Finite Automata

Part Two
DFAs

- A **DFA** is a
  - Deterministic
  - Finite
  - Automaton

- 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 the alphabet.
- There is a unique start state.
- There are zero or more accepting states.
Recognizing Languages with DFAs

\[ L = \{ w \in \{0, 1\}^* \mid w \text{ contains 00 as a substring} \} \]
Recognizing Languages with DFAs

\[ L = \{ w \in \{0, 1\}^* \mid w \text{ contains 00 as a substring } \} \]
Recognizing Languages with DFAs

$L = \{ w \in \{0, 1\}^* | w \text{ contains } 00 \text{ as a substring } \}$
Recognizing Languages with DFAs

$L = \{ w \in \{0, 1\}^* \mid w \text{ contains 00 as a substring} \}$
Recognizing Languages with DFAs

$L = \{ w \in \{0, 1\}^* | \text{every other character of } w, \text{ starting with the first character, is } 0 \}$
More Elaborate DFAs

\[ L = \{ w \mid w \text{ is a C-style comment} \} \]

Suppose the alphabet is

\[ \Sigma = \{ a, *, / \} \]

Try designing a DFA for comments!

Some test cases:

**ACCEPTED**

- /*a*/
- /***/
- /****/
- /*aaa*/

**REJECTED**

- /***
- /***/a
- a/b/***/
- /*/*
More Elaborate DFAs

\[ L = \{ \ w \mid \ w \text{ is a C-style comment} \ \} \]
More Elaborate DFAs

\[ L = \{ w \mid w \text{ is a C-style comment} \} \]
More Elaborate DFAs

\[ L = \{ w \mid w \text{ is a C-style comment} \} \]
More Elaborate DFAs

\[ L = \{ w \mid w \text{ is a C-style comment} \} \]
More Elaborate DFAs

\[ L = \{ w \mid w \text{ is a C-style comment} \} \]
More Elaborate DFAs

$L = \{ w \mid w \text{ is a C-style comment} \}$
More Elaborate DFAs

\[ L = \{ w \mid w \text{ is a C-style comment} \} \]
More Elaborate DFAs

\[ L = \{ w \mid w \text{ is a C-style comment} \} \]
More Elaborate DFAs

\[ L = \{ w \mid w \text{ is a C-style comment} \} \]
More Elaborate DFAs

\[ L = \{ \ w \mid w \text{ is a C-style comment} \ \} \]
More Elaborate DFAs

\[ L = \{ \ w \mid w \text{ is a C-style comment} \ \} \]
Tabular DFAs

\begin{center}
\begin{tikzpicture}
    
    \node[state, initial] (q0) {$q_0$};
    \node[state] (q1) [right of=q0] {$q_1$};
    \node[state] (q2) [right of=q1] {$q_2$};
    \node[state] (q3) [right of=q2] {$q_3$};

    \path[->]
    (q0) edge node {0} (q1)
    (q1) edge node {1} (q2)
    (q2) edge node {0} (q3)
    (q3) edge[bend left] node {$\Sigma$} (q0);
\end{tikzpicture}
\end{center}
Tabular DFAs
Tabular DFAs

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

- Transition table:

<table>
<thead>
<tr>
<th>state</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

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

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

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

<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></td>
</tr>
<tr>
<td>$q_3$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Diagram:

- Start state: $q_0$
- Transitions:
  - $q_0$ on $0$ goes to $q_1$
  - $q_0$ on $1$ goes to $q_0$
  - $q_1$ on $1$ goes to $q_2$
  - $q_1$ on $0$ goes to $q_1$
  - $q_2$ on $0$ goes to $q_2$
  - $q_2$ on $1$ goes to $q_3$
  - $q_3$ on $0$ goes to $q_3$
  - $q_3$ on $1$ goes to $q_3$
  - $q_3$ on $\Sigma$ goes to $q_3$
Tabular DFAs

![Graph of 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></td>
<td></td>
</tr>
</tbody>
</table>
Tabular DFAs

<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></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

<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>
The star indicates that this is an accepting state.
Code? In a Theory Course?

```c
int kTransitionTable[kNumStates][kNumSymbols] = {
  {0, 0, 1, 3, 7, 1, ...},
  ...,
};

bool kAcceptTable[kNumStates] = {
  false,
  true,
  true,
  true,
  ...,
};

bool SimulateDFA(string input) {
  int state = 0;
  for (char ch: input)
    state = kTransitionTable[state][ch];
  return kAcceptTable[state];
}
```
A language $L$ is called a regular language iff there exists a DFA $D$ such that $L(D) = L$. 
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 \( \Sigma^* \) not in \( L \).
- Formally:
  \[
  \overline{L} = \{ w \mid w \in \Sigma^* \land w \notin L \}
  \]
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 $\Sigma^*$ not in $L$.

- Formally:
  $$\overline{L} = \{ w \mid w \in \Sigma^* \land w \notin L \}$$
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 \( \Sigma^* \) not 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 $\Sigma^*$ not 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 $\Sigma^*$ not in $L$.

- Formally:

$$\overline{L} = \Sigma^* - L$$
Complementing Regular Languages

• Recall: A regular language is a language accepted by some DFA.

• **Question:** If \( L \) is a regular language, is \( \overline{L} \) a regular language?

• If the answer is “yes,” then there must be some way to construct a DFA for \( \overline{L} \).

• If the answer is “no,” then some language \( L \) can be accepted by a DFA, but \( \overline{L} \) cannot be accepted by any DFA.
Complementing Regular Languages

\[ L = \{ w \in \{0, 1\}^* \mid w \text{ contains 00 as a substring} \} \]
Complementing Regular Languages

\[ L = \{ \ w \in \{0, 1\}^* \mid w \text{ contains } 00 \text{ as a substring} \} \]

\[ \overline{L} = \{ \ w \in \{0, 1\}^* \mid w \text{ does not contain } 00 \text{ as a substring} \} \]
Complementing Regular Languages

\[ L = \{ w \in \{0, 1\}^* \mid w \text{ contains 00 as a substring} \} \]

\[ \bar{L} = \{ w \in \{0, 1\}^* \mid w \text{ does not contain 00 as a substring} \} \]
More Elaborate DFAs

\[ L = \{ w \mid w \text{ is a C-style comment} \} \]
More Elaborate DFAs

\[ \overline{L} = \{ w \mid w \text{ is not a C-style comment} \} \]
More Elaborate DFAs

\[ \overline{L} = \{ \, w \mid w \text{ is not a C-style comment} \, \} \]
Closure Properties

- **Theorem:** If $L$ is a regular language, then $\overline{L}$ is also a regular language.
- If we begin with a regular language and complement it, we end up with a regular language.
- This is an example of a **closure property of regular languages**.
  - The regular languages are **closed under complementation**.
  - We'll see more such properties later on.
Time-Out For Announcements!
Solutions Released

• Solutions released for
  • Problem Set 4 checkpoint.
  • Practice Midterm 1.
  • Practice Midterm 2.

• Please review the solution sets for the Problem Set 4 checkpoint. There are some common mistakes you probably want to review.
Midterm Logistics

• Midterm review sessions this weekend.
  • **Saturday, 2:15PM - 4:15PM:** Michael holding an on-campus review session.
  • **Sunday, 8PM - 10PM:** Dilli holding a review session for SCPD students.

• Neither time works for you? Let us know so we can try to schedule something extra!

• We have a Google Moderator page to see what questions to answer during the review session; link will be posted later today.
Your Questions
“The only difference between predicates and functions is that the first 'returns' booleans and the second 'returns' objects. Are booleans not objects? If they aren't, why aren't they, and if they are, how are predicates and functions different?”
NFAs
NFAs

- An **NFA** is a
  - **Nondeterministic**
  - **Finite**
  - **Automaton**
- Conceptually similar to a DFA, but equipped with the vast power of **nondeterminism**.
(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 may have multiple decisions that it can make at one point.

- The machine accepts if *any* series of choices leads to an accepting state.
A Simple NFA

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

Transitions:
- \( q_0 \rightarrow q_1 \) on 1
- \( q_1 \rightarrow q_2 \) on 1
- \( q_3 \rightarrow q_0 \) on 0, 1
- \( q_3 \rightarrow q_1 \) on 0
- \( q_3 \rightarrow q_2 \) on 0, 1
- \( q_3 \) is a dead state.
A Simple NFA

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

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

Start

$q_0$  1  $q_1$

0, 1

1  $q_2$

$0, 1$

$q_3$

$0, 1$

$0, 1$

0 1 0 1 1
A Simple NFA

\[
\begin{array}{c}
\text{start} \\
q_0 & -1 & q_1 & -1 & q_2 \\
0, 1 & 1 & 0, 1 \\
\end{array}
\]

0 1 0 1 1
A Simple NFA

0 1 0 1 1
A Simple NFA
A Simple NFA

$q_0 \xrightarrow{1} q_1 \xrightarrow{1} q_2
q_0 \xrightarrow{0, 1} q_3
q_3 \xrightarrow{0, 1} q_2
q_3 \xrightarrow{0, 1} q_2

0 1 0 1 1 1
A Simple NFA

0 1 0 1 1
A Simple NFA

Start

$q_0$ 1 $q_1$

1 1 $q_2$

$q_3$

0, 1 0, 1

0, 1 0, 1

0 1 0 1 1
A Simple NFA

start

$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3$

0, 1

0, 1

0, 1

0

1

1

0

0

1

1

0

1

0

1

1
A Simple NFA
A Simple NFA

0 1 0 1 1
A Simple NFA

\begin{itemize}
  \item \texttt{q}_0 \rightarrow 1 \rightarrow \texttt{q}_1 \rightarrow 1 \rightarrow \texttt{q}_2
  \item \texttt{q}_0 \rightarrow 0, 1 \rightarrow \texttt{q}_3 \rightarrow 0, 1 \rightarrow \texttt{q}_2
  \item \texttt{q}_3 \rightarrow 0, 1 \rightarrow \texttt{q}_2
\end{itemize}
A Simple NFA

\begin{center}
\begin{tikzpicture}
  \node[state, initial] (q0) {$q_0$};
  \node[state] (q1) [right of=q0] {$q_1$};
  \node[state, accepting] (q2) [right of=q1] {$q_2$};
  \node[state, accepting] (q3) [below of=q2] {$q_3$};

  \path[->]
  (q0) edge node {1} (q1)
  (q1) edge node {1} (q2)
  (q0) edge [loop below] node {0, 1} (q0)
  (q1) edge [loop below] node {0} (q1)
  (q2) edge [loop right] node {0, 1} (q2)
  (q3) edge [loop right] node {0, 1} (q3)

\end{tikzpicture}
\end{center}

\textbf{0 1 0 1 1}
A Simple NFA

0 1 0 1 1
A Simple NFA
A Simple NFA

0 1 0 1 1
A Simple NFA
A Simple NFA

start

$q_0$ 1 $q_1$

1 $q_2$

0, 1

0 $q_3$

0, 1

0, 1

0 1 0 1 1

0 1 0 1 1
A Simple NFA
A Simple NFA

Start the automaton from state $q_0$. Transition with 1 to state $q_1$. Transition with 1 to state $q_2$. The automaton accepts the string 010111.
A Simple NFA

start

$q_0$ 1 0, 1

$q_1$ 1

$q_2$

$q_3$

0, 1

0, 1

0, 1

0 1 0 1 1 1
A Simple NFA

0 1 0 1 1
A Simple NFA

0 1 0 1 1
A Simple NFA

Start

$q_0 \rightarrow 1 \rightarrow q_1 \rightarrow 1 \rightarrow q_2 \rightarrow 0 \rightarrow q_3 \rightarrow 0, 1 \rightarrow q_2

SEAL

OF APPROVAL

0 1 0 1 1
A More Complex NFA
A More Complex NFA

If a NFA needs to make a transition when no transition exists, the automaton **dies** and that particular path rejects.
A More Complex NFA

Start

$q_0$ -> 1 -> $q_1$ -> 1 -> $q_2$

0, 1

0 1 0 1 1 1
A More Complex NFA

0 1 0 1 1
A More Complex NFA

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

- \( q_0 \): \( 0, 1 \) on \( 1 \)
- \( q_1 \): \( 1 \) on \( 1 \)
- \( q_2 \):
A More Complex NFA

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

Input: 0 1 0 1 1

States: \( q_0, q_1, q_2 \)
A More Complex NFA
A More Complex NFA
A More Complex NFA

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

start → $q_0$ with transitions:
- 1 from $q_0$ to $q_1$
- 0, 1 from $q_0$

$q_1$ with transition:
- 1 to $q_2$

$q_2$ is a final state.
A More Complex NFA
A More Complex NFA

- Start state: $q_0$
- Transitions:
  - From $q_0$ to $q_1$: on 1
  - From $q_0$ to $q_0$: on 0, 1
  - From $q_1$ to $q_2$: on 1
  - From $q_2$ to $q_2$: on any input
A More Complex NFA
A More Complex NFA
A More Complex NFA

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

0 1 0 1 1 1
A More Complex NFA

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

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

\begin{center}
\begin{tikzpicture}
  \node[state, initial] (q0) {$q_0$};
  \node[state] (q1) [right of=q0] {$q_1$};
  \node[state, accepting] (q2) [right of=q1] {$q_2$};
  \draw [->] (q0) -- node {$1$} (q1);
  \draw [->] (q1) -- node {$1$} (q2);
  \draw [->, bend left] (q0) to node [swap] {$0, 1$} (q1);
\end{tikzpicture}
\end{center}
A More Complex NFA

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

Input sequence: 0 1 0 1 1 1
A More Complex NFA

- Start state: $q_0$
- Transition: $q_0 \xrightarrow{1} q_1$
- Transition: $q_1 \xrightarrow{1} q_2$

Symbols: 0, 1

Label: SEALS OF APPROVAL
Intuiting Nondeterminism

- Nondeterministic machines are a serious departure from physical computers.
- How can we build up an intuition for them?
- Three approaches:
  - Tree computation
  - Perfect guessing
  - Massive parallelism
Tree Computation

$q_0$ arrow 0 to $q_1$
$q_1$ arrow 1 to $q_2$
$q_0$ arrow 1 to $q_3$
$q_3$ arrow 0 to $q_4$
$q_4$ arrow 0 to $q_5$
$q_4$ arrow 1 to $q_2$

Start state: $q_0$
Tree Computation

0 1 0 1 0
Tree Computation

Start

$q_0$ 0 $q_1$ 1 $q_2$

1

$q_3$ 0 $q_4$ 0 $q_5$

1

$q_4$ 0 $q_5$

0, 1

0 1 0 1 0
Tree Computation

0 1 0 1 0
Tree Computation

start

$q_0$ 0 $q_1$

1 $q_2$

$q_3$ 0 $q_4$

1 0 $q_5$

$q_0, 1$

0, 1

0 1 0 1 0
Tree Computation

\[
\begin{align*}
&\text{start} \\
&q_0 \quad 0 \quad q_1 \quad 1 \quad q_2 \\
&\quad 1 \quad 1 \\
&q_3 \quad 0 \quad q_4 \quad 0 \quad q_5 \\
&\quad 0, 1
\end{align*}
\]
Tree Computation

0 1 0 1 0
Tree Computation

0 1 0 1 0
Tree Computation

0 1 0 1 0

0, 1

1

0, 1

1
Tree Computation

0 1 0 1 0 0
Tree Computation

The diagram illustrates a tree computation with nodes labeled as follows: $q_0$, $q_1$, $q_2$, $q_3$, $q_4$, $q_5$. The arrows indicate transitions based on input symbols: 0 or 1. The start node is $q_0$, and the computation follows through the states based on the input sequence provided.
Tree Computation

0 1 0 1 0 1 0
Tree Computation

0 1 0 1 0 0
Tree Computation

0 1 0 1 0 0
Tree Computation

0 1 0 1 0 0
Nondeterminism as a Tree

- At each decision point, the automaton clones itself for each possible decision.
- The series of choices forms a directed, rooted tree.
- At the end, if any active accepting states remain, we accept.
Perfect Guessing

start

$q_0 \rightarrow 0 \rightarrow q_1 \rightarrow 1 \rightarrow q_2$

$q_3 \rightarrow 1 \rightarrow q_1 \rightarrow 1 \rightarrow q_4 \rightarrow 0 \rightarrow q_5$

$q_4 \rightarrow 0, 1 \rightarrow q_5$
Perfect Guessing

![State Diagram]

0 1 0 1 0
Perfect Guessing

- Start state: $q_0$
- Transitions:
  - $q_0 \rightarrow q_1$: on 0
  - $q_1 \rightarrow q_2$: on 1
  - $q_1 \rightarrow q_1$: on 1
  - $q_1 \rightarrow q_3$: on 1
  - $q_3 \rightarrow q_0$: on 0
  - $q_3 \rightarrow q_4$: on 0
  - $q_4 \rightarrow q_5$: on 0
  - $q_5$: self-loop on 0, 1

- Input sequence: 0 1 0 1 0
Perfect Guessing

0 1 0 1 0

$q_0$ 0 $q_1$ 1 $q_2$

$q_3$ 1 $q_1$ 1 $q_2$

$q_4$ 0 $q_5$

$q_5$ 0, 1

Start state

0, 1
Perfect Guessing

Diagram:

- **start**
- $q_0$ (0 -> 1)
- $q_1$ (0 -> 1, 1 -> 0)
- $q_2$ (0 -> 1, 1 -> 0)
- $q_3$ (0 -> 1, 1 -> 0)
- $q_4$ (0 -> 1, 1 -> 0)
- $q_5$ (0 -> 1, 1 -> 0)

Sequence of symbols: 0 1 0 1 0 0

Arrow with symbols: 1, 0, 0, 1
Perfect Guessing

start

$q_0$ 0 $q_1$ 1 $q_2$

1

$q_3$ 0 $q_4$ 0 $q_5$

1

0, 1

0 1 0 1 0
Perfect Guessing

Graph with states and transitions:

- Start state: $q_0$
- Transition: $q_0 \xrightarrow{0} q_1$
- Transition: $q_0 \xrightarrow{1} q_3$
- Transition: $q_1 \xrightarrow{0} q_4$
- Transition: $q_1 \xrightarrow{1} q_1$ (loop)
- Transition: $q_3 \xrightarrow{0} q_4$
- Transition: $q_3 \xrightarrow{1} q_5$
- Transition: $q_4 \xrightarrow{0} q_5$
- Transition: $q_2$ is a final state

Input sequence: 0 1 0 1 0
Perfect Guessing

Diagram:

- Start state: $q_0$
- States: $q_0, q_1, q_2, q_3, q_4, q_5$
- Transitions:
  - $q_0$ to $q_1$: 0
  - $q_1$ to $q_2$: 1
  - $q_1$ to $q_4$: 1
  - $q_3$ to $q_4$: 0
  - $q_4$ to $q_5$: 0
  - $q_4$ to $q_4$: 0, 1
  - $q_5$ is a sink state

Input sequence: 0 1 0 1 0
Perfect Guessing

\[
\begin{align*}
q_0 &\xrightarrow{0} q_1 & q_1 &\xrightarrow{1} q_2 \\
q_3 &\xrightarrow{1} q_4 & q_4 &\xrightarrow{0} q_5 \\
q_4 &\xrightarrow{0, 1} q_4
\end{align*}
\]
Perfect Guessing

0 1 0 1 0
Perfect Guessing

start

$q_0$ → $q_1$ (0) → $q_2$ (1)

$q_3$ (1) → $q_4$ (1) → $q_5$ (0)

$q_0$, $q_1$, $q_2$, $q_3$, $q_4$, $q_5$

0 1 0 1 0
Perfect Guessing

\[
\begin{array}{cccc}
q_0 & 0 & q_1 & 1 \\
1 & & 1 \\
q_3 & 0 & q_4 & 0 \\
0, 1 & & 0, 1 & \\
\text{start} & & q_5 & \\
\end{array}
\]
Perfect Guessing

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

Diagram:
- Start state: $q_0$
- Transitions:
  - $q_0$: 0 to $q_1$, 1 to $q_3$
  - $q_1$: 1 to $q_2$
  - $q_3$: 0 to $q_4$, 1 to $q_5$
  - $q_4$: 0 to $q_5$
  - $q_5$: Self-loop on 0, 1

States:
- $q_0$, $q_1$, $q_2$, $q_3$, $q_4$, $q_5$
Perfect Guessing

- We can view nondeterministic machines as having Magic Superpowers that enable them to guess the correct choice of moves to make.
- Idea: Machine can always guess a path that leads to an accepting state if one exists.
- No known physical analog for this style of computation.
Massive Parallelism

\[ q_0 \xrightarrow{0} q_1 \xrightarrow{1} q_2 \]
\[ q_3 \xrightarrow{1} q_1 \xrightarrow{1} q_4 \xrightarrow{0} q_5 \]

Input sequence: 0 1 0 1 0 0
Massive Parallelism

0 1 0 1 0
Massive Parallelism

\[
\begin{array}{c}
q_0 \xrightarrow{0} q_1 \xrightarrow{1} q_2 \\
q_3 \xrightarrow{0} q_4 \xrightarrow{0} q_5 \\
\end{array}
\]

Input: 0 1 0 1 0
Massive Parallelism

The diagram shows a state transition diagram with states $q_0, q_1, q_2, q_3, q_4,$ and $q_5$. The transitions are as follows:

- 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, return to $q_1$.
- From $q_3$, on input 1, go to $q_4$.
- From $q_4$, on input 0, go to $q_5$.
- From $q_5$, on input 0, return to $q_5$.

The arrow labeled with "0, 1" indicates the transition from $q_3$ to $q_4$.
Massive Parallelism
Massive Parallelism

0 1 0 1 0
Massive Parallelism

- Start: $q_0$
- Transition: $q_0 \rightarrow q_1$ with input 0
- Transition: $q_1 \rightarrow q_2$ with input 1
- Transition: $q_3 \rightarrow q_4$ with input 1
- Transition: $q_4 \rightarrow q_5$ with input 0

Input sequence: 0 1 0 1 0

States: $q_0, q_1, q_2, q_3, q_4, q_5$
Massive Parallelism

The diagram illustrates a state transition model for massive parallelism. The states are represented by circles labeled $q_0, q_1, q_2, q_3, q_4, q_5$, and the transitions are indicated by arrows with labels 0 and 1. The labels on the arrows show the transitions between states. The initial state is $q_0$, and the transitions follow the path as described by the labels.
Massive Parallelism

start

q_0 \quad 0 \quad q_1 \quad 1 \quad q_2

1

q_3 \quad 0 \quad q_4 \quad 0 \quad q_5

1

0, 1

0 1 0 1 0
Massive Parallelism

0101010
Massive Parallelism

Start

$q_0 \xrightarrow{0} q_1 \xrightarrow{1} q_2$

$q_3 \xrightarrow{1} q_1 \xrightarrow{1} q_4 \xrightarrow{0} q_5$

$q_4$ is a loop with transitions $0, 1$

Input sequence: 0 1 0 1 0
Massive Parallelism
Massive Parallelism

\begin{figure}
\centering
\begin{tikzpicture}
  \node[draw, circle] (q0) at (0,0) {$q_0$};
  \node[draw, circle] (q1) at (2,0) {$q_1$};
  \node[draw, circle] (q2) at (4,0) {$q_2$};
  \node[draw, circle] (q3) at (0,-2) {$q_3$};
  \node[draw, circle, fill=yellow] (q4) at (2,-2) {$q_4$};
  \node[draw, circle, fill=green] (q5) at (4,-2) {$q_5$};

  \path[->]
    (q0) edge node {0} (q1)
    (q0) edge node {1} (q3)
    (q1) edge node {1} (q4)
    (q3) edge node {0} (q4)
    (q4) edge[loop below] node {0, 1} (q4)
    (q4) edge node {0} (q5)
    (q2) edge[loop right] node {0} (q2);
\end{tikzpicture}
\end{figure}
Massive Parallelism

The diagram shows a finite state machine with states $q_0, q_1, q_2, q_3, q_4, q_5$. The transitions are labeled with input symbols 0 and 1, and there is a loop labeled with input symbols 0 and 1 from $q_3$ back to $q_3$. The start state is $q_0$. The input sequence is 010100.
Massive Parallelism

• An NFA can be thought of as a DFA that can be in many states at once.
• Each symbol read causes a transition on every active state into each potential state that could be visited.
• Nondeterministic machines can be thought of as machines that can try any number of options in parallel.
  • No fixed limit on processors; makes multicore machines look downright wimpy!
So What?

- We will turn to these three intuitions for nondeterminism more later in the quarter.
- 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.
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 $\varepsilon$-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.
\( \varepsilon \)-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.
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 \textbf{\(\varepsilon\)-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 \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.

![Diagram of ε-transitions and NFA transitions]

0 0 1 0 0
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 **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.
Designing NFAs

- When designing NFAs, *embrace the nondeterminism!*
- Good model: **Guess-and-check:**
  - Have the machine *nondeterministically guess* what the right choice is.
  - Have the machine *deterministically check* that the choice was 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

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

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

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

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

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

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

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

• **Equivalence of NFAs and DFAs**
  • How are DFAs powerful enough to match NFAs?

• **Additional Closure Properties**
  • Closure under union, intersection, concatenation, and Kleene star!

• **Regular Expressions I**
  • A different formalism for regular languages.