Finite Automata

Part Three
Recap from Last Time
Tabular DFAs

These stars indicate accepting states.
Since this is the first row, it's the start state.
If $D$ is a DFA, the **language of $D$**, denoted $\mathcal{L}(D)$, is \{ $w \in \Sigma^* \mid D$ accepts $w$ \}.

A language $L$ is called a **regular language** if there exists a DFA $D$ such that $\mathcal{L}(D) = L$. 
NFAs

• An **NFA** is a
  • **N**ondeterministic
  • **F**inite
  • **A**utomaton

• Can have missing transitions or multiple transitions defined on the same input symbol.

• Accepts if *any possible series of choices* leads to an accepting state.
\(\varepsilon\)-Transitions

- NFAs have a special type of transition called the \textbf{\(\varepsilon\)-transition}.
- An NFA may follow any number of \(\varepsilon\)-transitions at any time without consuming any input.
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

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

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

```
0 1 0 1 0 1 0
```
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

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

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

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

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

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

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

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

Start state

---

Diagram:

Start state -> a, c

a, c -> a, b

a, b -> a, c

a, c -> b, c

b, c -> a, c

---

Input sequence: a c c c a c c c
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 \ \} \quad \begin{array}{c}
\varepsilon \\
\varepsilon \\
\varepsilon \\
\varepsilon \\
\end{array}
$
Guess-and-Check

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

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

- Any language that can be accepted by a DFA can be accepted by an NFA.
- Why?
  - Every DFA essentially already is an NFA!
- **Question**: Can any language accepted by an NFA also be accepted by a DFA?
- Surprisingly, the answer is **yes**!
Thought Experiment:
How would you simulate an NFA in software?
\[ q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \]

\[ \Sigma \]

Start state: \( q_0 \)

Input sequence: \( abaaba \)
\[
\begin{array}{c|cc}
\{q_0\} & a & b \\
\hline
\{q_0, q_1\} & \{q_0, q_1\} & \{q_0\}
\end{array}
\]
<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></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram:

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

Symbols:

- $\Sigma$
- $q_0$
- $q_1$
- $q_2$
- $q_3$
\[ \Sigma \]

\[
\begin{array}{c|c|c}
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\hline
\{q_0\} & & \\
\hline
& & \\
\hline
& & \\
\end{array}
\]
The given automaton has the following transitions:

- Start state $q_0$ transitions to $q_1$ on input $a$.
- $q_1$ transitions to $q_2$ on input $b$.
- $q_2$ transitions to $q_3$ on input $a$.

The table below represents the states and their 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></td>
<td></td>
</tr>
<tr>
<td>State (q)</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></td>
<td></td>
</tr>
<tr>
<td>{q_0}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Start state: \(q_0\)
<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></td>
<td></td>
</tr>
<tr>
<td>{q_0, q_1}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram:

- Start state: \(q_0\)
- Transitions:
  - \(q_0\) on \(a\) goes to \(q_1\)
  - \(q_0\) on \(\Sigma\) (input alphabet) goes to \(q_1\)
  - \(q_1\) on \(b\) goes to \(q_2\)
  - \(q_2\) on \(a\) goes to \(q_3\)
  - \(q_3\) is a final state (sink)

Transition table:

- \(a\) transition:
  - \(q_0\) to \(q_1\)
  - \(q_0\) to \(q_1\)
  - \(q_0\) to \(q_1\)
- \(b\) transition:
  - \(q_0\) to \(q_1\)
  - \(q_0\) to \(q_1\)
  - \(q_0\) to \(q_1\)
\begin{array}{c|c|c}
\{ q_0 \} & \{ q_0, q_1 \} & \{ q_0 \} \\
\{ q_0, q_1 \} & \{ q_0, q_1 \} & \\
\{ 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_0, q_1} )</td>
<td>( {q_0, q_1} )</td>
<td>( {q_0, q_1} )</td>
</tr>
<tr>
<td>( {q_0, q_1} )</td>
<td>( {q_0, q_1} )</td>
<td>( {q_0, q_1} )</td>
</tr>
</tbody>
</table>

Diagram:
- Start state: \( q_0 \)
- Transitions:
  - \( q_0 \) \(\xrightarrow{a}\) \( q_1 \)
  - \( q_1 \) \(\xrightarrow{b}\) \( q_2 \)
  - \( q_2 \) \(\xrightarrow{a}\) \( q_3 \)
<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>${q_0}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The diagram and table represent a nondeterministic finite automaton (NFA). The table illustrates the state transitions for symbols 'a' and 'b'.

<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></td>
<td></td>
</tr>
</tbody>
</table>
The following table represents the transitions of the automaton:

<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></td>
<td></td>
</tr>
</tbody>
</table>

The diagram shows the states (q_0, q_1, q_2, q_3) and the transitions labeled with 'a' and 'b'.
<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></td>
<td></td>
</tr>
</tbody>
</table>

**Diagram**

- Start state: $q_0$
- Transitions:
  - $q_0 \xrightarrow{a} q_1$
  - $q_1 \xrightarrow{b} q_2$
  - $q_2 \xrightarrow{a} q_3$
  - $q_0 \xrightarrow{\Sigma}$ $q_0$ (loop)

**Table**

<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></td>
<td></td>
</tr>
</tbody>
</table>
$$q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3$$

Transition table:

<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>${q_0, q_2}$</td>
<td></td>
</tr>
</tbody>
</table>
A non-deterministic finite automaton (NFA) with transitions:

- **Start state:** $q_0$
- **Transitions:**
  - From $q_0$ on input $a$: $q_1$
  - From $q_1$ on input $b$: $q_2$
  - From $q_2$ on input $a$: $q_3$

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></td>
<td></td>
</tr>
</tbody>
</table>
### Transition Table

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

### Diagram

- **Start State**: $q_0$
- **Final State**: $q_3$
- Transitions:
  - $q_0 \xrightarrow{a} q_1$
  - $q_1 \xrightarrow{b} q_2$
  - $q_2 \xrightarrow{a} q_3$
  - $\Sigma \xrightarrow{\cdot} q_0$
The given automaton has the following transition table:

<table>
<thead>
<tr>
<th>State</th>
<th>Transition</th>
<th>State</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>{q_0}</td>
<td>(a) \rightarrow {q_0, q_1}</td>
<td>{q_0}</td>
<td>(b) \rightarrow {q_0}</td>
</tr>
<tr>
<td>{q_0, q_1}</td>
<td>(a) \rightarrow {q_0, q_1}</td>
<td>{q_0, q_1}</td>
<td>(b) \rightarrow {q_0, q_2}</td>
</tr>
<tr>
<td>{q_0, q_2}</td>
<td>(a) \rightarrow {q_0, q_1, q_3}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Additionally, the automaton has a start state \(q_0\) and a transition on \(\Sigma\) from \(q_0\) to itself.
\begin{itemize}
\item \[q_0\] \text{start} \quad a \quad \Sigma \quad b \quad a \quad \{q_3\}
\end{itemize}

\begin{table}
\begin{tabular}{|c|c|c|}
\hline
 \{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\hline
\{q_0, q_1\} & \{q_0, q_1\} & \{q_0, q_2\} \\
\hline
\{q_0, q_2\} & \{q_0, q_1, q_3\} & \{q_0\} \\
\hline
\end{tabular}
\end{table}
<table>
<thead>
<tr>
<th></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>

Diagram:

- Start state: $q_0$
- Transitions:
  - $q_0 \xrightarrow{a} q_1$
  - $q_1 \xrightarrow{b} q_2$
  - $q_2 \xrightarrow{a} q_3$
- Final state: $q_3$
\[ \begin{align*}
q_0 & \xrightarrow{a} q_1 \\
q_1 & \xrightarrow{b} q_2 \\
q_2 & \xrightarrow{a} q_3 \\
\text{start} & \xrightarrow{} q_0
\end{align*} \]

\[ \begin{array}{|c|c|c|}
\hline
 & a & b \\
\hline
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & \{q_0, q_1\} & \{q_0, q_2\} \\
\{q_0, q_2\} & \{q_0, q_1, q_3\} & \{q_0\} \\
\{q_0, q_1, q_3\} & & \\
\hline
\end{array} \]
\[
\begin{array}{c}
q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \\
\text{start}
\end{array}
\]

\[
\begin{array}{c|c|c}
\Sigma & a & b \\
\hline
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & \{q_0, q_1\} & \{q_0, q_2\} \\
\{q_0, q_2\} & \{q_0, q_1, q_3\} & \{q_0\} \\
\{q_0, q_1, q_3\} & & \\
\end{array}
\]
\[
\begin{array}{c|c|c}
\text{state} & \text{transitions} & \\
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & \{q_0, q_1\} & \{q_0, q_2\} \\
\{q_0, q_2\} & \{q_0, q_1, q_3\} & \{q_0\} \\
\{q_0, q_1, q_3\} & \{q_0, q_1\} & \end{array}
\]
\[
\begin{array}{c}
\begin{array}{c}
\text{start} \\
q_0 \\
q_1 \\
q_2 \\
q_3
\end{array}
\end{array}
\begin{array}{c}
\Sigma \\
a \\
b \\
a
\end{array}
\]

\[
\begin{array}{|c|c|c|}
\hline
\text{state} & a & b \\
\hline
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & \{q_0, q_1\} & \{q_0, q_2\} \\
\{q_0, q_2\} & \{q_0, q_1, q_3\} & \{q_0\} \\
\{q_0, q_1, q_3\} & \{q_0, q_1\} & \\
\hline
\end{array}
\]
\begin{align*}
\Sigma & \xrightarrow{a} q_1 \\
& \xrightarrow{b} q_2 \\
& \xrightarrow{a} q_3
\end{align*}

<table>
<thead>
<tr>
<th>{q_0}</th>
<th>{q_0, q_1}</th>
<th>{q_0}</th>
</tr>
</thead>
<tbody>
<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>
<tr>
<td>State</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>{q_0}</td>
</tr>
<tr>
<td>{q_0, q_1, q_3}</td>
<td>{q_0, q_1}</td>
<td>{q_0, q_2}</td>
</tr>
</tbody>
</table>

**Diagram:**

- Start state: \(q_0\)
- States: \(q_0, q_1, q_2, q_3\)
- Transitions:
  - \(a\) from \(q_0\) to \(q_1\)
  - \(b\) from \(q_1\) to \(q_2\)
  - \(a\) from \(q_2\) to \(q_3\)
  - \(\Sigma\) loop from \(q_0\) to \(q_0\)
The given figure represents a deterministic finite automaton (DFA) with the following states and transitions:

- **States:** $q_0$, $q_1$, $q_2$, $q_3$
- **Transitions:**
  - From $q_0$, on input $a$, moves to $q_1$.
  - From $q_1$, on input $b$, moves to $q_2$.
  - From $q_2$, on input $a$, moves to $q_3$.
  - From $q_3$, on input $a$, moves to $q_3$.

The automaton starts at state $q_0$ and accepts strings that end in $q_3$.
Formal Description:

**Transition Table:**

<table>
<thead>
<tr>
<th>Current State</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>{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>{<em>q_0, q_1, q_3</em>}</td>
<td>{q_0, q_1}</td>
<td>{q_0, q_2}</td>
</tr>
</tbody>
</table>

**Diagram:**

- Start state: \{q_0\}
- Transitions:
  - \{q_0\} \xrightarrow{a} \{q_0, q_1\}
  - \{q_0, q_1\} \xrightarrow{b} \{q_0, q_2\}
  - \{q_0, q_2\} \xrightarrow{a} \{*q_0, q_1, q_3*\}
  - \{*q_0, q_1, q_3*\} \xrightarrow{b} \{q_0, q_1, q_3\}
  - \{q_0\} \xrightarrow{\Sigma} \{q_0\}

End state: \{*q_0, q_1, q_3*\}
The diagram shows a finite automaton with states labeled $q_0$, $q_1$, $q_2$, and $q_3$. The transitions are labeled with symbols $a$ and $b$, and the start state is $q_0$. The input string is $abaaba$. The automaton moves through states as follows:

- Start at $q_0$.
- On $a$, move to $q_1$.
- On $b$, move to $q_2$.
- On $a$, move to $q_3$.

The automaton accepts the input string if it ends in a final state. In this case, it ends in $q_3$, indicating acceptance.
The Subset Construction

- This procedure for turning an NFA for a language $L$ into a DFA for a language $L$ is called the \textit{subset construction}.
  - It’s sometimes called the \textit{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 \textit{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 \(2^n\)?
A language $L$ is called a *regular language* if there exists a DFA $D$ such that $\mathcal{L}(D) = L$. 


**Theorem:** A language $L$ is regular if and only if there is some NFA $N$ such that $\mathcal{L}(N) = L$.

**Proof Sketch:** Pick a language $L$. First, assume $L$ is regular. That means there’s a DFA $D$ where $\mathcal{L}(D) = L$. Every DFA is “basically” an NFA, so there’s an NFA $(D)$ whose language is $L$.

Next, assume there’s an NFA $N$ such that $\mathcal{L}(N) = L$. Using the subset construction, we can build a DFA $D$ where $\mathcal{L}(N) = \mathcal{L}(D)$. Then we have that $\mathcal{L}(D) = L$, so $L$ is regular. ■-ish
Why This Matters

• We now have two perspectives on regular languages:
  • Regular languages are languages accepted by DFAs.
  • Regular languages are languages accepted by NFAs.
• We can now reason about the regular languages in two different ways.
Properties of Regular Languages
The Union of Two Languages

- 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?
The Union of Two Languages

- 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?
The Union of Two Languages

- 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 $L_1 \cup L_2$?
The Union of Two Languages

- 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 $L_1 \cup L_2$?
The Union of Two Languages

- 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?
The Intersection of Two Languages

- 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?
The Intersection of Two Languages

- 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?

![Diagram of the intersection of two languages $L_1$ and $L_2$]
The Intersection of Two Languages

- 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?
The Intersection of Two Languages

- 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?

\[ \overline{L_1} \cup \overline{L_2} \]
The Intersection of Two Languages

• 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?
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

$$L_1L_2 = \{ wx \in \Sigma^* \mid w \in L_1 \land x \in L_2 \}$$
Concatenation Example

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

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

\[ L_1 L_2 = \{ wx \in \Sigma^* \mid w \in L_1 \land x \in L_2 \} \]

• Two views of $L_1 L_2$:
  • 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 $L_2$. 
Concatenating Regular Languages

- 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 $y \in L_2$?
Concatenating Regular Languages

- 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 $L_2$
Concatenating Regular Languages

- 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
Concatenating Regular Languages

- 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 $L_2$

bookkeeper
Concatenating Regular Languages

• 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
Concatenating Regular Languages

- 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.
Concatenating Regular Languages
Concatenating Regular Languages

Machine for $L_1$
Concatenating Regular Languages

Machine for $L_1$

Machine for $L_2$
Concatenating Regular Languages

Machine for $L_1$

Machine for $L_2$

ε
Concatenating Regular Languages

Machine for $L_1$

Machine for $L_2$
Concatenating Regular Languages

Machine for $L_1$

Machine for $L_2$

Machine for $L_1L_2$
Lots and Lots of Concatenation

• Consider the language $L = \{ \text{aa, b} \}$
• $LL$ 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 $L$.
  \[
  \{ \text{aaaaaa, aaab, aaba, aabb, baaaa, baab, bbaa, bbb} \}
  \]
• $LLLL$ is the set of strings formed by concatenating quadruples of strings in $L$.
  \[
  \{ \text{aaaaaaaa, aaaaaab, aaaaaba, aaabba, aabaaaa, aabaab, aabbaa, aababb, baaaaaa, baaaaab, baabaa, baaabb, bbaaaa, bbaaab, bbbbaa, bbbb} \}
  \]
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 \( L = \{ a, bb \} \), then \( L^* = \{ \)

\( \varepsilon, \)

\( a, bb, \)

\( aa, abb, bba, bbbb, \)

\( aaa, aabb, abba, abbbb, bbbaa, bbabb, bbbba, bbbbbbb, \)

\(...\)

\( \} \)

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.
Reasoning about Infinity

• If $L$ is regular, is $L^*$ necessarily regular?

⚠ A Bad Line of Reasoning: ⚠

• $L^0 = \{ \varepsilon \}$ is regular.
• $L^1 = L$ is regular.
• $L^2 = LL$ is regular
• $L^3 = L(LL)$ is regular
• ...

• Regular languages are closed under union.
• So the union of all these languages is regular.
Reasoning about Infinity
Reasoning about Infinity
Reasoning about Infinity
Reasoning about Infinity
Reasoning about Infinity
Reasoning about Infinity
Reasoning about Infinity
Reasoning about Infinity

\[ x \neq 2x \]
Reasoning About the Infinite

• If a series of finite objects all have some property, the “limit” of that process does not necessarily have that property.

• In general, it is not safe to conclude that some property that always holds in the finite case must hold in the infinite case.
  • (This is why calculus is interesting).

• So our earlier argument ($L^* = L^0 \cup L^1 \cup ...$) isn’t going to work.

• We need a different line of reasoning.
**Idea:** Can we directly convert an NFA for language $L$ to an NFA for language $L^*$?
The Kleene Star

Machine for $L$
The Kleene Star
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:
  - $\overline{L_1}$
  - $L_1 \cup L_2$
  - $L_1 \cap L_2$
  - $L_1L_2$
  - $L_1^*$

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