Finite Automata

Part Two
Recap from Last Time
DFAs

A *DFA* is a *Deterministic Finite Automaton*

DFAs are the simplest type of automaton that we will see in this course.
DFAs

A DFA is defined relative to some alphabet $\Sigma$.

For each state in the DFA, there must be exactly one transition defined for each symbol in $\Sigma$.

This is the “deterministic” part of DFA.

There is a unique start state.

There are zero or more accepting states.
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 **Nondeterministic** **Finite** **Automaton**

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.
Hello, NFA!

start \rightarrow q_0 \xrightarrow{h} q_1 \xrightarrow{i} q_2

h i
Hello, NFA!

start \xrightarrow{h} q_{0} \xrightarrow{i} q_{1} \xrightarrow{} q_{2}

h i
Hello, NFA!

start \rightarrow q_0 \xrightarrow{h} q_1 \xrightarrow{i} q_2

h \quad i
Hello, NFA!
Hello, NFA!
Hello, NFA!
Tragedy in Paradise

start $\rightarrow$ $q_0$ $\xrightarrow{h}$ $q_1$ $\xrightarrow{i}$ $q_2$

hipster

h i p
Tragedy in Paradise

start → $q_0$ → $q_1$ → $q_2$

h → i

h i p
Tragedy in Paradise

start \rightarrow q_0 \rightarrow h \rightarrow q_1 \rightarrow i \rightarrow q_2

\text{h i p}
Tragedy in Paradise
Tragedy in Paradise

\[
\begin{array}{c}
\text{start} \\ q_0 \quad h \quad q_1 \quad i \quad q_2
\end{array}
\]
Tragedy in Paradise

Start $q_0 \rightarrow h \rightarrow q_1 \rightarrow i \rightarrow q_2$

Input:

hi

Output:

The diagram shows a transition from $q_0$ to $q_1$ by reading the input 'h', and from $q_1$ to $q_2$ by reading the input 'i'. The starting state is marked as 'start' and the output sequence is 'hi'.
Tragedy in Paradise
Tragedy in Paradise
The **language of an NFA** is

\[ \mathcal{L}(N) = \{ w \in \Sigma^* \mid N \text{ accepts } w \} \].

What is the language of this NFA?

(Assume \( \Sigma = \{h, i\} \).)
The language of an NFA is

\[ \mathcal{L}(N) = \{ w \in \Sigma^* | N \text{ accepts } w \}. \]

\[ \Sigma = \{0, 1\} \]

\[ \begin{array}{c}
\text{start} \\
q_0 \\
\text{start} \\
q_0 \\
\text{start} \\
q_0
\end{array} \]

\[ \begin{array}{c}
q_1 \\
1 \\
q_0 \\
1 \\
q_2
\end{array} \]

\[ \begin{array}{c}
q_2 \\
0, 1
\end{array} \]
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.
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.
ε-Transitions

NFAs have a special type of transition called the \textit{ε-transition}.

An NFA may follow any number of ε-transitions at any time without consuming any input.
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.
**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.
ε-Transitions

NFAs have a special type of transition called the \textbf{ε-transition}.

An NFA may follow any number of ε-transitions at any time without consuming any input.
ε-Transitions

NFAs have a special type of transition called the \( \varepsilon \)-transition.

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

An NFA may follow any number of \( \varepsilon \)-transitions at any time without consuming any input.
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.
**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.
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.
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.
ε-Transitions

NFAs have a special type of transition called the \textit{ε-transition}.

An NFA may follow any number of ε-transitions at any time without consuming any input.
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.
**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.
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.

NFAs are not required to follow ε-transitions. It's simply another option at the machine's disposal.
Intuiting Nondeterminism

Nondeterministic machines are a serious departure from physical computers. How can we build up an intuition for them?

There are two particularly useful frameworks for interpreting nondeterminism:

*Perfect positive guessing*

*Massive parallelism*
Perfect Positive Guessing
Perfect Positive Guessing

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

\[ \Sigma \]

\[ \text{start} \]

\[ \text{a b a b a b a} \]
Perfect Positive Guessing

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

\[ \Sigma \]

Transition matrix:

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

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

\[ \Sigma \]

Start

a b a b a
Perfect Positive Guessing

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

\[ \Sigma \]

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a</th>
<th>b</th>
<th>a</th>
</tr>
</thead>
</table>

Diagram:
- Start state: \( q_0 \)
- Transitions: \( a \to q_1, b \to q_2, a \to q_3 \)
- \( q_3 \) is an accepting state.
Perfect Positive Guessing

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

Input: \( \Sigma \)
Perfect Positive Guessing

Start: $q_0$ -> $q_1$ (a) -> $q_2$ (b) -> $q_3$ (a)

Input: $\Sigma = \{a, b\}$

Sequence: a b a b a b a
Perfect Positive Guessing

Start $q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3$

Input sequence: $a \ b \ a \ b \ a \ b \ a$
Perfect Positive Guessing

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

\[ \Sigma \]

\[ a \ b \ a \ b \ a \ b \ a \]
Perfect Positive Guessing

\[
\begin{align*}
q_0 & \rightarrow a \rightarrow q_1 \\
q_1 & \rightarrow b \rightarrow q_2 \\
q_2 & \rightarrow a \rightarrow q_3
\end{align*}
\]

\[
\Sigma \rightarrow a b a b a b a
\]
Perfect Positive Guessing

\[ \sum \]

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

\[ a \rightarrow b \rightarrow a \]

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

SEAL

OF APPROVAL
Perfect Positive Guessing

We can view nondeterministic machines as having *Magic Superpowers* that enable them to guess choices that lead to an accepting state.

If there is at least one choice that leads to an accepting state, the machine will guess it.

If there are no choices, the machine guesses any one of the wrong guesses.

There is no known way to physically model this intuition of nondeterminism – this is quite a departure from reality!
Massive Parallelism

\[ a \ x b \ x a \ x b \ x a \ x a \]
Massive Parallelism

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

\[ \Sigma \]

Input sequence: a b a b a b a
Massive Parallelism

\[
\begin{align*}
q_0 & \xrightarrow{a} q_1 \\
q_1 & \xrightarrow{b} q_2 \\
q_2 & \xrightarrow{a} q_3
\end{align*}
\]
Massive Parallelism

\[
\begin{align*}
q_0 & \xrightarrow{a} q_1 \\
q_1 & \xrightarrow{b} q_2 \\
q_2 & \xrightarrow{a} q_3
\end{align*}
\]

\[\Sigma \]

Input sequence: \[a \, b \, a \, b \, a \, b \, a\]
Massive Parallelism

$q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3$

$a \ b \ a \ b \ a$
Massive Parallelism

\[ a \quad b \quad a \quad b \quad a \]
Massive Parallelism

\[ \Sigma \]

\[
\text{start} \quad q_0 \quad a \quad q_1 \quad b \quad q_2 \quad a \quad q_3
\]

\[
\begin{array}{ccccccc}
\text{a} & \text{b} & \text{a} & \text{b} & \text{a} & \text{a} & \text{a}
\end{array}
\]
Massive Parallelism

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

\[ a \ b \ a \ b \ a \ b \ a \]
Massive Parallelism

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

\[ \sum \rightarrow q_0 \text{ (start)} \rightarrow a \rightarrow q_1 \rightarrow b \rightarrow q_2 \rightarrow a \rightarrow q_3 \]

Input sequence: a b a b a b a
Massive Parallelism

\[ \Sigma \]

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

\[ a \ b \ a \ b \ a \ b \ a \]
Massive Parallelism

\[
\begin{align*}
&\text{start} \rightarrow q_0 \quad a \quad q_1 \\
&\quad \quad \quad \quad b \quad q_2 \\
&\quad \quad \quad \quad a \quad q_3
\end{align*}
\]

\[\Sigma\]

Input sequence: \[a \ b \ a \ b \ a \ a\]
Massive Parallelism

\[ \Sigma \]

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

\[ \begin{array}{ccccccc}
    a & b & a & b & a & b & a \\
\end{array} \]
Massive Parallelism

Σ

a

q₀

q₁

q₂

q₃

a b a b a

a b a b a
Massive Parallelism

$start \rightarrow \Sigma \rightarrow q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3$

$ababa$
Massive Parallelism

\begin{align*}
q_0 & \xrightarrow{a} q_1 \\
q_1 & \xrightarrow{b} q_2 \\
q_2 & \xrightarrow{a} q_3
\end{align*}

\begin{align*}
\Sigma & \xrightarrow{} q_0
\end{align*}

Input sequence: a b a b a a
Massive Parallelism

\[
\begin{align*}
\Sigma & \rightarrow a \rightarrow q_0 \rightarrow q_1 \rightarrow b \rightarrow q_2 \rightarrow a \rightarrow q_3 \\
\end{align*}
\]

Input: \( ababa \)
Massive Parallelism

\[
\begin{align*}
q_0 &\xrightarrow{\Sigma} q_1 \\
q_1 &\xrightarrow{a} q_2 \\
q_2 &\xrightarrow{a} q_3 \\
\end{align*}
\]
Massive Parallelism

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

Input sequence: a b a b a a
Massive Parallelism

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

\[ \Sigma \rightarrow q_0 \]

Input: a b a b a a
Massive Parallelism

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

\[ \Sigma \]

Input: a b a b b a a
Massive Parallelism

a b a b a

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

```
<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>a</th>
<th>b</th>
<th>a</th>
</tr>
</thead>
</table>
```

Diagram:
- Start state: $q_0$
- Transitions:
  - $q_0 \xrightarrow{a} q_1$
  - $q_1 \xrightarrow{b} q_2$
  - $q_2 \xrightarrow{a} q_3$
- Input alphabet: $\Sigma = \{a, b\}$
Massive Parallelism

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

\[ \Sigma \]

Input: \[ a \ b \ a \ b \ a \]
Massive Parallelism

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

\[ \text{start} \rightarrow q_0 \xrightarrow{\Sigma} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \]

a b a b a a
Massive Parallelism

\[
\begin{align*}
&\text{start} \quad q_0 \\
\quad \sum & \quad q_1 \quad b \\
\quad a & \quad q_2 \quad a \\
\quad \quad & q_3
\end{align*}
\]

\[
\begin{array}{cccc}
a & b & b & a \\
\end{array}
\]
Massive Parallelism

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

\[ \sum \]

\[ \text{start} \]

\[ a \ b \ a \ b \ a \ a \]
Massive Parallelism

\[ a \quad b \quad a \quad b \quad a \]
Massive Parallelism

\[
\sum \quad a \quad b \quad a \quad b \quad a
\]
Massive Parallelism

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

\[ \Sigma \]

\[ a b a b a a \]
Massive Parallelism

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

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

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a</th>
<th>b</th>
<th>a</th>
</tr>
</thead>
</table>
Massive Parallelism

We're in at least one accepting state, so there's some path that gets us to an accepting state.

a b a b a b a
Massive Parallelism

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

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

\[
\begin{array}{c}
a \\
b \\
aba
\end{array}
\]
Massive Parallelism

\[ \Sigma \]

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

\[ a \rightarrow b \rightarrow a \rightarrow b \]

\[ a \hspace{0.5cm} b \hspace{0.5cm} a \hspace{0.5cm} b \]
Massive Parallelism

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

\[ \Sigma \]

\[ a \ b \ a \ b \ a \ b \]
Massive Parallelism

\[ q_0, q_1, q_2, q_3 \]

\[
\begin{array}{c}
\text{start} \\
q_0 \xrightarrow{\Sigma} q_1 \xrightarrow{a} q_2 \xrightarrow{a} \overline{q_3} \\
\end{array}
\]
Massive Parallelism

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

\[ \Sigma \]

\[ \begin{array}{cccccc}
\text{a} & \text{b} & \text{a} & \text{b} & \text{a} & \text{b} \\
\end{array} \]
Massive Parallelism
Massive Parallelism

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

Input:

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

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

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

\[
\begin{array}{c}
a \\
b \\
\end{array}
\]
Massive Parallelism

The diagram shows a transition diagram with states and labels. The states are:

- $q_0$ (start)
- $q_1$
- $q_2$
- $q_3$ (loop)

The transitions are:

- From $q_0$ to $q_1$ on input $a$
- From $q_1$ to $q_2$ on input $b$
- From $q_2$ to $q_3$ on input $a$
- From $q_3$ to $q_0$ on input $\Sigma$

The sequence $a b a b b$ is shown at the bottom of the page.
Massive Parallelism
Massive Parallelism
Massive Parallelism

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

\[ \Sigma \]

\[
\begin{array}{cccc}
    & a & b & a & b \\
\end{array}
\]
Massive Parallelism

\[ \Sigma \]

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

\[
\begin{array}{cccc}
a & b & a & b \\
\end{array}
\]
Massive Parallelism

\[
\begin{align*}
\sum & \quad a \quad q_1 \quad b \quad q_2 \quad a \quad q_3 \\
\text{start} & \quad q_0
\end{align*}
\]
Massive Parallelism

\[ q_0 \stackrel{\Sigma}{\longrightarrow} a \quad q_1 \quad b \quad q_2 \quad a \quad q_3 \]

\[ a \quad b \quad a \quad b \quad a \quad b \]

\[ \uparrow \]
Massive Parallelism
Massive Parallelism

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

\[ \Sigma \]

a b a b b
Massive Parallelism

\[ q_0 \xrightarrow{\Sigma} q_1 \quad q_2 \quad q_3 \]

Input sequence: a b a b
Massive Parallelism

```
a b a b
```

Diagram:
- Start state: $q_0$
- Transitions: $a \rightarrow q_1$, $b \rightarrow q_2$, $a \rightarrow q_3$
- Input symbol: $\Sigma$
- Accepting state: $q_3$
Massive Parallelism

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

\[ \Sigma \]

\[ \begin{array}{cccc}
  a & b & a & b \\
\end{array} \]
Massive Parallelism

$$q_0 \xrightarrow{\Sigma} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3$$

Start

a b a b b
Massive Parallelism

\[ a, b, a, b \]

\[ \Sigma \]

\[ q_0 \to q_1 \to q_2 \to q_3 \]
Massive Parallelism
Massive Parallelism

\[ \Sigma \]

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

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

Start $q_0$ to $q_1$ with 'a', then $q_1$ to $q_2$ with 'b', then $q_2$ to $q_3$ with 'a'. The sequence 'a b a b' is shown in the diagram.
Massive Parallelism

We’re not in any accepting state, so no possible path accepts.

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

\[ \sum \]

\[ q_0 \quad q_1 \quad q_2 \quad q_3 \]

\[ a \quad b \quad a \quad b \quad a \quad b \]
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.

(Here's a rigorous explanation about how this works; read this on your own time).

Start off in the set of all states formed by taking the start state and including each state that can be reached by zero or more \( \varepsilon \)-transitions.

When you read a symbol \( a \) in a set of states \( S \):

Form the set \( S' \) of states that can be reached by following a single \( a \) transition from some state in \( S \).

Your new set of states is the set of states in \( S' \), plus the states reachable from \( S' \) by following zero or more \( \varepsilon \)-transitions.
So What?

Each intuition of nondeterminism is useful in a different setting:

Perfect guessing is a great way to think about how to design a machine.

Massive parallelism is a great way to test machines – and has nice theoretical implications.

Nondeterministic machines may not be feasible, but they give a great basis for interesting questions:

Can any problem that can be solved by a nondeterministic machine be solved by a deterministic machine?

Can any problem that can be solved by a nondeterministic machine be solved \textit{efficiently} by a deterministic machine?

The answers vary from automaton to automaton.
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

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

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

Nondeterministically guess when the end of the string is coming up.

Deterministically check whether you were correct.
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

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

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

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

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

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

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

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

Nondeterministically *guess* which character is missing.

Deterministically *check* whether that character is indeed missing.
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

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

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

\[ L = \{ w \in \{a, b, c\}^* | \text{at least one of } a, b, \text{ or } c \text{ is not in } w \} \]
Time out for Announcements
Midterm and Pset

• Congratulations on finishing the midterm!
• We’ll have this returned to you on Monday.
• Problem Set 4 is due next Thursday.
Just how powerful are NFAs?
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**!
Tabular DFAs

![Diagram of a DFA with states q₀, q₁, q₂, q₃ and transitions on 0 and 1]

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

\[
\begin{array}{|c|c|c|}
\hline
& 0 & 1 \\
\hline
*q_0 & q_1 & q_0 \\
q_1 & q_3 & q_2 \\
q_2 & q_3 & q_0 \\
*q_3 & q_3 & q_3 \\
\hline
\end{array}
\]
Tabular DFAs

These stars indicate accepting states.
Tabular DFAs

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

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?
A DFA with states $q_0$, $q_1$, $q_2$, and $q_3$. The transitions are labeled as follows:

- From $q_0$ to $q_1$ on input 'a'
- From $q_1$ to $q_2$ on input 'b'
- From $q_2$ to $q_3$ on input 'a'
- Transition from $q_0$ to $q_0$ on input $\Sigma$

The input string is 'a b a b a b a a'.
\[ \Sigma \]

start \( q_0 \) \( \xrightarrow{a} q_1 \) \( \xrightarrow{b} q_2 \) \( \xrightarrow{a} q_3 \)

\[
\begin{array}{cccccc}
  & a & b & a & b & a & a \\
\end{array}
\]
\[ \sum \]

```
*start*  \( q_0 \)  \( q_1 \)  \( q_2 \)  \( q_3 \)
```

```
```
The diagram represents a finite automaton with the following states:

- **Start state:** $q_0$
- **Transition labels:**
  - From $q_0$ to $q_1$, on input $a$
  - From $q_1$ to $q_2$, on input $b$
  - From $q_2$, on input $a$

The states are connected by arrows indicating the transitions. The automaton starts at $q_0$ and can move to $q_1$, then to $q_2$, and finally to an accepting state $q_3$. The transitions are labeled with the symbols $a$ and $b$. The diagram also shows an input tape with '?' symbols and an arrow pointing to the position where the automaton should process the input.
The diagram depicts a finite automaton with states labeled \( q_0, q_1, q_2, q_3 \). The transitions are as follows:

- From \( q_0 \) to \( q_1 \) on input \( a \)
- From \( q_1 \) to \( q_2 \) on input \( b \)
- From \( q_2 \) back to \( q_0 \) on input \( a \)
- \( q_3 \) is a final state.

The table below shows the transitions:

<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></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>
\begin{align*}
\Sigma & \xrightarrow{a} q_1 \\
q_1 & \xrightarrow{b} q_2 \\
q_2 & \xrightarrow{a} q_3
\end{align*}

\begin{array}{|c|c|c|}
\hline
 & a & b \\
\hline
\{q_0\} & \{q_0, q_1\} & \\
\hline
\hline
\hline
\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>$\sum$</td>
<td>$\rightarrow$</td>
<td>$\rightarrow$</td>
</tr>
<tr>
<td>start</td>
<td>$a$</td>
<td>$b$</td>
</tr>
</tbody>
</table>
\[ \Sigma \]

\[
\begin{array}{c c c}
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & & \\
 & & \\
 & & \\
\end{array}
\]
$$\Sigma \quad q_0 \quad a \quad q_1 \quad b \quad q_2 \quad a \quad q_3$$

<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></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
\begin{table}
\centering
\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\} & & \\
\hline
\end{tabular}
\end{table}
A finite automaton with the following transitions:

- Start state: $q_0$
- Transitions:
  - $a$: From $q_0$ to $q_1$
  - $b$: From $q_1$ to $q_2$
  - $a$: From $q_2$ to $q_3$ (-loop)

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></td>
<td></td>
</tr>
</tbody>
</table>
The diagram represents a finite automaton with the following states and transitions:

- **States:** 
  - $q_0$: Start state
  - $q_1$
  - $q_2$
  - $q_3$: Final state

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

The table below details the transition function for inputs $a$ and $b$:

<table>
<thead>
<tr>
<th>State Set</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>
<tr>
<td>${q_0, q_1}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
\[ q_3 \rightarrow q_3 \rightarrow q_2 \rightarrow q_1 \rightarrow q_0 \rightarrow \text{start} \]

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

The transition table for the automaton is as follows:

- From state \(q_0\) on input 'a' go to state \(q_1\).
- From state \(q_1\) on input 'b' go to state \(q_2\).
- From state \(q_2\) on input 'a' return to state \(q_3\).
<table>
<thead>
<tr>
<th>Table: Transition Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
</tr>
<tr>
<td>${q_0}$</td>
</tr>
<tr>
<td>${q_0, q_1}$</td>
</tr>
</tbody>
</table>
\[ q_3 \quad q_2 \quad q_1 \quad \Sigma \quad q_0 \]

<table>
<thead>
<tr>
<th>States</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></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_2 \} \\
\{ q_0, q_1 \} & \{ q_0, q_1 \} & \{ q_0, q_2 \} \\
\end{array}
Start: $q_0$ 

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></td>
<td></td>
</tr>
</tbody>
</table>
\[
\begin{array}{ccc}
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & \{q_0, q_1\} & \{q_0, q_2\} \\
\{q_0, q_2\} & & \\
\end{array}
\]
<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></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, 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 deterministic finite automaton (DFA) with states: \( q_0 \), \( q_1 \), \( q_2 \), and \( q_3 \). The transitions are labeled with inputs 'a' and 'b'. The automaton starts in state \( q_0 \) and accepts strings based on the sequence of inputs.
<table>
<thead>
<tr>
<th>States</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>

Diagram:

- **Start State**: \( q_0 \)
- **Final State**: \( q_3 \)
- **Transitions**:
  - \( q_0 \xrightarrow{a} q_1 \)
  - \( q_1 \xrightarrow{b} q_2 \)
  - \( q_2 \xrightarrow{a} q_3 \)
  - \( q_3 \) is a final state.
\[ \Sigma \]

<table>
<thead>
<tr>
<th>State Set</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_3 } )</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
\[ \begin{array}{c|c|c}
\emptyset & a & \emptyset \\
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & \{q_0, q_1\} & \{q_0, q_2\} \\
\{q_0, q_2\} & \{q_0, q_1, q_3\} & \emptyset \\
\end{array} \]
<table>
<thead>
<tr>
<th>States</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 finite automaton with states labeled with \{q_0\}, \{q_0, q_1\}, \{q_0, q_2\}, and \{q_0, q_1, q_3\}. The transitions are labeled with symbols a and b, starting from state \{q_0\} and ending in a loop at state \{q_3\}. The transitions are as follows:

- From \{q_0\} to \{q_1\} on input a.
- From \{q_1\} to \{q_2\} on input b.
- From \{q_2\} to \{q_3\} on input a.
\[
\begin{array}{|c|c|c|}
\hline
\text{States} & \text{a} & \text{b} \\
\hline
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & \{q_0, q_1\} & \{q_0, q_2\} \\
\{q_0, q_2\} & \{q_0, q_1, q_3\} & \{q_0\} \\
\{q_0, q_1, q_3\} & & \\
\hline
\end{array}
\]
\begin{array}{|c|c|c|}
\hline
 & \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\} & & \\
\hline
\end{array}
\[
\begin{array}{cccc}
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & \{q_0, q_1\} & \{q_0, q_2\} \\
\{q_0, q_2\} & \{q_0, q_1, q_3\} & \{q_0\} \\
\{q_0, q_1, q_3\} & & \\
\end{array}
\]
The given automaton is a nondeterministic finite automaton (NFA) with states labeled $q_0$, $q_1$, $q_2$, and $q_3$. The transitions are as follows:

- From $q_0$ on input $a$, the automaton can either stay in $q_0$ or transition to $q_1$.
- From $q_1$ on input $b$, the automaton can transition to $q_2$.
- From $q_2$ on input $a$, the automaton can transition to $q_3$.
- The initial state is $q_0$.

The table below shows the transitions 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>${q_0, q_1}$</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>
A DFA (Deterministic Finite Automaton)

- **States:** $q_0, q_1, q_2, q_3$
- **Start State:** $q_0$
- **Final State:** $q_3$
- **Transitions:**
  - $a$: $q_0 ightarrow q_1$
  - $b$: $q_1 ightarrow q_2$
  - $a$: $q_2 ightarrow q_3$

**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></td>
<td></td>
</tr>
</tbody>
</table>
### 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>
<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></td>
</tr>
</tbody>
</table>

**Diagram:**

- **States:** \(q_0, q_1, q_2, q_3\)
- **Start State:** \(q_0\)
- **Transitions:**
  - \(a\): \(q_0 \rightarrow q_1\)
  - \(b\): \(q_1 \rightarrow q_2\)
  - \(a\): \(q_2 \rightarrow q_3\)
- **Final States:** \(q_3\)
- Alphabet:** \(\Sigma\)
\[
\begin{array}{cc}
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & \{q_0, q_1\} & \{q_0, q_2\} \\
\{q_0, q_2\} & \{q_0, q_1, q_3\} & \{q_0\} \\
\{q_0, q_1, q_3\} & \{q_0, q_1\} & \{q_0\} \\
\end{array}
\]
<table>
<thead>
<tr>
<th>State Set</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>
\[ \begin{array}{|c|c|c|} \hline \text{States} & \text{a} & \text{b} \\ \hline \{ q_0 \} & \{ q_0, q_1 \} & \{ q_0 \} \\ \{ q_0, q_1 \} & \{ q_0, q_1 \} & \{ q_0, q_2 \} \\ \{ q_0, q_2 \} & \{ q_0, q_1, q_3 \} & \{ q_0 \} \\ \{ q_0, q_1, q_3 \} & \{ q_0, q_1 \} & \{ q_0 \} \\ \hline \end{array} \]
A DFA with transitions:
- \( q_0 \) to \( q_1 \) on input 'a'
- \( q_1 \) to \( q_2 \) on input 'b'
- \( q_2 \) to \( q_3 \) on input 'a'
- \( q_0 \) to \( q_0 \) on input 'b' (looping)

Transition table:

<table>
<thead>
<tr>
<th>State Set</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>
\[
\begin{array}{ccc}
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & \{q_0, q_1\} & \{q_0, q_2\} \\
\{q_0, q_2\} & \{q_0, q_1, q_3\} & \{q_0\} \\
*\{q_0, q_1, q_3\} & \{q_0, q_1\} & \{q_0, q_2\}
\end{array}
\]
The diagram represents a deterministic finite automaton (DFA) that accepts the string "ababaaba". The DFA starts at state $q_0$ and transitions through states $q_1$, $q_2$, and finally $q_3$ upon reading the input string. The transitions are labeled with characters from the alphabet $\Sigma$, which is not explicitly shown in the diagram but is typically the set of all input symbols.
Some Caveats

**Question**: what about $\varepsilon$-transitions?

Answer: always include any states you can reach by following $\varepsilon$-transitions.

**Question**: what happens if there are no transitions to follow from a set of states for the character you’re trying to fill in?

Answer: then the set of states you can reach is the empty set!

Example included in the appendix of this lecture showing this construction with both of these scenarios.
The Subset Construction

- This construction for transforming an NFA into a DFA is called the *subset construction* (or sometimes the *powerset construction*).
- Each state in the DFA is associated with a set of states in the NFA.
- The start state in the DFA corresponds to the start state of the NFA, plus all states reachable via $\epsilon$-transitions.
- If a state $q$ in the DFA corresponds to a set of states $S$ in the NFA, then the transition from state $q$ on a character $a$ is found as follows:
  - Let $S'$ be the set of states in the NFA that can be reached by following a transition labeled $a$ from any of the states in $S$. (*This set may be empty.*)
  - Let $S''$ be the set of states in the NFA reachable from some state in $S'$ by following zero or more epsilon transitions.
  - The state $q$ in the DFA transitions on $a$ to a DFA state corresponding to the set of states $S''$.
- *Read Sipser for a formal account.*
The Subset Construction

For the purposes of this class, we won’t ask you to actually perform the subset construction.

Hopefully though, you’ve been convinced that, in principle, you could follow this procedure to turn any NFA into a DFA.
The Subset Construction

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

*Useful fact:* $|\wp(S)| = 2^{|S|}$ 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 iff there is some NFA $N$ such that $\mathcal{L}(N) = L$. 
An Important Result

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

**Proof Sketch:**
An Important Result

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

**Proof Sketch:** If $L$ is regular, there exists some DFA for it, which we can easily convert into an NFA.
An Important Result

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

**Proof Sketch:** If $L$ is regular, there exists some DFA for it, which we can easily convert into an NFA.

If $L$ is accepted by some NFA, we can use the subset construction to convert it into a DFA that accepts the same language, so $L$ is regular.
An Important Result

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

**Proof Sketch:** If $L$ is regular, there exists some DFA for it, which we can easily convert into an NFA.

If $L$ is accepted by some NFA, we can use the subset construction to convert it into a DFA that accepts the same language, so $L$ is regular. ■
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 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 \textit{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

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

\[ \overline{L} = \{ w \in \{a, b\}^* \mid w \text{ does not contain } aa \text{ as a substring} \} \]
Complementing Regular Languages

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

\[ L = \{ w \in \{a, *, /\}^* \mid w \text{ doesn't represent a C-style comment} \} \]
**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.
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 $L_1 \cup L_2$?
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?
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?

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 $wx = \text{quokka}$.

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*:

$$wxy = w(xy) = (wx)y$$
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 $\Sigma = \{ a, b, \ldots, z, A, B, \ldots, Z \}$ and consider these languages over $\Sigma$:

- **Noun** = $\{ \text{Puppy, Rainbow, Whale, } \ldots \}$
- **Verb** = $\{ \text{Hugs, Juggles, Loves, } \ldots \}$
- **The** = $\{ \text{The} \}$

The language **TheNounVerbTheNoun** is

$\{ \text{ThePuppyHugsTheWhale, TheWhaleLovesTheRainbow, TheRainbowJugglesTheRainbow, } \ldots \}$
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 $L_2$. 
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:

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 $y \in L_2$?

**Idea:**

Machine for $L_1$  

Machine for $L_2$  

bookkeeper
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:

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 $y \in L_2$?

_Idea:_

![Machine for $L_1$](image1.png) ![Machine for $L_2$](image2.png)

**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 $L_2$
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 $L = \{ \text{aa, b} \}$

$L L$ is the set of strings formed by concatenating pairs of strings in $L$.

$$\{ \text{aaaa, aab, baa, bb} \}$$

$L L L$ is the set of strings formed by concatenating triples of strings in $L$.

$$\{ \text{aaaaaa, aaaab, aabaa, aabb, baaaa, baab, bbbaa, bbb} \}$$

$L L L L$ is the set of strings formed by concatenating quadruples of strings in $L$.

$$\{ \text{aaaaaaaa, aaaaaab, aaaaabaa, aaaaabb, aabaaaaa, aabbaab, aabbaaa, aabbb, baaaaaa, baaaaab, baabaa, baabb, bbbaaa, bbaab, bbbaa, bbbbb} \}$$
Language Exponentiation

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

$L^0 = \{ \varepsilon \}$

The set containing just the empty string.

Idea: Any string formed by concatenating zero strings together is the empty string.

$L^{n+1} = LL^n$

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 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:

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

Intuitively, 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^* = \{ \epsilon, \text{a, bb, aa, abb, bba, bbbb, aaa, aabb, abba, abbbbb, bbba, bbabb, bbbbb, 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: ⚠️

$L^0 = \{ \epsilon \}$ 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 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

\[0.999999 \approx 1\]
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

\[ \infty \text{ is finite} \]
Reasoning about Infinity

∞ 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).
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?

Machine for $L$

Machine for $L^*$
Closure Properties

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

$L_1$

$L_1 \cup L_2$

$L_1 \cap L_2$

$L_1L_2$

$L_1^*$

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!
Thought for the Weekend

Learning How to Learn