Finite Automata
Part Two
Recap from Last Time
Formal Language Theory

- An **alphabet** is a set, usually denoted $\Sigma$, consisting of elements called **characters**.
- A **string over $\Sigma$** is a finite sequence of zero or more characters taken from $\Sigma$.
- The **empty string** has no characters and is denoted $\varepsilon$.
- A **language over $\Sigma$** is a set of strings over $\Sigma$.
- The language $\Sigma^*$ is the set of all strings over $\Sigma$. 
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.
The Language of an Automaton

• If $D$ is a DFA that processes strings over $\Sigma$, the *language of $D$*, denoted $\mathcal{L}(D)$, is the set of all strings $D$ accepts.

• Formally:

$$\mathcal{L}(D) = \{ w \in \Sigma^* \mid D \text{ accepts } w \}$$
New Stuff!
Recognizing Languages with DFAs

$L = \{ w \in \{a, b\}^* | w \text{ contains } aa \text{ as a substring } \}$
Recognizing Languages with DFAs

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

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

$L = \{ w \in \{a, b\}^* \mid w \text{ contains } aa \text{ as a substring } \}$
Recognizing Languages with DFAs

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

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

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

$L = \{ w \in \{a, b\}^* | w \text{ contains } aa \text{ as a substring } \}$
Recognizing Languages with DFAs

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

$L = \{ \ w \in \{a, b\}^* \mid w \text{ contains } aa \text{ as a substring } \}$
Recognizing Languages with DFAs

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

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

Let’s have the a symbol be a placeholder for “some character that isn’t a star or slash.”

Let’s design a DFA for C-style comments. Those are the ones that start with /* and end with */.

Accepted:

`/**a*/`
`/***/`
`/***/`
`/*****/`
`/**aaa*/aaa*/`
`/*a/a*/`

Rejected:

`/***/a/*/aa*/`
`aaa/%%%/aa`
`/**/`
`//*/*/*a/`
`//aaaa`
More Elaborate DFAs

$L = \{ w \in \{a, *, /\}^* | w \text{ represents a C-style comment } \}$
Tabular DFAs

![Diagram of a deterministic finite automaton (DFA)]

The DFA consists of states $q_0, q_1, q_2, q_3$ with transitions labeled by 0 and 1. The start state is $q_0$, and the accepting states are $q_3$. The transitions are as follows:

- From $q_0$, on input 1, go to $q_0$.
- From $q_0$, on input 0, go to $q_1$.
- From $q_1$, on input 1, go to $q_2$.
- From $q_2$, on input 0, go to $q_3$.
- From $q_3$, on any input, stay in $q_3$.
Tabular DFAs

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

<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>
These stars indicate accepting states.
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];
}
The Regular Languages
A language $L$ is called a **regular language** if there exists a DFA $D$ such that $\mathcal{L}(D) = L$.

If $L$ is a language and $\mathcal{L}(D) = L$, we say that $D$ **recognizes** the language $L$. 
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:

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

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

\[ \bar{L} = \{ w \in \{a, b\}^* \mid w \text{ does not contain } aa \text{ as a substring } \} \]
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?
NFAs
Revisiting a Problem
NFAs

- An **NFA** is a
  - **N**ondeterministic
  - **F**inite
  - **A**utomaton
- Structurally similar to a DFA, but represents a fundamental shift in how we'll think about computation.
(Non)determinism

• A model of computation is *deterministic* if at every point in the computation, there is exactly one choice that can make.
  • The machine accepts if that series of choices leads to an accepting state.

• A model of computation is *nondeterministic* if the computing machine has a finite number of choices available to make at each point, possibly including zero.
  • The machine accepts if *any* series of choices leads to an accepting state.
    • (This sort of nondeterminism is technically called *existential nondeterminism*, the most philosophical-sounding term we’ll introduce all quarter.)
A Simple NFA
A Simple NFA

$q_0$ has two transitions defined on 1!
A Simple NFA

start → $q_0$ → $q_1$ → $q_2$ → $q_3$

$0, 1$ $1$ $0$ $0, 1$ $0$ $0, 1$ $0, 1$ $0$ $1$ $0$ $1$ $1$
A Simple NFA

0 1 0 1 1
A Simple NFA
A Simple NFA
A Simple NFA

\[
\begin{align*}
q_0 & \xrightarrow{0, 1} q_1 \\
q_1 & \xrightarrow{1} q_2 \\
q_3 & \xrightarrow{0, 1} q_3
\end{align*}
\]
A Simple NFA
A Simple NFA
A Simple NFA

0 1 0 1 1
A Simple NFA

\begin{figure}[h]
\centering
\begin{tikzpicture}[node distance = 2cm, thick, main/.style = {draw, circle}]
\node[main] (q0) {$q_0$};
\node[main] (q1) [right of=q0] {$q_1$};
\node[main] (q2) [right of=q1] {$q_2$};
\node[main] (q3) [below right of=q2] {$q_3$};

\path[->]
(q0) edge node {$1$} (q1)
(q1) edge node {$1$} (q2)
(q2) edge [loop above] node {$0,1$} (q2)
(q1) edge [loop below] node {$0,1$} (q1)
(q2) edge node {$0$} (q3)
(q3) edge [loop below] node {$0,1$} (q3)
(q3) edge node {$0,1$} (q2);
\end{tikzpicture}
\end{figure}
A Simple NFA
A Simple NFA

\[
\begin{align*}
q_0 & \xrightarrow{0, 1} q_3 & & q_1 & \xrightarrow{1} q_2 & & q_3 & \xrightarrow{0, 1} q_3 & & q_3 & \xrightarrow{0, 1} q_3
\end{align*}
\]
A Simple NFA
A Simple NFA
A Simple NFA

1. From $q_0$, on input 1, go to $q_1$.
2. From $q_1$, on input 1, go to $q_2$.
3. From $q_2$, on input 0, go back to $q_3$.
4. From $q_3$, on input 0, 1, go to $q_2$.

Input: 0 1 0 1 1
A Simple NFA
A Simple NFA

start
q₀

1
q₁

1
q₂

0, 1
q₃

0, 1

0, 1

0 1 0 1 1
A Simple NFA

\[ q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_2 \]

Start state: \( q_0 \)

Final states: \( q_3 \)

Input symbols: 0, 1

Input string: 0 1 0 1 1
A Simple NFA

The NFA has the following states and transitions:

- Start state: $q_0$
- States: $q_0, q_1, q_2, q_3$
- Transitions:
  - $q_0 \xrightarrow{1} q_1$
  - $q_1 \xrightarrow{1} q_2$
  - $q_2 \xrightarrow{0, 1} q_3$
  - $q_3 \xrightarrow{0, 1} q_2$

Input sequence: 0 1 0 1 1
A Simple NFA
A Simple NFA

Diagram:
- Start state: $q_0$
- Transitions:
  - $q_0 \xrightarrow{1} q_1$
  - $q_0 \xrightarrow{0,1} q_3$
  - $q_1 \xrightarrow{1} q_2$
  - $q_1 \xrightarrow{0} q_3$
  - $q_3 \xrightarrow{0,1} q_2$

Input: 0 1 0 1 1
A Simple NFA
A Simple NFA

```
| 0 | 1 | 0 | 1 | 1 |
```

The NFA transitions are:
- From $q_0$ to $q_1$: on $1$
- From $q_1$ to $q_2$: on $1$
- From $q_2$: on $0, 1$
- From $q_3$: on $0, 1$
A Simple NFA

\[ \begin{align*}
q_0 &\xrightarrow{1} q_1 \\
q_1 &\xrightarrow{1} q_2 \\
q_0 &\xrightarrow{0,1} q_3 \\
q_3 &\xrightarrow{0,1} q_0 \\
q_3 &\xrightarrow{0,1} q_2 \\
\end{align*} \]

Input: 0 1 0 1 1 1
A Simple NFA

Start: $q_0$
- Transition on '0' to $q_0$
- Transition on '1' to $q_1$

$q_1$
- Transition on '1' to $q_1$

$q_2$
- Transition on '0' to $q_2$
- Transition on '1' to $q_3$

$q_3$
- Transition on '0' to $q_3$
- Transition on '1' to $q_2$

Input: 01011
A Simple NFA

start

$q_0$\rightarrow 1 \rightarrow q_1 \rightarrow 1 \rightarrow q_2

$q_3$\rightarrow 0 \rightarrow 0 \rightarrow 0 \rightarrow 0, 1

0 1 0 1 1
A Simple NFA

\[
\begin{align*}
q_0 & \xrightarrow{1} q_1 \\
q_1 & \xrightarrow{1} q_2 \\
q_0 & \xrightarrow{0, 1} q_3 \\
q_3 & \xrightarrow{0, 1} q_2 \\
\end{align*}
\]

Input: 010111
A Simple NFA

Start:

\[ q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_2 \]

Transitions:

- \[ q_0 \xrightarrow{0,1} q_3 \]
- \[ q_3 \xrightarrow{0,1} q_2 \]
- \[ q_2 \xrightarrow{0,1} q_3 \]

Input: 010111
A More Complex NFA

A More Complex NFA

\[ \text{start} \rightarrow q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_2 \]

\[ \begin{align*}
q_0 & \xrightarrow{0, 1} q_0 \\
q_1 & \xrightarrow{1} q_1
\end{align*} \]
A More Complex NFA

If a NFA needs to make a transition when no transition exists, the automaton **dies** and that particular path does not accept.
A More Complex NFA
A More Complex NFA

\[ q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_2 \]

\[ \text{start} \]

\[ 0, 1 \]

0 1 0 1 1 1
A More Complex NFA

\[ q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_2 \]
A More Complex NFA

\[
\begin{array}{c}
\text{start} \\
q_0 \quad 1 \quad q_1 \\
\circlearrowleft \quad 0, 1 \quad \qquad 1 \\
\quad q_2
\end{array}
\]
A More Complex NFA

\[
\begin{array}{c}
\text{start} \\
q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_2
\end{array}
\]
A More Complex NFA
A More Complex NFA

Oh no! There's no transition defined!
A More Complex NFA
A More Complex NFA

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

\[
\begin{array}{c}
0, 1
\end{array}
\]
A More Complex NFA
A More Complex NFA

\begin{center}
\begin{tikzpicture}

\node[state, initial] (q0) at (0,0) {$q_0$};
\node[state] (q1) at (2,0) {$q_1$};
\node[state, accepting] (q2) at (4,0) {$q_2$};
\draw (q0) edge[above] node {$1$} (q1);
\draw (q1) edge[above] node {$1$} (q2);
\draw (q0) edge[loop below] node [below] {$0, 1$} (q0);
\end{tikzpicture}
\end{center}

\begin{center}
\begin{tabular}{cccccc}
0 & 1 & 0 & 1 & 1 \\
\end{tabular}
\end{center}
A More Complex NFA

start

$q_0$ → $1$ → $q_1$ → $1$ → $q_2$

$0, 1$

0 1 0 1 1

↑
A More Complex NFA

0 1 0 1 1
A More Complex NFA

Start: $q_0$, 1

$q_0$ to $q_1$: 1

$q_1$ to $q_2$: 1

$q_2$: $\epsilon$, 0, 1

Input: 0 1 0 1 1
A More Complex NFA

The diagram shows a non-deterministic finite automaton (NFA) with states $q_0$, $q_1$, and $q_2$. The transitions are as follows:

- From $q_0$, on input 1, move to $q_1$.
- From $q_1$, on input 1, move to $q_2$.
- From $q_2$, on input 0 or 1, loop back to $q_2$.

The input sequence is 0 1 0 1 1.
A More Complex NFA
A More Complex NFA
A More Complex NFA

\[ q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_2 \]

Start

SEAL

OF APPROVAL

0 1 1
Hello, NFA!

![NFA Diagram]

- Start state: $q_0$
- Transition on $h$: $q_0 \rightarrow q_1$
- Transition on $i$: $q_1 \rightarrow q_2$
- Accepting state: $q_2$

Input alphabet: $\{h, i\}$
Hello, NFA!

The diagram shows a non-deterministic finite automaton (NFA) with states $q_0$, $q_1$, and $q_2$. The transitions are labeled with symbols $h$ and $i$. The start state is $q_0$.
Hello, NFA!
Hello, NFA!
Hello, NFA!

\[
\begin{align*}
q_0 &\xrightarrow{h} q_1 \\
q_1 &\xrightarrow{i} q_2
\end{align*}
\]
Hello, NFA!

\[ q_0 \xrightarrow{h} q_1 \xrightarrow{i} q_2 \]

Start state: \(q_0\)

States: \(q_0, q_1, q_2\)

Transitions:
- \(q_0 \xrightarrow{h} q_1\)
- \(q_1 \xrightarrow{i} q_2\)

Symbols: \(h, i\)

Picture: Seal with the text "SEAL OF APPROVAL"
Tragedy in Paradise

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

\begin{array}{ccc}
  h & i & p
\end{array}
Tragedy in Paradise
Tragedy in Paradise

\[
\text{start} \quad \xrightarrow{h} \quad q_0 \quad \xrightarrow{i} \quad q_1 \quad \xrightarrow{} \quad q_2
\]
Tragedy in Paradise

\begin{center}
\begin{tikzpicture}
    \node[state, initial] (q0) {$q_0$};
    \node[state, accepting, right of=q0] (q1) {$q_1$};
    \node[state, accepting, right of=q1] (q2) {$q_2$};
    \draw[->] (q0) edge node {$h$} (q1);
    \draw[->] (q1) edge node {$i$} (q2);
\end{tikzpicture}
\end{center}

\begin{center}
\begin{tabular}{c}
\texttt{h} \texttt{i} \texttt{p}
\end{tabular}
\end{center}
Tragedy in Paradise
Tragedy in Paradise
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 each NFA? (Assume $\Sigma = \{a, b\}$.)

**Question to ponder:** Why is the answer $\{ w \in \Sigma^* \mid w \text{ ends in } aaa \}$ not correct?

Note that flipping the accept and reject states of an NFA doesn’t always give an NFA for the complement of the original language. (Why?)
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.
\(\varepsilon\)-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.
**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 \textbf{ε-transition}.

• An NFA may follow any number of ε-transitions at any time without consuming any input.
\section*{\varepsilon-Transitions}

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

\begin{itemize}
\item \begin{tikzpicture}
  \node[state,initial] (q0) at (0,0) {$q_0$};
  \node[state] (q1) at (1,0) {$q_1$};
  \node[state] (q2) at (2,0) {$q_2$};
  \node[state,accepting] (q3) at (0,-1) {$q_3$};
  \node[state] (q4) at (1,-1) {$q_4$};
  \node[state] (q5) at (2,-1) {$q_5$};

  \draw[->] (q0) edge node {$a$} (q1);
  \draw[->] (q1) edge node {$a$} (q2);
  \draw[->] (q3) edge node {$b$} (q4);
  \draw[->] (q4) edge node {$b$} (q5);
  \draw[->, bend left] (q0) edge node {$\varepsilon$} (q3);
  \draw[->, bend left] (q3) edge node {$\varepsilon$} (q4);
  \draw[->] (q4) edge node {$\varepsilon$} (q5);
  \draw[->] (q0) edge[bend left] node {$\varepsilon$} (q4);
  \draw[->] (q3) edge[bend left] node {$b$} (q5);
\end{tikzpicture}
\end{itemize}
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 **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.

![NFA Diagram]

Not at all fun or rewarding exercise: what is the language of this NFA?
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.
Time-Out For Announcements!
Stanford CURIS

Funded CS Research Opportunities Workshop

Learn about two CURIS programs designed to provide funding for undergraduate students who are new to research! In this workshop, we will provide details on both programs, share application tips, and answer your questions.

Applications for both programs will be due at the end of Fall Quarter. Slides and a recording of the workshop will be shared afterwards.

---

**CURIS Fellowships**

Apply to the CURIS Fellowship program for guaranteed CURIS summer research funding (in advance of the standard CURIS matching process)!

The goal of this program is to support students who do not have prior CS research experience and to make research more accessible to a diverse group of students.

---

**Paid Undergraduate Research Experience (P. U. R. E.)**

Apply to take part in the new Paid Undergraduate Research Experience (P. U. R. E.) program for paid academic-year research through Federal Work-Study!

The goal of this program is to help make research more accessible to FLI students and to set them up for success by enabling them to be compensated for their work.

---

Wed, Oct 27
5-6pm
On Zoom

https://stanford.zoom.us/j/99396091443?pwd=WUxRekt2SVppb3JBKzZQU3JVTXd1QT09

Right after class!
Your Questions
“I love all the interesting problems we covered in lectures so far, but I sometimes can't relate what we learn in class about writing proof and discrete math to the CS programming side. I know this is only halfway through the quarter, but will it be clearer about what we learn in class connect to real programming?”

The content for the rest of this quarter contains a bunch of gems that are immediately useful in programming. Automata, for example, are used in the design of UI elements, network controllers, compilers and interpreters, etc.

More generally, the techniques you've learned so far - how to formalize concepts, edge cases in formal logic, etc. - are surprisingly useful in designing complex systems. Building big systems is largely about getting the abstractions right. Concepts like functions, graphs, and the like are supremely useful here. MapReduce is a good example of this, as are the protocols that power the internet. And knowing FOL is useful for figuring out what to do in edge cases in systems.
Back to CS103!
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

$\sum \rightarrow q_0 \rightarrow a \rightarrow q_1 \rightarrow b \rightarrow q_2 \rightarrow a \rightarrow q_3$
Perfect Positive Guessing

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

\[ \Sigma \]

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

Start
Perfect Positive Guessing

\[ \Sigma \]

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

Input: 'aababaaba'
Perfect Positive Guessing
Perfect Positive Guessing

\[ q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{a} Q_3 \]

\[ \Sigma \]

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

(start)

q_0

q_1

q_2

Q_3
Perfect Positive Guessing

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

\[ \Sigma \]

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

Start

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

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

\[\Sigma \]

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

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

Input: \( \Sigma \)

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 a
Perfect Positive Guessing
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

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

Start

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

Input:

a b a b a b a
Massive Parallelism

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

Input alphabet: \( \Sigma = \{a, b\} \)

Sequence: \( ababaaba \)
Massive Parallelism

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

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

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

q₀ → q₁ → q₂ → q₃

\[ \Sigma \]

a b a b a b a
Massive Parallelism

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

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

Diagram shows states \( q_0, q_1, q_2, q_3 \) with transitions labeled by symbols: \( a \) and \( b \).
Massive Parallelism

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

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

- Start state: $q_0$
- Transitions:
  - $a \rightarrow q_1$
  - $b \rightarrow q_2$
  - $a \rightarrow q_3$
- Accepting state: $q_3$
- Input string: $a b a b a b a$
Massive Parallelism

-start-

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

\[ \Sigma \]

Input: a b a b a b a
Massive Parallelism

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

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

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

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

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

Start

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

Input sequence:

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

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

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

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

\[ \Sigma \]

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

\[
\begin{align*}
apababa
\end{align*}
\]
Massive Parallelism

a b a b a b a
Massive Parallelism

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

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

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

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

\[ \Sigma \]

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

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

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

\Sigma

a b a b a
Massive Parallelism

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

$\Sigma$

Input: $abaaba$
Massive Parallelism

\[ \Sigma \]

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

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

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

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

\[
\sum
\]

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

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

Input sequence:

\[ \text{abaaba} \]
Massive Parallelism

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

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

Sequence: $a \ b \ a \ b \ a \ b \ a$
Massive Parallelism

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

Input sequence: $a b a b a a$
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 alphabet: \( \sum \)
Massive Parallelism
Massive Parallelism

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

Start state: \( q_0 \)

Inputs: \( a, b \)

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

Transitions:
- \( q_0 \rightarrow q_1 \) on \( a \)
- \( q_1 \rightarrow q_2 \) on \( b \)
- \( q_2 \rightarrow q_3 \) on \( a \)
- \( q_3 \) is a trap state
Massive Parallelism

\[ \sum \]

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

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

\[ q_3 \]

\[ q_2 \]

\[ q_1 \]

\[ q_0 \]

\[ \sum \]

\[ \text{start} \]

\[ a \]

\[ b \]

\[ a \]

\[ b \]

\[ a \]

\[ b \]
Massive Parallelism

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

\[ \Sigma \]

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

\[ \Sigma \]

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

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

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

Start \( \Sigma \) a b a b
Massive Parallelism

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

\[ \sum \]

Input sequence: a b a b a b
Massive Parallelism

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

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

\[ \Sigma \]

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

Input: \[ ababa \]
Massive Parallelism
Massive Parallelism

\[ \sum \]

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

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

a b a b b
Massive Parallelism

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 sequence: $abaabb$
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
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

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

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

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

\[
\begin{array}{cccccc}
q_0 & q_1 & q_2 & q_3 \\
\rightarrow & a & b & a & \downarrow
\end{array}
\]
Massive Parallelism

Diagram:

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

Input sequence: ab a b a b
Massive Parallelism

```
Σ
```

```
start
```

```
q₀
```

```
q₁
```

```
q₂
```

```
q₃
```

```

```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
```
We’re not in any accepting state, so no possible path accepts.
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 ε-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 ε-transitions.
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

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

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

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

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

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

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

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

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

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

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

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

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

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

Nondeterministically **guess** which character is missing.

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

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

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

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

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

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

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

![Diagram of a non-deterministic finite automaton (NFA) recognizing the language $L$. The NFA starts at the 'start' state and can transition to states labeled 'a', 'b', and 'c' based on input symbols. The diagram also includes a transition to an accepting state upon reading 'c'.]
Just how powerful are NFAs?
Next Time

- **The Powerset Construction**
  - So beautiful. So elegant. So cool!
- **More Closure Properties**
  - Other set-theoretic operations.
- **Language Transformations**
  - What’s the deal with the notation $\Sigma^*$?