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.
If $D$ is a DFA, the **language of $D$**, denoted $\mathcal{L}(D)$, is \{ $w \in \Sigma^*$ | $D$ 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.
New Stuff!
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

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

\[ \Sigma \xrightarrow{\text{start}} q_0 \]

\[ \Sigma \]
Perfect Positive Guessing

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

Input alphabet: \( \Sigma \)

States transition:
- Start state: \( q_0 \)
- \( q_0 \) to \( q_1 \): on input 'a'
- \( q_1 \) to \( q_2 \): on input 'b'
- \( q_2 \) to \( q_3 \): on input 'a'
- \( q_3 \) is a loop on 'a'

Input sequence: \( a \ b \ a \ b \ a \ b \ a \ a \)
Perfect Positive Guessing
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

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

\[\Sigma\]

\[
\begin{array}{ccccccc}
a & b & a & b & a & b & a \\
\end{array}
\]
Perfect Positive Guessing

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

\[ \Sigma \]

\[ \text{start} \]

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

Input sequence: \[ a \ b \ a \ b \ a \ b \ a \]
Perfect Positive Guessing

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

\[ \Sigma \]

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

Transition:
- $q_0 \xrightarrow{a} q_1$
- $q_1 \xrightarrow{b} q_2$
- $q_2 \xrightarrow{a} q_3$

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

Sequence: $abaaba$
Perfect Positive Guessing

Start

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

$a \ b \ a \ b \ a \ b \ a$
Perfect Positive Guessing

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

State transition diagram:
- Start state: $q_0$
- States: $q_0$, $q_1$, $q_2$, $q_3$
- Transitions:
  - $q_0 \xrightarrow{a} q_1$
  - $q_1 \xrightarrow{b} q_2$
  - $q_2 \xrightarrow{a} q_3$

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

Strings: $a b a b a b a a$
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

### State Transition Diagram

- **Start State:** $q_0$
- **States:** $q_0, q_1, q_2, q_3$
- **Transitions:**
  - $q_0 \xrightarrow{a} q_1$
  - $q_1 \xrightarrow{b} q_2$
  - $q_2 \xrightarrow{a} q_3$

- **Input Alphabet:** $\Sigma = \{a, b\}$

### Sample Input String

- $a b a b a a$

The diagram illustrates a finite automaton with states $q_0, q_1, q_2,$ and $q_3$, where $q_0$ is the start state, and $q_3$ is the only accepting state. The input symbols $a$ and $b$ transition between states according to the labeled arrows.
Massive Parallelism

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

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

\[ \sum \]

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

\[ \text{start} \]

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

\[ \sum \]

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

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

The diagram shows a transition diagram with states $q_0, q_1, q_2,$ and $q_3$, and transitions labeled with symbols $a$ and $b$. The input alphabet is denoted as $\sum$. The transitions are as follows:

- From $q_0$ to $q_1$ with input $a$.
- From $q_1$ to $q_2$ with input $b$.
- From $q_2$ to $q_3$ with input $a$.
- From $q_3$ back to $q_0$ with a loop.

The sequence of symbols on the path is $a b a b a b a$. The diagram illustrates the concept of massive parallelism, where multiple states can be active simultaneously, representing parallel execution.
Massive Parallelism

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

Start

\[ \Sigma \]

Input:

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

Next state:

\[ q_3 \]
Massive Parallelism

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

\[ \Sigma \]

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

\[ \uparrow \]
Massive Parallelism

\[ \sum \]

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

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

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

- \( \Sigma \)
- start
- \( q_0 \)
- \( q_1 \)
- \( q_2 \)
- \( q_3 \)

Input sequence: \( abaaba \)
Massive Parallelism

\[ \Sigma \]

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

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

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

\[ \Sigma \]

Start

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

\[ \rightarrow \]
Massive Parallelism

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

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

\[ \sum \]

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

\[
\text{a b a b a a}
\]
Massive Parallelism

\[ \Sigma \]

- Start: \( q_0 \) → \( q_1 \) → \( q_2 \) → \( q_3 \)

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

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

\[ \Sigma \]

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

**Diagram Description:**
- Initial state: $q_0$
- Transition: $a \rightarrow q_1$
- Transition: $b \rightarrow q_2$
- Transition: $a \rightarrow q_3$
- Input symbols: $\Sigma$
- String input: $abaaba$
Massive Parallelism

\[ \Sigma \]

\[ q_0 \overset{a}{\rightarrow} q_1 \overset{b}{\rightarrow} q_2 \overset{a}{\rightarrow} q_3 \]

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

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

Input string: \text{a b a b a b a a}
Massive Parallelism

\[ \sum \]

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

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

\[ a \overset{\text{a}}{\longrightarrow} q_0 \overset{\text{a}}{\longrightarrow} q_1 \overset{\text{b}}{\longrightarrow} q_2 \overset{\text{a}}{\longrightarrow} q_3 \]

\( a \ b \ a \ b \ a \ b \ a \ a \)
Massive Parallelism

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

Start state: \( q_0 \)

Input alphabet: \( \Sigma \)

Sequence of inputs: \( ababaaba \)
Massive Parallelism

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

a b a b a b a
Massive Parallelism

\[ \Sigma \]

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

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

\[ a \quad b \quad a \quad b \quad a \quad b \quad a \]
Using the massive parallelism intuition, if we are in the states $q_0$ and $q_2$, what set of states will we be in after reading the character $a$?

Respond at pollev.com/zhenglian740
Massive Parallelism

\begin{align*}
\Sigma \\
\text{start} & \rightarrow q_0 \quad a \quad q_1 \quad b \quad q_2 \quad a \quad q_3 \\
\text{a b a b a b a a}
\end{align*}
Massive Parallelism

\[ \sum \]

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

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

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

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

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

\[ \sum \]

\[ \text{start} \]

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

\[ \Sigma \]

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

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

\[ \text{start} \]
Massive Parallelism

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

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

Input symbols: \( \Sigma \)

Input sequence: a b a b b
Massive Parallelism

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

\[ \Sigma \]

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

\[ \sum \]

- start \rightarrow q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3
  - a \rightarrow q_1
  - b \rightarrow q_2
  - a \rightarrow q_3

Input sequence: a b a b b
Massive Parallelism

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

Start

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

\[ q_3 \]
Massive Parallelism

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

\[ \sum \]

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

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

\[ \sum \]

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

\[ \text{start} \]

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

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

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

\[ \Sigma \]

Start

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

\[ \Sigma \]

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

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

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

Input: \[ \sum \]

Input strings:

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

\[ \Sigma \]

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

\[ a \ b \ a \ b \]

\[ \text{a} \ b \ a \ b \]
Massive Parallelism

The diagram shows a transition from state $q_0$ to $q_3$ with labels $a$, $b$, and $a$ on the transitions. The start state is $q_0$. The input alphabet is denoted by $\Sigma$. The sequence of labels read from left to right is $a b a b$.
Massive Parallelism
Massive Parallelism

\[ \Sigma \]

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

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

```
a b a b
```

Diagram:

- Start at $q_0$
- Transition on $a$ to $q_1$
- Transition on $b$ to $q_2$
- Transition on $a$ to $q_3$
- Transition on $\Sigma$ to $q_0$
Massive Parallelism

\[ q_3 \]

\[ q_2 \]

\[ q_1 \]

\[ q_0 \]

start

\( \Sigma \)

- a
- b
- a
- b
- a
- b
- b
Massive Parallelism
Massive Parallelism

\[ \Sigma \]

- Start: \( q_0 \) to \( q_1 \) on 'a'
- \( q_1 \) to \( q_2 \) on 'b'
- \( q_2 \) to \( q_3 \) on 'a'

Input sequence: \( a \ b \ a \ b \ a \ b \)
Massive Parallelism

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

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

Symbols:
- \( \Sigma \) for input alphabet
- 'a', 'b' for input symbols
Massive Parallelism

\[ \Sigma \]

Start \rightarrow q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3

\( a b a b a b \)
Massive Parallelism

\[ \Sigma \]

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

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

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

\[ \Sigma \]

Start

| a | b | a | b |
---|---|---|---|

|

\[ q_0 \]

\[ q_1 \]

\[ q_2 \]

\[ q_3 \]
Massive Parallelism

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

Input alphabet: \( \Sigma \)

States: \( q_0, q_1, q_2, q_3 \)

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

Final state: \( q_3 \)

Input sequence: \( a b a b b \)
Massive Parallelism

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

\[ \Sigma \]

\begin{align*}
  q_0 & \xrightarrow{a} q_1 & q_1 & \xrightarrow{b} q_2 & q_2 & \xrightarrow{a} q_3 \\
  \text{start} & & & & & \text{We're not in any accepting state, so no possible path accepts.}
\end{align*}
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.
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

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

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

Which of these states should we mark as accepting states?

Respond at pollev.com/zhenglian740
Guess-and-Check

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

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

Deterministically **check** whether you were correct.
Guess-and-Check

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

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

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

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

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

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

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

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

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

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

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

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

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

$L = \{ \ w \in \{a, b, c\}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w \ \}$
Time-Out For Announcements!
Midterm Exam on Friday!

- Our midterm exam will be on Friday, July 26th from 5:00 – 8:00 PM in Hewlett 201 (our normal lecture room).
- You’re responsible for lectures up to the end of week 3 and topics from PS1 – PS3. Later lectures and problem sets won’t be tested here. Exam problems may build on the written or coding components from the problem sets.
- The exam is open-book, open-note, and closed-other-humans/AI.
Back to CS103!
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!
Thought Experiment:
How would you simulate a finite automata in software?
Tabular DFAs

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

These stars indicate accepting states.
Tabular DFAs

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

The diagram shows a finite automaton with states $q_0$, $q_1$, $q_2$, and $q_3$. The transitions are labeled with symbols 0 and 1, indicating the input alphabet. The start state is $q_0$. The table below represents the transitions:

<table>
<thead>
<tr>
<th>State</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

Question to ponder:
Why isn’t there a column here for Σ?
Code? In a Theory Class?

```cpp
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];
}
```
Can we do something similar for NFAs?
The diagram represents a finite automaton with transitions:

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

The automaton accepts the string $abaabaa$.
The diagram shows a finite state automaton (FSA) with states $q_0$, $q_1$, $q_2$, and $q_3$. The transitions are labeled with input symbols $a$ and $b$. The automaton starts at state $q_0$ and can move to $q_1$ on input $a$, then to $q_2$ on input $b$, and finally to $q_3$ on input $a$. The input alphabet is denoted by $\Sigma$. The sequence $abaaba$ is shown at the bottom, indicating a possible input to the automaton.
\[ q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \]

Input alphabet: \( \Sigma \)

A finite automaton with states $q_0, q_1, q_2, q_3$. The transitions are labeled with symbols $a$ and $b$. The input alphabet is $\Sigma$. The automaton starts at state $q_0$. The diagram shows transitions from $q_0$ to $q_1$ on input $a$, from $q_1$ to $q_2$ on input $b$, and a loop from $q_2$ back to $q_1$ on input $a$. The state $q_3$ is a dead state.
\[
\begin{array}{c|c|c}
 & a & b \\
\hline
\{q_0\} & \{q_0, q_1\} & \\
\{q_0, q_1\} & & \\
\{q_1, q_2\} & & \\
\{q_2, q_3\} & & \\
\{q_3\} & & \\
\end{array}
\]
\begin{align*}
\Sigma & \rightarrow q_0 \\
& \rightarrow q_1 \quad a \rightarrow b \\
& \rightarrow q_2 \quad b \rightarrow a \\
& \rightarrow q_3
\end{align*}

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

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>
</tbody>
</table>
\[
\begin{array}{c|c|c}
      & a & b \\
\hline
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\end{array}
\]
\[ \begin{array}{c|cc}
\text{ } & \text{a} & \text{b} \\
\hline
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & & \\
\end{array} \]
\[
\begin{array}{c}
\Sigma \\
\end{array}
\]

Diagram:
- Start state: \( q_0 \)
- Transitions:
  - \( q_0 \) on \( a \) to \( q_1 \)
  - \( q_1 \) on \( b \) to \( q_2 \)
  - \( q_2 \) on \( a \) 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></td>
<td></td>
</tr>
<tr>
<td></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></td>
<td></td>
</tr>
</tbody>
</table>
\[
\begin{array}{c}
\Sigma \\
\end{array}
\]

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

- **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>
<tr>
<td>({q_0, q_1})</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
\[ \Sigma \]

**Diagram:**
- **Start State:** \( q_0 \)
- **States:** \( q_0, q_1, q_2, q_3 \)
- **Transitions:**
  - \( a \) from \( q_0 \) to \( q_1 \)
  - \( b \) from \( q_0 \) to \( q_2 \)
  - \( a \) from \( q_1 \) to \( q_2 \)
  - \( a \) from \( q_2 \) to \( q_3 \)

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

\[
\begin{array}{|c|c|c|}
\hline
 & a & b \\
\hline \{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & & \\
\hline
\end{array}
\]
The given image is a finite automaton (FA) with states labeled as $q_0$, $q_1$, $q_2$, and $q_3$. The start state is $q_0$. The FA transitions are as follows:

- From $q_0$, on input $a$, it transitions to $q_1$.
- From $q_1$, on input $b$, it transitions to $q_2$.
- From $q_2$, on input $a$, it loops back to $q_2$.
- From $q_2$, on input $a$, it transitions to $q_3$.

The accepting states are $q_2$ and $q_3$.

The input alphabet is denoted by $\Sigma$, and the transitions are determined by the following 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>
</tbody>
</table>
\[ \begin{array}{c|cc}
\{ q_0 \} & \{ q_0, q_1 \} & \{ q_0 \} \\
\{ q_0, q_1 \} & \{ q_0, q_1 \} & \\
\{ q_0 \} & & \\
\end{array} \]
The given diagram represents a finite automaton with states $q_0$, $q_1$, $q_2$, and $q_3$. 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 $a$.
- There is an implicit transition from $q_3$ back to $q_0$ on any input $\Sigma$.

The automaton starts in state $q_0$. 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>${q_0}$</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>
<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_1}</td>
<td>{q_0, q_1}</td>
<td>{q_0, q_2}</td>
</tr>
<tr>
<td>q_3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Diagram:**

- **States:** q_0, q_1, q_2, q_3
- **Start state:** q_0
- **Final state:** q_3
- **Transitions:**
  - a: q_0 \rightarrow q_1
  - b: q_1 \rightarrow q_2
  - a: q_2 \rightarrow q_3 (loop)

**Input alphabet:** \(\Sigma\)
\[ \sum \]

**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>
For the given automaton, the transition table is as follows:

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

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

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 \( q_3 \)
- \( q_3 \) is a final state

Transition table:

<table>
<thead>
<tr>
<th>State</th>
<th>Input</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>( { q_0 } )</td>
<td>( a )</td>
<td>( b )</td>
</tr>
<tr>
<td>( { q_0, q_1 } )</td>
<td>( { q_0, q_1 } )</td>
<td>( { q_0 } )</td>
</tr>
<tr>
<td>( { q_0, q_1 } )</td>
<td>( { q_0, q_1 } )</td>
<td>( { q_0, q_2 } )</td>
</tr>
<tr>
<td>( { q_0, q_2 } )</td>
<td>( { q_0, q_1, q_3 } )</td>
<td>-</td>
</tr>
</tbody>
</table>
The diagram shows a finite automaton with states $q_0$, $q_1$, $q_2$, and $q_3$. The automaton transitions as follows:

- The start state is $q_0$.
- On input $a$, the automaton transitions from $q_0$ to $q_1$.
- On input $b$, the automaton transitions from $q_1$ to $q_2$.
- On input $a$, the automaton transitions from $q_2$ to $q_3$.
- On input $a$, the automaton transitions from $q_3$ back to $q_0$.

The table below describes the transition function of the automaton:

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

The alphabet of the automaton is $\Sigma$. The automaton accepts strings over the alphabet $\{a, b\}$. The state $q_3$ is a final state, indicating acceptance of strings.
\[
\begin{align*}
\Sigma & \quad q_0 \quad a \quad q_1 \quad b \quad q_2 \quad a \quad q_3 \\
\{q_0\} & \quad \{q_0, q_1\} \quad \{q_0\} \\
\{q_0, q_1\} & \quad \{q_0, q_1\} \quad \{q_0, q_2\} \\
\{q_0, q_2\} & \quad \{q_0, q_1, q_3\} \\
\end{align*}
\]
\[
\begin{array}{c}
\Sigma \\
\downarrow \\
\text{start} \\
\longrightarrow \\
q_0 \\
\longrightarrow \ a \longrightarrow \ q_1 \\
\longrightarrow \ b \longrightarrow \ q_2 \\
\longrightarrow \ a \longrightarrow \ q_3 \\
\end{array}
\]

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

\[ \Sigma \]

<table>
<thead>
<tr>
<th>State</th>
<th>(a)</th>
<th>(b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>({q_0})</td>
<td>({q_0, q_1})</td>
<td>({q_0})</td>
</tr>
<tr>
<td>({q_0, q_1})</td>
<td>({q_0, q_1})</td>
<td>({q_0, q_2})</td>
</tr>
<tr>
<td>({q_0, q_2})</td>
<td>({q_0, q_1, q_3})</td>
<td>({q_0})</td>
</tr>
<tr>
<td>({q_0, q_1, q_3})</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
\[
\begin{align*}
\Sigma & \quad a \quad b \\
\{ q_0 \} & \{ q_0, q_1 \} & \{ q_0 \} \\
\{ q_0, q_1 \} & \{ q_0, q_1 \} & \{ q_0, q_2 \} \\
\{ q_0, q_2 \} & \{ q_0, q_1, q_3 \} & \{ q_0 \} \\
\{ q_0, q_1, q_3 \} & & \text{---}
\end{align*}
\]
\[
\begin{array}{ccc}
\Sigma & a & b \\
\{q_0\} & \{q_0, q_1\} & \{q_0\} \\
\{q_0, q_1\} & \{q_0, q_1\} & \{q_0, q_2\} \\
\{q_0, q_2\} & \{q_0, q_1, q_3\} & \{q_0\} \\
\{q_0, q_1, q_3\} & & \\
\end{array}
\]
\[a \rightarrow \{q_0, q_1\}\]
\[b \rightarrow \{q_0, q_2\}\]

The transitions and states are as follows:

- \(q_0\) starts and transitions on \(a\) to \(q_1\).
- \(q_1\) transitions on both \(a\) and \(b\) to \(q_2\) and \(q_3\) respectively.
- \(q_3\) is an accepting state.

The table details 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>({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>
\[
\begin{align*}
\Sigma &
\begin{array}{ccc}
\text{start} & a & b \\
q_0 & a & b \\
q_1 & b & \\
q_2 & a & b \\
q_3 & \\
\end{array}
\end{align*}
\]

<table>
<thead>
<tr>
<th>State</th>
<th>States for a</th>
<th>States for 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>
\[ q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3 \]

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

<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>{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>

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

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

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>{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>

The transitions are:

- Start state: \{q_0\}
- From \{q_0\} on a: \{q_0, q_1\}
- From \{q_0, q_1\} on b: \{q_0, q_2\}
- From \{q_0, q_2\} on a: \{q_0, q_1, q_3\}
- From \{*q_0, q_1, q_3*\} on b: \{q_0, q_1, q_3\}
- From \{*q_0, q_1, q_3*\} on a: \{q_0, q_1, q_3\}

The diagram above illustrates the states and transitions.
The diagram represents a deterministic finite automaton (DFA) with the following transitions:

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

The input string is 'a b a a b a b a', and the corresponding path in the automaton is:

1. Start at $q_0$
2. Move to $q_1$ on 'a'
3. Move back to $q_0$ on 'b'
4. Move to $q_2$ on 'b'
5. Move to $q_3$ on 'a'

The automaton concludes in state $q_3$.
Start:

\[ \Sigma \]

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

Input String:
- \( a b a a b b a \)

States:
- \{q_0\}
- \{q_0, q_1\}
- \{q_0, q_2\}
- \{q_0, q_1, q_3\}
\[
q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_3
\]

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

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

\[
\{q_0, q_2\}
\]

\[
\{q_0\}
\]
The image shows a finite automaton with transitions labeled as follows:

- Start state: $q_0$
- 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 $a$

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

The states are also labeled with sets of states:

- $q_0$
- $q_1$
- $q_2$
- $q_3$

The transitions are represented as follows:

- From $q_0$ to $q_1$ on $a$
- From $q_1$ to $q_2$ on $b$
- From $q_2$ to $q_3$ on $a$

The automaton is deterministic as there is a single transition for each letter in the input alphabet from each state.

The text below the automaton shows a sequence of inputs: $ababaaba$.
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.
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 if and only if there is some NFA $N$ such that $ℒ(N) = L$.

**Proof Sketch:** Pick a language $L$. First, assume $L$ is regular. That means there’s a DFA $D$ where $ℒ(D) = L$. 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 $ℒ(N) = L$. Using the subset construction, we can build a DFA $D$ where $ℒ(N) = ℒ(D)$. Then we have that $ℒ(D) = L$, 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 Complement of a Language

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

• Formally:

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

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

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

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

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

• Formally:

$$\overline{L} = \Sigma^* - L$$
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 $\overline{\overline{L}} = L$ for any language $L$. 
Complementing Regular Languages

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

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

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

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

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

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

**Question to ponder:** are the nonregular languages closed under complementation?
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 $L_1 \cup L_2$ 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 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?
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**:
    \[
    wxy = w(xy) = (wx)y
    \]
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 $\Sigma$:

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

- 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 $y \in L_2$?
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 $L_1$  

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 $L_2$
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 $y \in L_2$?

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 $L_2$
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 $L$.
  \[
  \{ \text{aaaaaa, aaaaab, aabaa, aabb, baaaaa, baab, bbbaa, bbb} \}
  \]
- $LLLL$ is the set of strings formed by concatenating quadruples of strings in $L$.
  \[
  \{ \text{aaaaaaaaa, aaaaaab, aaaaabaa, aaaaabb, aabaaaaa, aabaab, aabbaa, aabb, baaaaaa, baaaab, baabaa, baabb, bbaaaa, bbaab, bbbaa, bbb} \}
  \]
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 \( L^* = \{ \)

\[ \varepsilon, \]
\[ \text{a, bb,} \]
\[ \text{aa, abb, bba, bbbb,} \]
\[ \text{aaa, aabb, abba, abbbb, bbba, bbabb, bbbba, bbbbbbb,} \]
\[ \ldots \]
\}

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