Recap from Last Time
Tabular DFAs

These stars indicate accepting states.

<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>
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 $\{ w \in \Sigma^* \mid D \text{ accepts } w \}$.

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.
ε-Transitions

- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.
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.
Just how powerful are NFAs?
New Stuff!
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!
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?
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*!
NFAs and DFAs

• **Question:** Can any language accepted by an NFA also be accepted by a DFA?

• Surprisingly, the answer is **yes**!
  
  • To prove this, we need to:
    - Pick an arbitrary NFA
    - Describe how we would construct a DFA with the same language (in a generalizable way)
    - For the next few slides, we’ll ponder how to approach that...
Thought Experiment:
How would you simulate an NFA in software?
\[ \Sigma \]

Start

\[ q_0 \rightarrow a \rightarrow q_1 \rightarrow b \rightarrow q_2 \rightarrow a \rightarrow q_3 \]

\[ a \ b \ a \ b \ a \ b \ a \]
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 $a$: $q_3$

Input alphabet: $\Sigma$

Input string: $abaaba$
\[
\Sigma
\]

Start state: \(q_0\)

\(q_0 \rightarrow q_1\) on input \(a\)

\(q_1 \rightarrow q_2\) on input \(b\)

\(q_2 \rightarrow q_3\) on input \(a\)

Transitions:
- \(q_0 \rightarrow q_1\) on \(a\)
- \(q_1 \rightarrow q_2\) on \(b\)
- \(q_2 \rightarrow q_3\) on \(a\)

Input string: \(aababa\)

Accepting state: \(q_3\)
\[ q_3 \xrightarrow{q_2} q_1 \xrightarrow{q_0} q_2 \xrightarrow{\Sigma} q_3 \]

Start

\[ \begin{array}{c}
\text{a} \\
\text{b}
\end{array} \]

\[ \begin{array}{ccccc}
\text{a} & \text{b} & \text{a} & \text{b} & \text{a}
\end{array} \]
\[
\begin{align*}
\sum & \quad q_3 \\
q_0 & \quad a \quad q_1 \\
& \quad b \quad q_2 \\
& \quad a \quad q_3 \\
\ldots & \quad ? \quad ? \quad ? \quad ? \quad a \quad ? \quad ? \quad ? \quad ? \quad ? \quad \ldots
\end{align*}
\]
\[ \Sigma \]

Start

\[ q_0 \] → a \[ q_1 \] → b \[ q_2 \] → a \[ q_3 \] →

\[ \Sigma \]

\[ q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \]

\[
\begin{array}{|c|c|}
\hline
\text{State} & \text{Input} \\
\hline
\{q_0\} & \{q_0, q_1\} \\
\hline
\end{array}
\]
\[
\begin{array}{c|cc}
 & a & b \\
\hline
\{q_0\} & \{q_0, q_1\} & \\
\hline
\end{array}
\]
The given diagram represents a state transition graph with the following states and transitions:

- States: \( q_0, q_1, q_2, q_3 \)
- Transitions:
  - From \( q_0 \) on input \( a \) to \( q_1 \)
  - From \( q_1 \) on input \( b \) to \( q_2 \)
  - From \( q_2 \) on input \( a \) to \( q_3 \)

The table below shows the transitions for inputs \( a \) and \( b \):

<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></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>
\[ \sum \]

\[ a \text{ \rightarrow } q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \]

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

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>{q_0}</td>
<td>{q_0, q_1}</td>
</tr>
</tbody>
</table>
\[ \Sigma \]

<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>
</tbody>
</table>

- Start state: \( q_0 \)
- Transitions:
  - \( q_0 \rightarrow q_1 \) on \( a \)
  - \( q_1 \rightarrow q_2 \) on \( b \)
  - \( q_2 \rightarrow q_3 \) on \( a \)
The diagram represents a finite automaton with states $q_0$, $q_1$, $q_2$, and $q_3$. The transitions are as follows:

- 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 $q_3$.
- From $q_3$ (a final state), any input returns to $q_3$.

The table below shows the transition function for inputs $a$ 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></td>
<td></td>
</tr>
</tbody>
</table>

The alphabet $\Sigma$ includes $a$ and $b$. The start state is $q_0$. The diagram also shows the transitions for each state.
\[
\begin{array}{c|c|c}
& a & b \\
\hline
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & & \\
\hline
\end{array}
\]
\[ \Sigma \]

Transition 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></td>
<td></td>
</tr>
</tbody>
</table>
\[ \begin{array}{c|cc}
\{q_0\} & a & b \\
\{q_0, q_1\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & & \\
\end{array} \]
\( \Sigma \)

State transition diagram:

- Start state: \( q_0 \)
- Transition on \( a \) from \( q_0 \) to \( q_1 \)
- Transition on \( b \) from \( q_1 \) to \( q_2 \)
- Transition on \( a \) from \( q_2 \) to \( q_3 \)

Transition 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></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>${q_0}$</td>
</tr>
</tbody>
</table>

- **Start State:** $q_0$
- Transitions:
  - $a$: $q_0 \rightarrow q_1$
  - $b$: $q_1 \rightarrow q_2$
  - $a$: $q_2 \rightarrow q_3$
\[ \sum \]

\[
\begin{array}{ccc}
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & \{q_0, q_1\} & \\
\end{array}
\]
\[
\begin{array}{ccc}
\Sigma & \rightarrow & \{q_0, q_1\} \\
\{q_0\} & \rightarrow & \{q_0, q_1\} \\
\{q_0, q_1\} & \rightarrow & \{q_0, q_1\} \\
\end{array}
\]
<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>
</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:** $\Sigma$
\[ q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \]

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

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

<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>
\[ \Sigma \]

```
\begin{align*}
\{q_0\} & \rightarrow \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & \rightarrow \{q_0, q_1\} & \{q_0, q_2\} \\
\{q_0, q_2\} & \rightarrow \quad & \\
\end{align*}
```
\[ \sum \]

\[
\begin{array}{c}
q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3
\end{array}
\]

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

<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>{q_0, q_2}</td>
</tr>
<tr>
<td>{q_0, q_2}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Start state: \(q_0\)
\[ a \{ q_0 \} a \{ q_0, q_1 \} a \{ q_0, q_1 \} a \{ q_0, q_2 \} \]
The given DFA has the following states and transitions:

- **States:** $q_0, q_1, q_2, q_3$
- **Start State:** $q_0$
- **Accepting State:** $q_3$

### 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></td>
<td></td>
</tr>
</tbody>
</table>
\[
\begin{array}{c}
\Sigma \\
\end{array}
\]

\[
\begin{array}{c}
\text{start} \\
q_0 \\
\text{a} \\
q_1 \\
b \\
q_2 \\
\text{a} \\
q_3 \\
\end{array}
\]

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

\[ q_2 \]

\[ q_1 \]

\[ q_0 \]

\[ \Sigma \]

The diagram shows a finite automaton with states \( q_0, q_1, q_2, q_3 \) and transitions labeled with symbols \( a \) and \( b \). The transitions are as follows:

- From \( q_0 \) on input \( a \) to \( q_1 \)
- From \( q_1 \) on input \( b \) to \( q_2 \)
- From \( q_2 \) on input \( a \) to \( q_3 \)

The table below lists the transitions for each state and input:

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

The automaton starts at \( q_0 \) and accepts strings based on the transitions and inputs.
\begin{align*}
\Sigma & \rightarrow a \rightarrow q_0 \rightarrow a \rightarrow q_1 \rightarrow b \rightarrow q_2 \rightarrow a \rightarrow q_3 \\
\{q_0\} & \rightarrow \{q_0, q_1\} \rightarrow \{q_0\} \\
\{q_0, q_1\} & \rightarrow \{q_0, q_1\} \rightarrow \{q_0, q_2\} \\
\{q_0, q_2\} & \rightarrow \{q_0, q_1, q_3\} \\
\{q_0, q_1, q_3\} & \rightarrow 
\end{align*}
The diagram represents a finite automaton with states labeled $q_0$, $q_1$, $q_2$, and $q_3$. The input alphabet, denoted by $\sum$, consists of two symbols: $a$ and $b$. The transitions are as follows:

- From $q_0$: with $a$, the transition is to $q_1$; with $b$, the transition is to $q_2$.
- From $q_1$: with $a$, the transition is to $q_3$.
- From $q_2$: with $a$, the transition is to $q_3$.

The table below shows the transition function $\delta$:

<table>
<thead>
<tr>
<th>Transition</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>
\Sigma

\begin{array}{ccc}
\{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}
\[ q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \]

<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></td>
</tr>
<tr>
<td>({q_0, q_2})</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
\( \Sigma \)

```
\begin{array}{c|cc}
   & 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 \} \\
\end{array}
```
\[ \Sigma \]

Start: \( q_0 \)

- \( q_0 \) to \( q_1 \) on input \( a \)
- \( q_1 \) to \( q_2 \) on input \( b \)
- \( q_2 \) to \( q_3 \) on input \( a \)

<table>
<thead>
<tr>
<th>( q )</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>
\[\begin{array}{c|cc}
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & \{q_0, q_1\} & \{q_0, q_2\} \\
\{q_0, q_2\} & \{q_0, q_1, q_3\} & \{q_0\} \\
\{q_0, q_1, q_3\} & & \end{array}\]
\[ \Sigma \]

```
start
\[
\begin{array}{ccc}
q_0 & \xrightarrow{a} & q_1 \\
& \xrightarrow{b} & q_2 \\
& \xrightarrow{a} & q_3
\end{array}
\]
```

<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>
\[
\begin{array}{c|c|c}
\Sigma & a & b \\
\hline
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & \{q_0, q_1\} & \{q_0, q_2\} \\
\{q_0, q_2\} & \{q_0, q_1, q_3\} & \{q_0\} \\
\{q_0, q_1, q_3\} & & \\
\end{array}
\]
\[
\begin{array}{c|c|c}
\text{State} & \text{a} & \text{b} \\
\hline
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & \{q_0, q_1\} & \{q_0, q_2\} \\
\{q_0, q_2\} & \{q_0, q_1, q_3\} & \{q_0\} \\
\{q_0, q_1, q_3\} & & \\
\end{array}
\]
\[ \sum \]

\[
\begin{array}{c}
\text{start} \\
\rightarrow \\
q_0 \\
\rightarrow a \\
q_1 \\
\rightarrow b \\
q_2 \\
\rightarrow a \\
q_3 \\
\end{array}
\]

<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>{q_0}</td>
</tr>
<tr>
<td>{q_0, q_1, q_3}</td>
<td>{q_0, q_1}</td>
<td></td>
</tr>
</tbody>
</table>
\[ \Sigma \]

start

\[
\begin{array}{c}
q_0 \\
q_1 \\
q_2 \\
q_3
\end{array}
\]

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

## 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_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>
\[
\begin{array}{c}
\Sigma \\
\text{start} \\
q_0 \\
q_1 \\
q_2 \\
q_3 \\
\end{array}
\]

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

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

\[
\begin{array}{ccc}
q_0 & a & q_1 \\
\downarrow & a & \downarrow b \\
q_1 & b & q_2 \\
\downarrow & a & \downarrow \\
q_2 & & q_3
\end{array}
\]

<table>
<thead>
<tr>
<th>State Sets</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>
A finite automaton with the following transitions:

- Start state: $q_0$
- Transitions:
  - From $q_0$: $a$ to $q_1$, $b$ to $q_2$
  - From $q_1$: $a$ to $q_3$
  - From $q_2$: $a$ to $q_0$

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>${q_0, q_1, q_3}$</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>
\[
\begin{array}{c|c|c}
\Sigma & a & b \\
\hline
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & \{q_0, q_1\} & \{q_0, q_2\} \\
\{q_0, q_2\} & \{q_0, q_1, q_3\} & \{q_0\} \\
\{q_0, q_1, q_3\} & \{q_0, q_1\} & \{q_0, q_2\} \\
\end{array}
\]
Start

\[
q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3
\]

\[
\Sigma
\]

\[
\{ q_0 \}
\]

\[
\{ q_0, q_1 \}
\]

\[
\{ q_0, q_2 \}
\]

\[
\{ q_0, q_1, q_3 \}
\]

Input:

\[
a b a a a b b a
\]
\[
\Sigma
\]

```
start
\{q_0\}

{q_0, q_1}

{q_0, q_2}

\{q_0, q_1, q_3\}
```

```
\begin{array}{ccccccc}
  a & b & a & a & a & b & a
\end{array}
```

```
q_0 \rightarrow a \rightarrow q_1
q_1 \rightarrow b \rightarrow q_2
q_2 \rightarrow a \rightarrow q_3
```

```
\text{start}
\uparrow
```
\[
\Sigma
\]

\[
\{q_0\}
\]

\[
\{q_0, q_1\}
\]

\[
\{q_0, q_2\}
\]

\[
\{q_0, q_1, q_3\}
\]

\[
\begin{array}{c}
\text{start} \\
q_0 \\
\downarrow a \\
q_1 \\
\downarrow b \\
q_2 \\
\downarrow a \\
q_3 \\
\end{array}
\]
\[
\Sigma
\]

Transition diagram:

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

Input string:

\[
\text{a b a a a b a a a}
\]
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 ε-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 $\mathcal{L}(D) = L$. 
An Important Result

**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 $L_1 \cup L_2$? 
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 $L_1 \cup L_2$ also 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 $L_1 \cup L_2$?

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

Hey, it's De Morgan's laws!
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 $\Sigma = \{ \text{a, b, ..., z, A, B, ..., Z} \}$ and consider these languages over $\Sigma$:
  - $\textbf{Noun} = \{ \text{Puppy, Rainbow, Whale, ...} \}$
  - $\textbf{Verb} = \{ \text{Hugs, Juggles, Loves, ...} \}$
  - $\textbf{The} = \{ \text{The} \}$

- The language $\textbf{TheNounVerbTheNoun}$ is
  $\{ \text{ThePuppyHugsTheWhale, TheWhaleLovesTheRainbow, TheRainbowJugglesTheRainbow, ...} \}$
Concatenation

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

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

- 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 $L_2$.

This is closely related to, but different than, the Cartesian product.

**Question to ponder:** In what ways are concatenations similar to Cartesian products? In what ways are they different?
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:
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 remainder, \( w \in L_1L_2 \).
  - If the automaton for \( L_2 \) rejects the remainder, the split was incorrect.
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 remainder, $w \in L_1L_2$.
  - If the automaton for $L_2$ rejects the remainder, the split was incorrect.
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 remainder, $w \in L_1L_2$.
  - If the automaton for $L_2$ rejects the remainder, the split was incorrect.
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 remainder, $w \in L_1L_2$.
  - If the automaton for $L_2$ rejects the remainder, the split was incorrect.

Machine for $L_1$  
Machine for $L_2$
Concatenating Regular Languages

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

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, $w \in L_1 L_2$.
- 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 $L_2$
Concatenating Regular Languages

Machine for $L_1$

Machine for $L_2$
Concatenating Regular Languages

Machine for $L_1$

Machine for $L_2$

Machine for $L_1 L_2$
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 $L$.
  
  $\{\text{ aaaaaa, aaaaab, aabaa, aabb, baaaa, baab, bbaa, bbb}\}$
• $LLLL$ is the set of strings formed by concatenating quadruples of strings in $L$.
  
  $\{\text{ aaaaaaaaa, aaaaaaab, aaaaabaa, aaaaabb, aabaaaa, aabaab, aabbaa, aabbb, baaaaaa, baaaaab, baabaa, baabb, bbaaaa, bbaaab, bbbaa, bbbb}\}$
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 $L = \{ \text{a, bb} \}$, then $L^* = \{ \varepsilon, \text{a, bb, aa, abb, bba, bbbb, aaaa, aabb, abba, abbb, bbab, bbbba, bbbbbbb, ...} \}$

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

$\chi \{ \chi \neq 2\chi \}$
Reasoning about Infinity

0.9 < 1
Reasoning about Infinity

0.99 < 1
Reasoning about Infinity

$0.999 < 1$
Reasoning about Infinity

0.9999 < 1
Reasoning about Infinity

0.99999 < 1
Reasoning about Infinity

0.99999\bar{9} \leq 1
Reasoning about Infinity

0 is finite
Reasoning about Infinity

1 is finite
Reasoning about Infinity

2 is finite
Reasoning about Infinity

3 is finite
Reasoning about Infinity

4 is finite
Reasoning about Infinity

$\infty$ is finite

^ not
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

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:
  - $\overline{L}_1$
  - $L_1 \cup L_2$
  - $L_1 \cap L_2$
  - $L_1L_2$
  - $L_1^*$

- 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!