current State</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>{q_0}</td>
<td>{q_0, q_1}</td>
<td>{q_0}</td>
</tr>
<tr>
<td>{q_0, q_1}</td>
<td>{q_0, q_1}</td>
<td>{q_0, q_2}</td>
</tr>
<tr>
<td>{q_0, q_2}</td>
<td>{q_0, q_1, q_3}</td>
<td>{q_0}</td>
</tr>
<tr>
<td>{q_0, q_1, q_3}</td>
<td>{q_0, q_1}</td>
<td></td>
</tr>
</tbody>
</table>
- **States and Transitions**:
  - **Start State**: $q_0$
  - **Final State**: $q_3$
  - Transitions:
    - $a$: $q_0 ightarrow q_1$
    - $b$: $q_1 ightarrow q_2$
    - $a$: $q_2 ightarrow q_3$
    - $\Sigma$: $q_0 ightarrow q_0$

- **Transition Table**:

<table>
<thead>
<tr>
<th>Current State</th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${ q_0 }$</td>
<td>${ q_0, q_1 }$</td>
<td>${ q_0 }$</td>
</tr>
<tr>
<td>${ q_0, q_1 }$</td>
<td>${ q_0, q_1 }$</td>
<td>${ q_0, q_2 }$</td>
</tr>
<tr>
<td>${ q_0, q_2 }$</td>
<td>${ q_0, q_1, q_3 }$</td>
<td>${ q_0 }$</td>
</tr>
<tr>
<td>${ q_0, q_1, q_3 }$</td>
<td>${ q_0, q_1 }$</td>
<td>${ q_0 }$</td>
</tr>
</tbody>
</table>
The graph shows a finite automaton with the following states:

- Start state: $q_0$
- States: $q_0$, $q_1$, $q_2$, $q_3$

The transitions are as follows:
- On input $a$, $q_0$ transitions to $q_1$.
- On input $b$, $q_1$ transitions to $q_2$.
- On input $a$, $q_2$ transitions to $q_3$.

The table below shows the transitions for inputs $a$ and $b$:

<table>
<thead>
<tr>
<th>Current State</th>
<th>$a$</th>
<th>$b$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${q_0}$</td>
<td>${q_0, q_1}$</td>
<td>${q_0}$</td>
</tr>
<tr>
<td>${q_0, q_1}$</td>
<td>${q_0, q_1}$</td>
<td>${q_0, q_2}$</td>
</tr>
<tr>
<td>${q_0, q_2}$</td>
<td>${q_0, q_1, q_3}$</td>
<td>${q_0}$</td>
</tr>
<tr>
<td>${q_0, q_1, q_3}$</td>
<td>${q_0, q_1}$</td>
<td></td>
</tr>
</tbody>
</table>
\[
\begin{array}{cccc}
\{ q_0 \} & \{ q_0, q_1 \} & \{ q_0 \} \\
\{ q_0, q_1 \} & \{ q_0, q_1 \} & \{ q_0, q_2 \} \\
\{ q_0, q_2 \} & \{ q_0, q_1, q_3 \} & \{ q_0 \} \\
\{ q_0, q_1, q_3 \} & \{ q_0, q_1 \} & \\
\end{array}
\]
\[
\begin{array}{c|cc|c}
\text{start} & a & b \\
\hline
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & \{q_0, q_1\} & \{q_0, q_2\} \\
\{q_0, q_2\} & \{q_0, q_1, q_3\} & \{q_0\} \\
\{q_0, q_1, q_3\} & \{q_0, q_1\} & \{q_0, q_2\} \\
\end{array}
\]
\[\Sigma \]

- Start with state \(q_0\)
  - \(\rightarrow a \rightarrow q_1\)
  - \(\rightarrow b \rightarrow q_2\)
  - \(\rightarrow a \rightarrow q_3\)

Input sequence: \(abaababa\)

Transition table:

<table>
<thead>
<tr>
<th>Input</th>
<th>(q_0)</th>
<th>(q_1)</th>
<th>(q_2)</th>
<th>(q_3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>({q_0, q_1})</td>
<td>({q_0, q_2})</td>
<td>({q_0, q_1, q_3})</td>
<td>({q_3})</td>
</tr>
<tr>
<td>(b)</td>
<td>({q_0, q_1})</td>
<td>({q_0, q_2})</td>
<td>({q_0, q_1, q_3})</td>
<td>({q_3})</td>
</tr>
</tbody>
</table>
The given automaton starts in state $q_0$. On input symbol $a$, it transitions to state $q_1$. On input symbol $b$, it transitions to state $q_2$. On input symbol $a$, it transitions to state $q_3$, which is a terminal state.

The input string is $aabbabaa$. The automaton starts in state $q_0$ and transitions through $q_1$, $q_2$, then back to $q_3$ after processing the entire string.

The corresponding set representation shows the following:

- Start state: $\{q_0\}$
- On input $a$: $\{q_0, q_1\}$
- On input $b$: $\{q_0, q_2\}$
- On input $a$: $\{q_0, q_1, q_3\}$
The Subset Construction

• This procedure for turning an NFA for a language $L$ into a DFA for a language $L$ is called the **subset construction**.
  • It’s sometimes called the **powerset construction**; it’s different names for the same thing!

• Intuitively:
  • Each state in the DFA corresponds to a set of states from the NFA.
  • Each transition in the DFA corresponds to what transitions would be taken in the NFA when using the massive parallel intuition.
  • The accepting states in the DFA correspond to which sets of states would be considered accepting in the NFA when using the massive parallel intuition.

• There’s an online *Guide to the Subset Construction* with a more elaborate example involving $\varepsilon$-transitions and cases where the NFA dies; check that for more details.
The Subset Construction

- In converting an NFA to a DFA, the DFA's states correspond to sets of NFA states.

- **Useful fact:** $|\mathcal{P}(S)| = 2^{|S|}$ for any finite set $S$.

- In the worst-case, the construction can result in a DFA that is exponentially larger than the original NFA.

- **Question to ponder:** Can you find a family of languages that have NFAs of size $n$, but no DFAs of size less than $2^n$?
Time-Out for Announcements!
Problem Set Four Grades

75th Percentile: 58 / 63 (92%)
50th Percentile: 55 / 63 (87%)
25th Percentile: 48 / 63 (77%)

Many of these grades are because folks forgot to list partners - please check to make sure you're getting credit for the work you're doing, and let us know if your partner forgot to add you.
Problem Set Six

• Problem Set Five was due at 2:30PM today.

• Problem Set Six goes out today. It’s due next Friday at 2:30PM.
  • Design DFAs and NFAs for a range of problems!
  • Explore formal language theory!
  • See some clever applications!
Midterm Revise-and-Resubmit

• The first midterm exam is now open for revise-and-resubmit.

• For each problem you’d like to submit a new answer to, file a regrade request and type in your new answer.

• The deadline to submit is Monday at 2:30 PM.
Extra Practice Problems 2

• Looking for more practice with functions, graphs, and induction? We’ve just posted Extra Practice Problems 2 to the course website.

• It’s a collection of 30 problems on these topics, plus others from PS3 – PS5.

• Solutions are available as well.
Your Questions
Your Questions

We’ll do this next time when we have a little more breathing room.
Back to CS103!
The Regular Languages
Regular Languages

- Let $L \subseteq \Sigma^*$ be a language.
- We say that $L$ is a regular language if there is a DFA $D$ where $\mathcal{L}(D) = L$.
- Equivalently, $L$ is a regular language if there is an NFA $N$ where $\mathcal{L}(N) = L$.

Key questions:
- What do the regular languages “feel” like?
- What properties do they have?
- What languages aren’t regular?
Closure Under Union

- If $L_1$ and $L_2$ are languages over the alphabet $\Sigma$, the language $L_1 \cup L_2$ is the language of all strings in at least one of the two languages.
- If $L_1$ and $L_2$ are regular languages, is $L_1 \cup L_2$?
Closure Under Union

- If $L_1$ and $L_2$ are languages over the alphabet $\Sigma$, the language $L_1 \cup L_2$ is the language of all strings in at least one of the two languages.

- If $L_1$ and $L_2$ are regular languages, is $L_1 \cup L_2$?
Closure Under Union

- If $L_1$ and $L_2$ are languages over the alphabet $\Sigma$, the language $L_1 \cup L_2$ is the language of all strings in at least one of the two languages.

- If $L_1$ and $L_2$ are regular languages, is $L_1 \cup L_2$ regular?
Closure Under Union

- If $L_1$ and $L_2$ are languages over the alphabet $\Sigma$, the language $L_1 \cup L_2$ is the language of all strings in at least one of the two languages.
- If $L_1$ and $L_2$ are regular languages, is $L_1 \cup L_2$?
Closure Under Union

- If $L_1$ and $L_2$ are languages over the alphabet $\Sigma$, the language $L_1 \cup L_2$ is the language of all strings in at least one of the two languages.

- If $L_1$ and $L_2$ are regular languages, is $L_1 \cup L_2$ regular?

Question to ponder: where have you seen this idea before?
Concatenation
String Concatenation

• If \( w \in \Sigma^* \) and \( x \in \Sigma^* \), the **concatenation** of \( w \) and \( x \), denoted \( wx \), is the string formed by tacking all the characters of \( x \) onto the end of \( w \).

• Example: if \( w = \text{quo} \) and \( x = \text{kka} \), the concatenation \( wx = \text{quokka} \).

• This is analogous to the + operator for strings in many programming languages.

• Some facts about concatenation:
  • The empty string \( \varepsilon \) is the **identity element** for concatenation:
    \[
    w\varepsilon = \varepsilon w = w
    \]
  • Concatenation is **associative**:
    \[
    wxy = w(xy) = (wx)y
    \]
Concatenation

- The *concatenation* of two languages $L_1$ and $L_2$ over the alphabet $\Sigma$ is the language

\[ L_1L_2 = \{ x \mid \exists w_1 \in L_1. \exists w_2 \in L_2. x = w_1w_2 \} \]
Concatenation Example

- Let $\Sigma = \{ a, b, ..., z, A, B, ..., Z \}$ and consider these languages over $\Sigma$:
  - $Noun = \{ \text{Puppy, Rainbow, Whale, ... } \}$
  - $Verb = \{ \text{Hugs, Juggles, Loves, ... } \}$
  - $The = \{ \text{The } \}$
  - $TheNounVerbTheNoun$ is
    - $\{ \text{ThePuppyHugsTheWhale, TheWhaleLovesTheRainbow, TheRainbowJugglesTheRainbow, ... } \}$
Concatenation

• The *concatenation* of two languages $L_1$ and $L_2$ over the alphabet $\Sigma$ is the language

\[ L_1L_2 = \{ x | \exists w_1 \in L_1. \exists w_2 \in L_2. x = w_1w_2 \} \]

• Two views of $L_1L_2$:
  • The set of all strings that can be made by concatenating a string in $L_1$ with a string in $L_2$.
  • The set of strings that can be split into two pieces: a piece from $L_1$ and a piece from $L_2$. 
Closure Under Concatenation

- If $L_1$ and $L_2$ are regular languages, is $L_1L_2$?
- Intuition – can we split a string $w$ into two strings $xy$ such that $x \in L_1$ and $y \in L_2$?
Closure Under Concatenation

- If $L_1$ and $L_2$ are regular languages, is $L_1L_2$?
- Intuition – can we split a string $w$ into two strings $xy$ such that $x \in L_1$ and $y \in L_2$?

![Machine for $L_1$](image1)

![Machine for $L_2$](image2)
Closure Under Concatenation

- If $L_1$ and $L_2$ are regular languages, is $L_1L_2$?

- Intuition – can we split a string $w$ into two strings $xy$ such that $x \in L_1$ and $y \in L_2$?

Machine for $L_1$  

Machine for $L_2$
Closure Under Concatenation

- If $L_1$ and $L_2$ are regular languages, is $L_1L_2$?
- Intuition – can we split a string $w$ into two strings $xy$ such that $x \in L_1$ and $y \in L_2$?

![Machine for $L_1$](image1)

![Machine for $L_2$](image2)

bookkeeper
Closure Under Concatenation

- If $L_1$ and $L_2$ are regular languages, is $L_1L_2$?
- Intuition – can we split a string $w$ into two strings $xy$ such that $x \in L_1$ and $y \in L_2$?

Machine for $L_1$

Machine for $L_2$

book
keeper
Closure Under Concatenation

- If $L_1$ and $L_2$ are regular languages, is $L_1L_2$?
- Intuition – can we split a string $w$ into two strings $xy$ such that $x \in L_1$ and $y \in L_2$?

**Idea:**
- Run a DFA/NFA for $L_1$ on $w$.
- Whenever it reaches an accepting state, optionally hand the rest of $w$ to a DFA/NFA for $L_2$.
- If the automaton for $L_2$ accepts the rest, $w \in L_1L_2$.
- If the automaton for $L_2$ rejects the remainder, the split was incorrect.
Closure Under Concatenation
Closure Under Concatenation

Machine for $L_1$
Closure Under Concatenation

Machine for $L_1$

Machine for $L_2$
Closure Under Concatenation

Machine for $L_1$

Machine for $L_2$
Closure Under Concatenation

Machine for $L_1$

Machine for $L_2$
Closure Under Concatenation

Machine for $L_1$

Machine for $L_2$

Machine for $L_1L_2$
The Kleene Star
Lots and Lots of Concatenation

- Consider the language $L = \{ \text{aa}, \text{b} \}$
- $LL$ is the set of strings formed by concatenating pairs of strings in $L$.
  
  $$\{ \text{aaaa, aab, baa, bb} \}$$
- $LLL$ is the set of strings formed by concatenating triples of strings in $L$.
  
  $$\{ \text{aaaaaa, aaab, aabaa, aabb, baaaa, baab, bbbaa, bbb} \}$$
- $LLLL$ is the set of strings formed by concatenating quadruples of strings in $L$.
  
  $$\{ \text{aaaaaaaa, aaaaaab, aaaaabaa, aaaaabb, aabaaaaa, aabaab, aabbaa, aabbb, baaaaaa, baaaaab, baabaa, baabb, bbaaaaa, bbaab, bbbbaa, bbbb} \}$$
Language Exponentiation

• We can define what it means to “exponentiate” a language as follows:

• $L^0 = \{ \varepsilon \}$
  - Intuition: The only string you can form by gluing no strings together is the empty string.
  - Notice that $\{\varepsilon\} \neq \emptyset$. Can you explain why?

• $L^{n+1} = LL^n$
  - Idea: Concatenating $(n+1)$ strings together works by concatenating $n$ strings, then concatenating one more.

• **Question to ponder:** Why define $L^0 = \{\varepsilon\}$?

• **Question to ponder:** What is $\emptyset^0$?
The Kleene Star
The Kleene Closure

• An important operation on languages is the **Kleene Closure**, which is defined as

\[ L^* = \{ w \in \Sigma^* \mid \exists n \in \mathbb{N}. w \in L^n \} \]

• Mathematically:

\[ w \in L^* \iff \exists n \in \mathbb{N}. w \in L^n \]

• Intuitively, \( L^* \) is the language all possible ways of concatenating zero or more strings in \( L \) together, possibly with repetition.

• **Question to ponder:** What is \( \emptyset^* \)?
The Kleene Closure

If \( L = \{ \text{a}, \text{bb} \} \), then \( L^* = \{ \)

\[ \varepsilon, \]

\[ \text{a}, \text{bb}, \]

\[ \text{aa}, \text{abb}, \text{bba}, \text{bbbb}, \]

\[ \text{aaa}, \text{aabb}, \text{abba}, \text{abbbb}, \text{bbaa}, \text{bbabb}, \text{bbbaa}, \text{bbbbbb}, \]

\[ \ldots \]

\} \]

Think of \( L^* \) as the set of strings you can make if you have a collection of stamps – one for each string in \( L \) – and you form every possible string that can be made from those stamps.
Idea: Can we convert an NFA for a language $L$ to an NFA for language $L^*$?
The Kleene Star

Machine for $L$
The Kleene Star

Machine for $L$
The Kleene Star

Machine for $L$
The Kleene Star

Machine for $L$
The Kleene Star

Machine for $L$
The Kleene Star

Machine for $L$

Machine for $L^*$
The Kleene Star

Question: Why add the new state out front? Why not just make the old start state accepting?
Closure Properties

- **Theorem:** If $L_1$ and $L_2$ are regular languages over an alphabet $\Sigma$, then so are the following languages:
  - $L_1 \cup L_2$
  - $L_1L_2$
  - $L_1^*$

- These properties are called **closure properties of the regular languages**.
In the Appendix

• The appendix to this slide deck contains
  • three examples of applying closure properties to concrete languages and how it relates to machines, and
  • two more closure properties beyond what we saw today.
• Take a look – there’s some really cool stuff here!
Next Time

- **Regular Expressions**
  - Building languages from the ground up!

- **Thompson’s Algorithm**
  - A UNIX Programmer in Theoryland.

- **Kleene’s Theorem**
  - From machines to programs!
Appendix 1: Closure Properties Applied
$L_1 = \{ w \in \{a, b\}^* \mid w \text{ has even length} \}$

$L_2 = \{ w \in \{a, b\}^* \mid w \text{ has length exactly three} \}$

Construct an NFA for $L_1 \cup L_2$. 
\[ L_1 = \{ \ w \in \{ a, b \}^* \mid \text{w has even length} \} \]
\[ L_2 = \{ \ w \in \{ a, b \}^* \mid \text{w has length exactly three} \} \]

Construct an NFA for \( L_1 \cup L_2 \).
$L_1 = \{ w \in \{a, b\}^* | w \text{ has even length} \}$

$L_2 = \{ w \in \{a, b\}^* | w \text{ has length exactly three} \}$

Construct an NFA for $L_1 \cup L_2$. 
$L_1 = \{ w \in \{a, b\}^* \mid w \text{ has even length} \}$

$L_2 = \{ w \in \{a, b\}^* \mid w \text{ has length exactly three} \}$

Construct an NFA for $L_1 \cup L_2$. 
$L_1 = \{ w \in \{a, b\}^* \mid w \text{ has even length} \}$

$L_2 = \{ w \in \{a, b\}^* \mid w \text{ has length exactly three} \}$

Construct an NFA for $L_1 \cup L_2$. 
\[ L_1 = \{ w \in \{a, b\}^* \mid w \text{ has even length} \} \]
\[ L_2 = \{ w \in \{a, b\}^* \mid w \text{ has length exactly three} \} \]

Construct an NFA for \( L_1 \cup L_2 \).
\( L_1 = \{ w \in \{a, b\}^* | w \text{ has even length } \} \)

\( L_2 = \{ w \in \{a, b\}^* | w \text{ has length exactly three } \} \)

Construct an NFA for \( L_1 \cup L_2 \).
$L_1 = \{ w \in \{ a, b \}^* \mid w \text{ has even length} \}$

$L_2 = \{ w \in \{ a, b \}^* \mid w \text{ has length exactly three} \}$

Construct an NFA for $L_1 \cup L_2$. 
\( L_1 = \{ w \in \{a, b\}^* | w \text{ has even length} \} \)
\( L_2 = \{ w \in \{a, b\}^* | w \text{ has length exactly three} \} \)

Construct an NFA for \( L_1 \cup L_2 \).
\[ L_1 = \{ w \in \{a, b\}^* \mid w \text{ has even length} \} \]

\[ L_2 = \{ w \in \{a, b\}^* \mid w \text{ has length exactly three} \} \]

Construct an NFA for \( L_1 \cup L_2 \).
Construct an NFA for $L_1 \cup L_2$. 

$L_1 = \{ w \in \{a, b\}^* \mid w \text{ has even length} \}$

$L_2 = \{ w \in \{a, b\}^* \mid w \text{ has length exactly three} \}$
\[ L_1 = \{ w \in \{a, b\}^* \mid w \text{ has even length} \} \]
\[ L_2 = \{ w \in \{a, b\}^* \mid w \text{ has length exactly three} \} \]

Construct an NFA for \( L_1 \cup L_2 \).
Let $L_1 = \{ w \in \{a, b\}^* \mid w \text{ has even length} \}$.

Let $L_2 = \{ w \in \{a, b\}^* \mid w \text{ has length exactly three} \}$.

Construct an NFA for $L_1 \cup L_2$.

$$L_1 = \{ w \in \{a, b\}^* \mid w \text{ has even length} \}$$

$$L_2 = \{ w \in \{a, b\}^* \mid w \text{ has length exactly three} \}$$
\[
L_1 = \{ w \in \{a, b\}^* \mid w \text{ has even length} \}
\]
\[
L_2 = \{ w \in \{a, b\}^* \mid w \text{ has length exactly three} \}
\]

Construct an NFA for \( L_1 \cup L_2 \).
Construct an NFA for $L_1 \cup L_2$.

$L_1 = \{ \ w \in \{a, b\}^* \ | \ w \text{ has even length} \ \}$

$L_2 = \{ \ w \in \{a, b\}^* \ | \ w \text{ has length exactly three} \ \}$
Construct an NFA for \(L_1 \cup L_2\).

\[
L_1 = \{ w \in \{a, b\}^* \mid w \text{ has even length} \}
\]

\[
L_2 = \{ w \in \{a, b\}^* \mid w \text{ has length exactly three} \}
\]

\[
\text{Construct an NFA for } L_1 \cup L_2.
\]
$$L_1 = \{ w \in \{a, b\}^* \mid w \text{ has even length} \}$$

$$L_2 = \{ w \in \{a, b\}^* \mid w \text{ has length exactly three} \}$$

Construct an NFA for $$L_1 \cup L_2$$. 
$L_1 = \{ w \in \{a, b\}^* \mid w \text{ has even length} \}$

$L_2 = \{ w \in \{a, b\}^* \mid w \text{ has length exactly three} \}$

Construct an NFA for $L_1 \cup L_2$. 
\( L_1 = \{ w \in \{a, b\}^* | \ w \ \text{has even length} \} \)
\( L_2 = \{ w \in \{a, b\}^* | \ w \ \text{has length exactly three} \} \)

Construct an NFA for \( L_1 \cup L_2 \).
$L_1 = \{ w \in \{a, b\}^* \mid w \text{ has even length} \}$

$L_2 = \{ w \in \{a, b\}^* \mid w \text{ has length exactly three} \}$

Construct an NFA for $L_1 \cup L_2$. 
$L_1 = \{ w \in \{a, b\}^* | w \text{ has even length} \}$

$L_2 = \{ w \in \{a, b\}^* | w \text{ has length exactly three} \}$

Construct an NFA for $L_1 \cup L_2$. 
\[ L_1 = \{ w \in \{a, b\}^* \mid w \text{ has even length} \} \]
\[ L_2 = \{ w \in \{a, b\}^* \mid w \text{ has length exactly three} \} \]

Construct an NFA for \( L_1 \cup L_2 \).
$L_1 = \{ w \in \{a, b\}^* \mid w \text{ has even length} \}$

$L_2 = \{ w \in \{a, b\}^* \mid w \text{ has length exactly three} \}$

Construct an NFA for $L_1 \cup L_2$. 
$L_1 = \{ w \in \{a, b\}^* | \text{w has even length} \}$

$L_2 = \{ w \in \{a, b\}^* | \text{w has length exactly three} \}$

Construct an NFA for $L_1 \cup L_2$. 
$L_1 = \{ \, w \in \{a, b\}^* \mid w \text{ has odd length} \, \}$

$L_2 = \{ \, w \in \{a, b\}^* \mid w \text{ has length exactly three} \, \}$

Construct an NFA for $L_1L_2$. 
$L_1 = \{ \ w \in \{a, b\}^* \ | \ w \text{ has odd length} \ \}$

$L_2 = \{ \ w \in \{a, b\}^* \ | \ w \text{ has length exactly three} \ \}$

Construct an NFA for $L_1L_2$. 

DFA for $L_1$
\[ L_1 = \{ \ w \in \{a, b\}^* \mid \text{w has odd length} \} \]
\[ L_2 = \{ \ w \in \{a, b\}^* \mid \text{w has length exactly three} \} \]

Construct an NFA for \( L_1L_2 \).
\[ L_1 = \{ w \in \{a, b\}^* \mid w \text{ has odd length} \} \]

\[ L_2 = \{ w \in \{a, b\}^* \mid w \text{ has length exactly three} \} \]

Construct an NFA for \( L_1 L_2 \).
\[ L_1 = \{ w \in \{a, b\}^* \mid w \text{ has odd length} \} \]
\[ L_2 = \{ w \in \{a, b\}^* \mid w \text{ has length exactly three} \} \]

Construct an NFA for \( L_1L_2 \).
\[ L_1 = \{ \ w \in \{a, b\}^* \mid w \text{ has odd length} \} \]
\[ L_2 = \{ \ w \in \{a, b\}^* \mid w \text{ has length exactly three} \} \]

Construct an NFA for \( L_1L_2 \).
Let $L_1 = \{ w \in \{a, b\}^* | w \text{ has odd length} \}$

Let $L_2 = \{ w \in \{a, b\}^* | w \text{ has length exactly three} \}$

Construct an NFA for $L_1L_2$. 

---

DFA for $L_1$

NFA for $L_2$
\( L_1 = \{ w \in \{ a, b \}^* \mid w \text{ has odd length} \} \)
\( L_2 = \{ w \in \{ a, b \}^* \mid w \text{ has length exactly three} \} \)

Construct an NFA for \( L_1L_2 \).
\[ L_1 = \{ w \in \{a, b\}^* \mid w \text{ has odd length} \} \]
\[ L_2 = \{ w \in \{a, b\}^* \mid w \text{ has length exactly three} \} \]

Construct an NFA for \( L_1L_2 \).
\( L_1 = \{ w \in \{ a, b \}^* \mid w \text{ has odd length} \} \)

\( L_2 = \{ w \in \{ a, b \}^* \mid w \text{ has length exactly three} \} \)

Construct an NFA for \( L_1L_2 \).
\[ L_1 = \{ w \in \{a, b\}^* \mid w \text{ has odd length} \} \]
\[ L_2 = \{ w \in \{a, b\}^* \mid w \text{ has length exactly three} \} \]

Construct an NFA for \( L_1 L_2 \).
\[ L_1 = \{ w \in \{a, b\}^* \mid w \text{ has odd length} \} \]
\[ L_2 = \{ w \in \{a, b\}^* \mid w \text{ has length exactly three} \} \]

Construct an NFA for \( L_1L_2 \).
\[ L_1 = \{ w \in \{a, b\}^* \mid w \text{ has odd length} \} \]
\[ L_2 = \{ w \in \{a, b\}^* \mid w \text{ has length exactly three} \} \]

Construct an NFA for \( L_1L_2 \).
$L_1 = \{ w \in \{a, b\}^* \mid w \text{ has odd length} \}$

$L_2 = \{ w \in \{a, b\}^* \mid w \text{ has length exactly three} \}$

Construct an NFA for $L_1L_2$. 
\[ L_1 = \{ w \in \{a, b\}_* \mid w \text{ has odd length} \} \]
\[ L_2 = \{ w \in \{a, b\}_* \mid w \text{ has length exactly three} \} \]

Construct an NFA for \( L_1 L_2 \).
Construct an NFA for $L_1L_2$. 

$L_1 = \{ w \in \{a, b\}^* | w \text{ has odd length} \}$

$L_2 = \{ w \in \{a, b\}^* | w \text{ has length exactly three} \}$
$L_1 = \{ \text{ } w \in \{ \text{a, b}\}^* \mid \text{ } w \text{ has odd length} \}$

$L_2 = \{ \text{ } w \in \{ \text{a, b}\}^* \mid \text{ } w \text{ has length exactly three} \}$

Construct an NFA for $L_1L_2$. 
\[ L_1 = \{ w \in \{a, b\}^* | w \text{ has odd length} \} \]
\[ L_2 = \{ w \in \{a, b\}^* | w \text{ has length exactly three} \} \]

Construct an NFA for \( L_1L_2 \).
$L_1 = \{ w \in \{a, b\}^* | w \text{ has odd length} \}$

$L_2 = \{ w \in \{a, b\}^* | w \text{ has length exactly three} \}$

Construct an NFA for $L_1L_2$. 
\[ L_1 = \{ w \in \{a, b\}^* \mid w \text{ has odd length} \} \]
\[ L_2 = \{ w \in \{a, b\}^* \mid w \text{ has length exactly three} \} \]

Construct an NFA for \( L_1L_2 \).
\[ L_1 = \{ w \in \{a, b\}^* \mid w \text{ has odd length} \} \]
\[ L_2 = \{ w \in \{a, b\}^* \mid w \text{ has length exactly three} \} \]

Construct an NFA for \( L_1L_2 \).
Let $L_1 = \{ w \in \{a, b\}^* | w \text{ has odd length} \}$ and $L_2 = \{ w \in \{a, b\}^* | w \text{ has length exactly three} \}$.

Construct an NFA for $L_1L_2$. 

---

DFA for $L_1$

NFA for $L_2$

---

$\begin{align*}
L_1 &= \{ w \in \{a, b\}^* | w \text{ has odd length} \} \\
L_2 &= \{ w \in \{a, b\}^* | w \text{ has length exactly three} \}
\end{align*}$

Construct an NFA for $L_1L_2$. 

---

---
$L_1 = \{ \, w \in \{a, b\}^* \mid w \text{ has odd length} \, \}$

$L_2 = \{ \, w \in \{a, b\}^* \mid w \text{ has length exactly three} \, \}$

Construct an NFA for $L_1L_2$. 
$L_1 = \{ w \in \{a, b\}^* \mid w \text{ has odd length} \}$

$L_2 = \{ w \in \{a, b\}^* \mid w \text{ has length exactly three} \}$

Construct an NFA for $L_1L_2$. 

\[ \begin{align*} 
\text{DFA for } L_1 & \quad \text{NFA for } L_2 \\
\begin{array}{c}
\text{start} \\
\times \\
\times \end{array} & \quad \begin{array}{c}
\varepsilon \\
\Sigma \\
\Sigma \\
\Sigma \quad \Sigma \\
\Sigma \quad \text{trap} \\
\Sigma \quad \text{trap} \\
\end{array}
\end{align*} \]
$L_1 = \{ \, w \in \{a, b\}^* \mid w \text{ has odd length} \, \}$

$L_2 = \{ \, w \in \{a, b\}^* \mid w \text{ has length exactly three} \, \}$

Construct an NFA for $L_1L_2$. 
$L = \{ w \in \{a, b\}^* \mid w \text{ has an odd number of } a\text{'s and an even number of } b\text{'s } \}$

Construct an NFA for $L^*$. 
\( L = \{ w \in \{a, b\}^* \mid w \text{ has an odd number of } a\text{'s and an even number of } b\text{'s } \} \)

Construct an NFA for \( L^* \).
\[ L = \{ w \in \{a, b\}^* | \text{w has an odd number of a's and an even number of b's} \} \]

Construct an NFA for \( L^* \).
$L = \{ w \in \{a, b\}^* \mid w \text{ has an odd number of } a\text{'s and an even number of } b\text{'s }\}$

Construct an NFA for $L^*$. 
$L = \{ w \in \{a, b\}^* | w \text{ has an odd number of } a'\text{'s and an even number of } b'\text{'s } \}$

Construct an NFA for $L^*$. 

DFA for $L$: 

```
start ε
ε       a
b       b
b       a
b       a
b       b
```

NFA for $L^*$:
$L = \{ w \in \{a, b\}^* \mid w \text{ has an odd number of } a\text{'s and an even number of } b\text{'s } \}$

Construct an NFA for $L^*$.
$L = \{ w \in \{a, b\}^* \mid w \text{ has an odd number of } a\text{'s and an even number of } b\text{'s } \}$

Construct an NFA for $L^*$. 

DFA for $L$ 

\[
\begin{array}{cccccccccc}
  & & & a & & & & & & b \\
  & & b & & b & & a & & b & & b \\
  & & & a & & & & & & b \\
  & & b & & b & & a & & b & & b \\
  & & & a & & & & & & b \\
  & & b & & b & & a & & b & & b \\
  & & & a & & & & & & b \\
  & & b & & b & & a & & b & & b \\
  & & & a & & & & & & b \\
  & & b & & b & & a & & b & & b \\
\end{array}
\]
$L = \{ w \in \{a, b\}^* \mid w \text{ has an odd number of a’s and an even number of b’s} \}$

Construct an NFA for $L^*$. 
$L = \{ \ w \in \{a, b\}^* \mid w \text{ has an odd number of } a\text{'s and an even number of } b\text{'s } \}$

Construct an NFA for $L^*$. 
L = \{ w \in \{a, b\}^* \mid w \text{ has an odd number of } a\text{'s and an even number of } b\text{'s} \}\)

Construct an NFA for L*.
Construct an NFA for $L^*$. 

$L = \{ w \in \{a, b\}^* | w \text{ has an odd number of } a \text{'s and an even number of } b \text{'s } \}$

Construct an NFA for $L^*$. 

DFA for $L$
$L = \{ \ w \in \{a, b\}^* \ | \ w \ \text{has an odd number of a's and an even number of b's} \ \}$

Construct an NFA for $L^*$. 
$L = \{ w \in \{a, b\}^* \mid w \text{ has an odd number of } a\text{'s and an even number of } b\text{'s } \}$

Construct an NFA for $L^*$. 

DFA for $L$
Construct an NFA for $L^*$.

$L = \{ w \in \{a, b\}^* \mid w \text{ has an odd number of } a \text{'s and an even number of } b \text{'s } \}$

Construct an NFA for $L^*$. 
\( L = \{ w \in \{a, b\}^* \mid w \text{ has an odd number of } a\text{'s and an even number of } b\text{'s} \} \)

Construct an NFA for \( L^* \).
\( L = \{ w \in \{a, b\}^* | w \text{ has an odd number of } a\text{'s and an even number of } b\text{'s } \}

Construct an NFA for \( L^* \).
$L = \{ w \in \{a, b\}^* \mid w \text{ has an odd number of } a\text{'s and an even number of } b\text{'s } \}$

Construct an NFA for $L^*$. 
\( L = \{ w \in \{ a, b \}^* \mid w \) has an odd number of \( a \)'s and an even number of \( b \)'s \} 

Construct an NFA for \( L^* \).
Construct an NFA for $L^*$. 

$L = \{ w \in \{a, b\}^* \mid w \text{ has an odd number of } a\text{'s and an even number of } b\text{'s } \}$

Construct an NFA for $L^*$. 
$L = \{ w \in \{a, b\}^* \mid w \text{ has an odd number of } a\text{'s and an even number of } b\text{'s} \}$

Construct an NFA for $L^*$. 
$L = \{ w \in \{a, b\}^* \mid w \text{ has an odd number of } a\text{'s and an even number of } b\text{'s } \}$

Construct an NFA for $L^*$. 
$$L = \{ w \in \{a, b\}^* \mid w \text{ has an odd number of } a \text{'s and an even number of } b \text{'s } \}$$

Construct an NFA for $L^*$. 
\[ L = \{ w \in \{a, b\}^* \mid w \text{ has an odd number of } a\text{'s and an even number of } b\text{'s} \} \]

Construct an NFA for \( L^* \).
$L = \{ w \in \{a, b\}^* \mid w \text{ has an odd number of } a\text{'s and an even number of } b\text{'s } \}$

Construct an NFA for $L^*$. 
Appendix 2: More Closure Properties
The Complement of a Language

- Given a language \( L \subseteq \Sigma^* \), the complement of that language (denoted \( \overline{L} \)) is the language of all strings in \( \Sigma^* \) that aren't in \( L \).
- Formally:

\[
\overline{L} = \Sigma^* - L
\]
The Complement of a Language

- Given a language \( L \subseteq \Sigma^* \), the complement of that language (denoted \( \overline{L} \)) is the language of all strings in \( \Sigma^* \) that aren't in \( L \).

- Formally:

\[
\overline{L} = \Sigma^* - L
\]
The Complement of a Language

- Given a language $L \subseteq \Sigma^*$, the complement of that language (denoted $\overline{L}$) is the language of all strings in $\Sigma^*$ that aren't in $L$.
- Formally:

$$\overline{L} = \Sigma^* - L$$
The Complement of a Language

- Given a language $L \subseteq \Sigma^*$, the complement of that language (denoted $\overline{L}$) is the language of all strings in $\Sigma^*$ that aren't in $L$.
- Formally:

$$\overline{L} = \Sigma^* - L$$

Good proofwriting exercise: prove $\overline{L} = L$ for any language $L$. 
Complementing Regular Languages

$L = \{ \ w \in \{a, b\}* \mid w \text{ contains } aa \text{ as a substring } \}$

![Diagram of state transition for L]

$\overline{L} = \{ \ w \in \{a, b\}* \mid w \text{ does not contain } aa \text{ as a substring } \}$

![Diagram of state transition for $\overline{L}$]
Complementing Regular Languages

\[ L = \{ w \in \{a, *, /\}* \mid w \text{ represents a C-style comment} \} \]
Complementing Regular Languages

\[ \overline{L} = \{ w \in \{a, *, /\}^* \mid w \text{ doesn't represent a C-style comment} \} \]
Complementing Regular Languages

\[ \overline{L} = \{ w \in \{a, *, /\}^* \mid w \text{ doesn't represent a C-style comment} \} \]
Closure Properties

- **Theorem:** If $L$ is a regular language, then $\overline{L}$ is also a regular language.
- As a result, we say that the regular languages are **closed under complementation**.

**Question to ponder:** are the nonregular languages closed under complementation?
Closure Under Intersection

- If $L_1$ and $L_2$ are languages over $\Sigma$, then $L_1 \cap L_2$ is the language of strings in both $L_1$ and $L_2$.

- Question: If $L_1$ and $L_2$ are regular, is $L_1 \cap L_2$ regular as well?
Closure Under Intersection

- If $L_1$ and $L_2$ are languages over $\Sigma$, then $L_1 \cap L_2$ is the language of strings in both $L_1$ and $L_2$.

- Question: If $L_1$ and $L_2$ are regular, is $L_1 \cap L_2$ regular as well?
Closure Under Intersection

- If $L_1$ and $L_2$ are languages over $\Sigma$, then $L_1 \cap L_2$ is the language of strings in both $L_1$ and $L_2$.

- Question: If $L_1$ and $L_2$ are regular, is $L_1 \cap L_2$ regular as well?

\[ \overline{L}_1 \quad \overline{L}_2 \]
Closure Under Intersection

- If \( L_1 \) and \( L_2 \) are languages over \( \Sigma \), then \( L_1 \cap L_2 \) is the language of strings in both \( L_1 \) and \( L_2 \).

- Question: If \( L_1 \) and \( L_2 \) are regular, is \( L_1 \cap L_2 \) regular as well?

\[
\overline{L_1} \cup \overline{L_2}
\]
Closure Under Intersection

- If $L_1$ and $L_2$ are languages over $\Sigma$, then $L_1 \cap L_2$ is the language of strings in both $L_1$ and $L_2$.
- Question: If $L_1$ and $L_2$ are regular, is $L_1 \cap L_2$ regular as well?

Hey, it's De Morgan's laws!