Recap from Last Time
Old MacDonald Had a Symbol, ☢ Σ-eye-ε-eyε, Oh! ☢

- You may have noticed that we have several letter-E-ish symbols in CS103, which can get confusing!

- Here’s a quick guide to remembering which is which:
  - Typically, we use the symbol Σ to refer to an alphabet.
  - The empty string is length 0 and is denoted ε.
  - In set theory, use ∈ to say “is an element of.”
  - In set theory, use ⊆ to say “is a subset of.”
DFAs

- A **DFA** is a
  - **D**eterministic
  - **F**inite
  - **A**utomaton

- DFAs are the simplest type of automaton that we will see in this course.
Recognizing Languages with DFAs

\[ L = \{ w \in \{a, b\}^* \mid w \text{ contains } aa \text{ as a substring} \} \]
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.
New Stuff!
Which table best represents the transitions for the DFA shown below?

(A)  
\[
\begin{array}{c|cc}
  \text{0} & \text{1} \\
  \hline
  q_0 & q_1 & q_0 \\
  q_1 & q_3 & q_2 \\
  q_2 & q_3 & q_0 \\
  q_3 & q_3 & q_3 \\
\end{array}
\]

(B)  
\[
\begin{array}{c|cc}
  \text{0} & \text{1} \\
  \hline
  q_0 & q_0 & q_1 \\
  q_1 & q_2 & q_3 \\
  q_2 & q_0 & q_3 \\
  q_3 & q_3 & q_3 \\
\end{array}
\]

(C)  
\[
\begin{array}{c|ccc}
  \text{} & \text{0} & \text{1} & \Sigma \\
  \hline
  q_0 & q_1 & q_0 & / \\
  q_1 & q_3 & q_2 & / \\
  q_2 & q_3 & q_0 & / \\
  q_3 & / & / & q_3 \\
\end{array}
\]

Answer at PollEv.com/cs103 or text CS103 to 22333 once to join, then A, B, C, or D (none of the above).
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

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

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

These stars indicate accepting states.
Tabular DFAs

Since this is the first row, it's the start state.
My Turn to Code Things Up!

```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];
}
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:

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

- As we saw a few minutes ago, a *regular language* is a language accepted by some DFA.
- **Question**: If $L$ is a regular language, is $\overline{L}$ necessarily a regular language?
- If the answer is “yes,” then if there is a way to construct a DFA for $L$, 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 some DFA, but $\overline{L}$ cannot be accepted by any DFA.
Computational Device for $L$

input

Yep!
Nope!
Computational Device for $L$

Yep!

Nope!

input

Yep!

Nope!
Computational Device for $L$

Computational Device for $\overline{L}$
Computational Device for $L$

Computational Device for $\overline{L}$
Complementing Regular Languages

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

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

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

$$\overline{L} = \{ \, w \in \{a, *, /\}^* \mid w \text{ doesn't represent a C-style comment} \, \}$$

![DFA Diagram]
More Elaborate DFAs

$$\bar{L} = \{ w \in \{a, *, /\}^* \mid w \text{ doesn't represent a C-style comment} \}$$
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*.
Time-Out For Announcements!
Additional Practice

• Looking to improve your performance in CS103? We’ve released two handouts:
  • Handout 31: How to Improve in CS103
  • Handout 32: Extra Practice Problems 2
• Set aside a few minutes each day to get some additional practice. That will add up extremely quickly!
STANFORD’S NATIONAL HACKATHON
THIS WEEKEND FEB 16-18
HUANG ENGINEERING CENTER
BEGINNERS WELCOME!
Ever felt you weren't good enough to be in STEM? Afraid of being "found out" because you don't think you belong?

Learn how to combat Imposter Phenomenon at a very special workshop led by Dr. Nicole Cabrera Salazar! Lunch will be served.

~Wed. 2/14, 11:30 am - 1:30 pm, PAB 232~
Back to CS103!
NFAs
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 may have multiple decisions that it can make at one point.

• 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

\[
\begin{array}{c}
\text{start} \\
\begin{array}{c}
q_0 \\
\downarrow 0, 1 \\
q_1 \\
\downarrow 1 \\
q_2 \\
\end{array}
\end{array}
\begin{array}{c}
q_1 \\
\downarrow 1 \\
q_2 \\
\end{array}
\begin{array}{c}
q_3 \\
\downarrow 0, 1 \\
q_3 \\
\end{array}
\begin{array}{c}
q_3 \\
\downarrow 0, 1 \\
q_2 \\
\end{array}
\end{array}
\]
A Simple NFA

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

start \rightarrow q_0 \rightarrow q_1 \rightarrow q_2

\begin{align*}
q_0 & \rightarrow 1, q_1 \\
q_1 & \rightarrow 1, q_2 \\
q_2 & \rightarrow 0, 1 \\
q_3 & \rightarrow 0, 1
\end{align*}

0, 10, 11
A Simple NFA

0 1 0 1 1 1
A Simple NFA

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

- Start state: $q_0$
- Transitions:
  - $q_0$ to $q_1$: 1
  - $q_0$ to $q_3$: 0, 1
  - $q_1$ to $q_2$: 1
  - $q_1$ to $q_3$: 0
  - $q_3$ to $q_2$: 0, 1

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

start

$q_0$ 1 $q_1$

0, 1

$q_2$

1

$q_3$

0

0, 1

0, 1

0, 1

0, 1

0 1 0 1 1
A Simple NFA

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

Start state: $q_0$

$0, 1$ from $q_0$ to $q_1$
$1$ from $q_1$ to $q_2$

$0$ from $q_1$ to $q_3$
$0, 1$ from $q_3$ to $q_2$

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

Start state: $q_0$

- From $q_0$: 1 to $q_1$, 0, 1 to $q_3$
- From $q_1$: 1 to $q_2$
- From $q_3$: 0 to $q_2$, 0, 1 to $q_3$
- $q_2$: Final state

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

Start

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

\[ q_3 \xrightarrow{0} q_3 \]

Input:

\[ 01011 \]
A Simple NFA

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

0 1 0 1 1
A Simple NFA

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

Input: 0 1 0 1 1
A Simple NFA

0 1 0 1 1

0 1 0 1 1
A Simple NFA

Start

$q_0$

$q_1$

$q_2$

$q_3$

0, 1

0

0, 1

0, 1

0, 1

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

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

0 1 0 1 1
A Simple NFA

0 1 0 1 1
A Simple NFA

-start-

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

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

$q_3 \xleftarrow{0, 1} q_2$

Input sequence: 0 1 0 1 1
A Simple NFA

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

transitions:

- From \(q_0\) to \(q_1\) on input 1
- From \(q_1\) to \(q_2\) on input 1
- From \(q_0\) to \(q_3\) on input 0, 1
- From \(q_3\) to \(q_2\) on input 0
- From \(q_1\) to \(q_3\) on input 0, 1
- From \(q_3\) to \(q_3\) on input 0, 1

Input sequence: 0 1 0 1 1
A Simple NFA

start

$q_0$ 1 $q_1$

$q_1$ 1 $q_2$

$q_0$ 0 $q_3$

$q_3$ 0, 1 $q_2$

$q_3$ 0, 1 $q_2$

$q_2$ 0, 1 $q_2$

$q_2$ 0, 1 $q_2$

$q_2$ 0, 1 $q_2$

$q_2$ 0, 1 $q_2$

$q_2$ 0, 1 $q_2$

$q_2$ 0, 1 $q_2$
A More Complex NFA

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

\[
q_0 \xrightarrow{0, 1} q_1
\]
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

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

\[ \text{start} \]

Input:
0 1 0 1 1
A More Complex NFA

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

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

- Start state: $q_0$
- Transitions:
  - From $q_0$: 1 to $q_1$
  - From $q_1$: 1 to $q_2$
  - From $q_0$: 0, 1 to $q_0$
- States: $q_0, q_1, q_2$
- Input symbols: 0, 1

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

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

Oh no! There's no transition defined!

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

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

0 1 0 1 1
A More Complex NFA

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

Start: \( q_0 \)

Input sequence: 010111
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 $q_0$ to $q_1$ on input $1$.
- From $q_1$ to $q_2$ on input $1$.
- From $q_1$ to itself on input $0$, $1$.

The input string $010111$ starts at state $q_0$ and follows the transitions to reach $q_2$.
A More Complex NFA
A More Complex NFA

\[
\begin{align*}
&\text{start} \\
&\quad \rightarrow q_0 \quad 1 \quad q_1 \quad 1 \quad q_2 \\
&\quad \quad \quad \quad \quad \quad 0, 1
\end{align*}
\]
A More Complex NFA
A More Complex NFA
A More Complex NFA

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

0 1 0 1 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:
- Start at $q_0$ and move to $q_1$ on input 1.
- From $q_1$, move to $q_2$ on input 1.
- From $q_0$, there is a loop on input 0, 1.

The accepted strings are 0 1 0 1 1.
A More Complex NFA

Start

$q_0$ 1 $q_1$ 1 $q_2$

Seal

Of Approval

0 1 1
As with DFAs, the language of an NFA $N$ is the set of strings that $N$ accepts:

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

What is the language of the NFA shown above?

A. $\{01011\}$
B. $\{ w \in \{0, 1\}^* \mid w \text{ contains at least two } 1\text{s} \}$
C. $\{ w \in \{0, 1\}^* \mid w \text{ ends with } 11 \}$
D. $\{ w \in \{0, 1\}^* \mid w \text{ ends with } 1 \}$
E. None of these, or two or more of these.
NFA Acceptance

• An NFA $N$ accepts a string $w$ if there is some series of choices that lead to an accepting state.

• Consequently, an NFA $N$ rejects a string $w$ if no possible series of choices lead it into an accepting state.

• It's easier to show that an NFA does accept something than to show that it doesn't.
ε-Transitions

- NFAs have a special type of transition called the ε-transition.
- An NFA may follow any number of ε-transitions at any time without consuming any input.
ε-Transitions

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

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

- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.
**ε-Transitions**

- NFAs have a special type of transition called the **ε-transition**.
- An NFA may follow any number of ε-transitions at any time without consuming any input.
ε-Transitions

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

\[\text{start} \quad q_0 \quad a \quad a \quad q_1 \quad a \quad a \quad q_2 \quad \varepsilon \quad q_3 \quad b, \varepsilon \quad \varepsilon \quad b \quad q_4 \quad b \quad q_5 \quad b \quad a \quad a \quad b \quad b \quad b\]
**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.
- NFAs are not *required* to follow ε-transitions. It's simply another option at the machine's disposal.
Suppose we run the above NFA on the string $10110$. How many of the following statements are true?

- There is at least one computation that finishes in an accepting state.
- There is at least one computation that finishes in a rejecting state.
- There is at least one computation that dies.
- This NFA accepts $10110$.
- This NFA rejects $10110$.

Answer at PollEv.com/cs103 or text CS103 to 22333 once to join, then a number.
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 guessing*
  • *Massive parallelism*
Perfect Guessing
Perfect Guessing

$\Sigma$

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

$\begin{align*}
a & \quad b & \quad a & \quad b & \quad a & \quad b & \quad a
\end{align*}$
Perfect Guessing

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

\[ \Sigma \]

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

\[ \uparrow \]
Perfect Guessing
Perfect Guessing
Perfect Guessing

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

\[\Sigma \]

Input sequence: a b a b a b a
Perfect Guessing

$\Sigma$
Perfect Guessing

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

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

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

\[\Sigma\]

The sequence of inputs is: a b a b a b a a
Perfect Guessing

q₀ → q₁ → q₂ → q₃

a a b a b a a

Σ
Perfect 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.
- No known physical analog for this style of computation – this is totally new!
Massive Parallelism

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

abababa
Massive Parallelism

\[ \Sigma \]

- Start at \( q_0 \)
- Transition: \( q_0 \xrightarrow{a} q_1 \)
- Transition: \( q_1 \xrightarrow{b} q_2 \)
- Transition: \( q_2 \xrightarrow{a} q_3 \)

Input sequence: \( abaabbaa \)
Massive Parallelism
Massive Parallelism

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

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

Input string: $abaaba$
Massive Parallelism

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

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

Σ

start

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

a b a b a a
Massive Parallelism

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

\[ \Sigma \]

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

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

- Start state: $q_0$
- Transitions: $a \rightarrow q_1$, $b \rightarrow q_2$, $a \rightarrow q_3$
- String: a b a b a b a
Massive Parallelism

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

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

$\Sigma$

Start

Input: a b a b a
Massive Parallelism

\[ \Sigma \]

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

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

\[ \Sigma \]

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

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

$$\Sigma$$

start

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

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

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

\[ \Sigma \]

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

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

Input sequence: ababaaba
Massive Parallelism

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

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

Path: \( a, b, a, b, a \)
Massive Parallelism

\[ \sum \]

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

Input:

a b a b a b a
Massive Parallelism

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

\[ \Sigma \]

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

\[
\sum
\]

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

Input sequence: \texttt{a b a b a b a}
Massive Parallelism

\[ \Sigma \]

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

From start to \( q_0 \) to \( q_1 \) to \( q_2 \) to \( q_3 \)
Massive Parallelism

a b a b a b a
Massive Parallelism

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

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

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

\[ \Sigma \]

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

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

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

Input sequence: \text{abaaba}
Massive Parallelism

\[ \sum \]

start \rightarrow q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3

a b a b a b a
Massive Parallelism

\[ \sum \]

Start

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

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

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

\[ \Sigma \]

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

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

\[ q_3 \]

\[ q_2 \]

\[ q_1 \]

\[ q_0 \]

\[ \Sigma \]

\[ \text{start} \]

\[ \text{a} \]

\[ \text{b} \]

\[ \text{a} \]

\[ \text{b} \]

\[ \text{a} \]

\[ \text{a} \]

\[ \text{a} \]
We’re in at least one accepting state, so there’s some path that gets us to an accepting state.

Therefore, we accept!
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.
So What?

• Each intuition of nondeterminism is useful in a different setting:
  • Perfect guessing is a great way to think about how to design a machine.
  • Massive parallelism is a great way to test machines – and has nice theoretical implications.

• Nondeterministic machines may not be feasible, but they give a great basis for interesting questions:
  • Can any problem that can be solved by a nondeterministic machine be solved by a deterministic machine?
  • Can any problem that can be solved by a nondeterministic machine be solved efficiently by a deterministic machine?

• The answers vary from automaton to automaton.
Designing NFAs
Designing NFAs

• When 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

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

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

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

Nondeterministically guess when to leave the start state.
Deterministically check whether that was the right time to do so.
Guess-and-Check

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

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

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

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

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

\begin{align*}
&\quad \quad a, b \\
&\quad \quad a, c
\end{align*}
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

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

Nondeterministically guess which character is missing.

Deterministically check whether that character is indeed missing.
Just how powerful are NFAs?