Nondeterministic Finite Automata

8 September 2022
Assignment 1

Corrections due today

Assignment 2

Out tonight
Due Tuesday
Corrections due Thursday
Where are we?
Formal language theory

An *alphabet* is a set, denoted $\Sigma$, whose elements are called *characters*.

A *string over $\Sigma$* is a finite sequence of zero or more characters drawn from $\Sigma$.

The *empty string*, denoted $\varepsilon$, has no characters.

A *language over $\Sigma$* is a set of strings over $\Sigma$.

The language $\Sigma^*$ is the set of all strings over $\Sigma$. 

DFAs

A *deterministic finite automaton* (DFA) is a simple model of computation, 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!

There is a unique start state.

There are zero or more accepting states.
Warm-up

EXERCISE  Design an automaton to recognize the language of strings that start and end with the same symbol. Let $\Sigma = \{a, b\}$. 
A sample DFA

$L = \{w \in \{0, 1\}^* \mid w \text{ contains } 11 \text{ as a substring}\}$
A sample DFA

\[ L = \{ w \in \{0, 1\}^* \mid w \text{ contains 11 as a substring}\} \]
Another way we can write down a DFA is as a transition table:

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

![Diagram of DFA](image-url)
Another way we can write down a DFA is as a transition table:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \rightarrow q_0 )</td>
<td>( q_0 )</td>
<td>( q_1 )</td>
</tr>
<tr>
<td>( q_1 )</td>
<td>( q_0 )</td>
<td>( q_2 )</td>
</tr>
<tr>
<td>*</td>
<td>( q_2 )</td>
<td>( q_2 )</td>
</tr>
</tbody>
</table>

By marking the start state with \( \rightarrow \) and accepting states with *, the transition table that defines \( \delta \) also specifies the entire DFA.
Tabular DFAs suggest how easy it is to implement a DFA in software.

```python
transition_table = {
    "q0": {"0": "q0", "1": "q1"},
    "q1": {"0": "q0", "1": "q2"},
    "q2": {"0": "q2", "1": "q2"}
}

accept_states = ["q2"]

def run_dfa(word: str) -> bool:
    state = "q0"
    for char in word:
        state = transition_table[state][char]
    return state in accept_states
```
Formal definition of DFAs

A DFA is represented as a five-tuple \((Q, \Sigma, \delta, q_0, F)\) where

- \(Q\): Finite set of states.
- \(\Sigma\): Finite set of input symbols (the alphabet).
- \(\delta\): \(Q \times \Sigma \rightarrow Q\): A transition function.
- \(q_0 \in Q\): One state is the start state.
- \(F \subseteq Q\): Set of zero or more accept states (or final states).
If $D$ is a DFA that processes strings over $\Sigma$, the *language of $D$*, denoted $L(D)$ is the set of all strings $D$ accepts:

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

If $L(D) = L$, we say that $D$ *recognizes* the language $L$. 
A language $L$ is called a *regular language* if there exists a DFA $D$ such that $L(D) = L$. 
Beware of type errors

A major source of confusion when dealing with automata – or mathematics in general – is making “type errors”.

Example: The start state $q_0$ is of type “state”, but the accepting states $F$ is of type “set of states”.

Example: Don’t confuse a DFA $D$ (essentially, a program) with $L(D)$, which is a language (set of strings).
Nondeterministic finite automata
All of the computers we’ve seen so far are **deterministic finite automata** (DFAs)

A model of computation is *deterministic* if, at every point in the computation, there is exactly *one choice* it can make.

A model of computation is *nondeterministic* if the machine has *zero or more* decisions it can make at one point.
"q_0 has two transitions defined on 1."

"q_1 has no transitions defined on 0."
Nondeterministic finite automata are structurally similar to DFAs, but they represent a fundamental shift in how we’ll think about computation.

The present state does not determine the next state; there are multiple possible futures!

An NFA accepts if any series of choices leads to an accepting state.
A simple NFA
A simple NFA

0, 1

\( q_0 \) 1 \( q_1 \) 1 \( q_2 \)

0, 1

\( q_3 \)

0, 1

0 1 0 1 1
A simple NFA

\begin{center}
\begin{tikzpicture}
  \node[state, fill=green] (q0) at (0,0) {$q_0$};
  \node[state, fill=white] (q1) at (2,0) {$q_1$};
  \node[state, fill=white, double] (q2) at (4,0) {$q_2$};
  \node[state, fill=white] (q3) at (2,-2) {$q_3$};

  \draw[->] (q0) edge[loop above] node {0, 1} (q0);
  \draw[->] (q0) edge[bend left] node {1} (q1);
  \draw[->] (q1) edge[bend left] node {1} (q2);
  \draw[->] (q1) edge[loop below] node {0} (q1);
  \draw[->] (q2) edge[loop above] node {0, 1} (q2);
  \draw[->] (q3) edge[loop above] node {0, 1} (q3);
  \draw[->] (q3) edge[bend left] node {0, 1} (q1);
  \draw[->] (q3) edge[bend left] node {0, 1} (q2);
  \draw[->] (q3) edge[bend right] node {0, 1} (q0);
\end{tikzpicture}
\end{center}
The diagram depicts a finite automaton with the following transitions:

- Start state: $q_0$
- Transition from $q_0$ on input $0, 1$ to $q_0$
- Transition from $q_0$ on input $1$ to $q_1$
- Transition from $q_1$ on input $1$ to $q_2$
- Transition from $q_2$ on input $0, 1$ back to $q_2$
- Transition from $q_2$ on input $0, 1$ to $q_3$
- Transition from $q_3$ on input $0, 1$ back to $q_3$
- Transition from $q_3$ on input $0, 1$ to $q_1$

The input string $01011$ is shown below the diagram, indicating that the automaton processes this string starting from the start state $q_0$.
A deterministic finite automaton (DFA) with states:

- Start state: $q_0$
- States: $q_0$, $q_1$, $q_2$, $q_3$

Transitions:

- From $q_0$:
  - $0, 1$ to $q_0$
  - $1$ to $q_1$

- From $q_1$:
  - $0$ to $q_3$
  - $1$ to $q_2$

- From $q_2$:
  - Self-loop on $0, 1$

- From $q_3$:
  - Self-loop on $0, 1$

Input sequence: 0 1 0 1 1
The given diagram represents a finite automaton with the following states: $q_0$, $q_1$, $q_3$, and $q_2$. The transitions are as follows:

- From $q_0$, on reading $0$, remain in $q_0$, and on reading $1$, move to $q_1$.
- From $q_1$, on reading $1$, move to $q_2$.
- From $q_2$, on reading $0$, remain in $q_2$, and on reading $1$, move to $q_3$.
- From $q_3$, on reading $0$, remain in $q_3$, and on reading $1$, move to $q_0$.
- The start state is $q_0$.

The black box at the bottom contains the sequence $01011$. This sequence is recognized by the automaton, as the path from the start state $q_0$ to $q_2$ through $q_1$ and $q_3$ matches the sequence.
0, 1

0, 1

1

0

0, 1

0, 1

q0

q1

q2

q3

start

0 1 0 1 1
The diagram represents a finite automaton with the following states:

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

The input sequence is: $01011$
The given DFA follows the transition rules:

- From start to $q_0$: Input 1
- From $q_0$ to $q_1$: Input 1
- From $q_1$ to $q_2$: Input 1
- From $q_1$ to $q_3$: Input 3
- From $q_3$ to $q_2$: Input 0, 1

The black block shows the input sequence: 0 1 0 1 1
A more complex NFA

- Initial state: $q_0$
- Transitions:
  - $q_0 \xrightarrow{0, 1} q_0$
  - $q_0 \xrightarrow{1} q_1$
  - $q_1 \xrightarrow{1} q_2$
  - $q_2$ is a final state
A more complex NFA

If an NFA needs to make a transition when no transition exists, the automaton dies and that particular path does not accept.
0 1 0 1 1
A finite automaton with the following states and transitions:

- **States:** q₀, q₁, q₂
- **Transitions:**
  - q₀ -> q₁ on input 1
  - q₁ -> q₂ on input 1
  - q₀ -> q₀ on input 0, 1

The automaton starts at q₀. The input sequence is 0 1 0 1 1.
The diagram represents a finite state machine (FSM) with three 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'.
- There is a loop from $q_0$ to $q_0$ on input '0' and '1'.

The initial state is $q_0$ (indicated by "start").
Nowhere to go!

0, 1

0 1 0 1 1
A finite automaton with the following states and transitions:

- States: q0, q1, q2
- Transitions:
  - From q0 to q1 on input 1
  - From q1 to q2 on input 1
  - Self-loop on q0 with input 0 or 1

Input sequence: 0 1 0 1 1
The language of an NFA is

\[ L(N) = \{ w \in \Sigma^* \mid N \text{ accepts } w \} \]
Let $\Sigma = \{a, b\}$.

What’s the language of this NFA?
Let $\Sigma = \{a, b\}$.

What’s the language of this NFA?

$L = \{ab\}$
Let $\Sigma = \{a, b\}$.

What’s the language of this NFA?
Let $\Sigma = \{a, b\}$.

What’s the language of this NFA?

$$L = \{w \in \Sigma^* \mid ab \text{ is a suffix of } w\}$$
For DFAs, you must read a symbol in order for the machine to make a move.

However, NFAs can move without consuming an input symbol – an \( \varepsilon \)-transition.

An NFA can follow any number of \( \varepsilon \)-transitions at any time without consuming any input.
Example

Transition Table:

<table>
<thead>
<tr>
<th>Input</th>
<th>q</th>
<th>r</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>ε</td>
<td>ε</td>
</tr>
</tbody>
</table>

Transition Diagram:

- Start state: q
- Transitions:
  - q to r on input 1
  - q to s on input ε
  - r to q on input 0
  - r to s on input ε
  - s to q on input 1
Example

```
0 0 1
```

```
0
```

```
\varepsilon
```

```
\varepsilon
```

```
start
```

```
s
```

```
qu 1 r \varepsilon s
```

```
1
```

```
0
```

```
0
```

```
1
```

```
1
```

```
\varepsilon
```

```
\varepsilon
```

```
\varepsilon
```

```
\varepsilon
```
Example

0 0 1

0 ε

ε

ε

0 1
Example

- **Transition Matrix:**
  - \[ \begin{bmatrix} 0 & 0 & 1 \\ 0 & \varepsilon & 0 \end{bmatrix} \]

- **Diagram:**
  - **States:** \( q, r, s \)
  - **Start State:** \( q \)
  - **Transition Arrows:**
    - \( q \) to \( r \): \( 1 \)
    - \( q \) to \( s \): \( 0 \)
    - \( r \) to \( s \): \( \varepsilon \)
    - \( r \) to \( q \): \( \varepsilon \)
    - \( s \) to \( r \): \( \varepsilon \)
Example

\begin{array}{ccc}
0 & 0 & 1 \\
0 & \varepsilon & 0 & 1 \\
\end{array}

\begin{tikzpicture}
  \node[state, initial] (q) at (0,0) {$q$};
  \node[state] (r) at (2,0) {$r$};
  \node[state, accepting] (s) at (4,0) {$s$};
  \draw[->] (q) edge node {$1$} (r);
  \draw[->] (r) edge node {$\varepsilon$} (s);
  \draw[->] (s) edge node {$\varepsilon$} (r);
  \draw[->] (q) edge[loop below] node {$\varepsilon$} (q);
  \draw[->] (q) edge[loop above] node {$0$} (q);
  \draw[->] (q) edge[loop right] node {$1$} (q);
\end{tikzpicture}
Example

\[
\begin{array}{ccc}
0 & 0 & 1 \\
0 & \varepsilon & 0 & 1 & \varepsilon
\end{array}
\]

\[
\begin{align*}
& q \xrightarrow{1} r \\
& r \xrightarrow{\varepsilon} s
\end{align*}
\]

\[
\begin{align*}
& q \xrightarrow{0} q \\
& r \xrightarrow{0} r & r \xrightarrow{\varepsilon} r
\end{align*}
\]

\[
\begin{align*}
& q \xleftarrow{1} s \\
& r \xleftarrow{0} q & r \xleftarrow{\varepsilon} r
\end{align*}
\]
NFAs are not required to follow $\varepsilon$-transitions; they’re just another choice of path for the computation.
Formally, an NFA is defined like a DFA:

\[ N = (Q, \Sigma, \delta, q_0, F) \]

Except now the output of the transition function \( \delta \) – e.g., \( \delta(q_0, a) \) – isn’t a single state but a set of states.
Thinking about NFAs
Nondeterministic machines are a serious departure from physical computers.

There are two helpful ways to think about nondeterministic computation:

- Perfect positive guessing
- Massive parallelism
Perfect positive guessing

\[ q_0, a, b, q_1, b, q_2, a, q_3 \]
Perfect positive guessing

\begin{center}
\begin{tikzpicture}[node distance=2cm]
  \node (q0) [initial] {$q_0$};
  \node (q1) [right of=q0] {$q_1$};
  \node (q2) [right of=q1] {$q_2$};
  \node (q3) [right of=q2,accepting] {$q_3$};

  \path[->]
  (q0) edge node {a} (q1)
  (q1) edge node {b} (q2)
  (q2) edge node {a} (q3)
  (q3) edge [loop above] node {a, b} (q3);
\end{tikzpicture}
\end{center}

\begin{center}
\textcolor{black}{a \ b \ a \ b \ a}
\end{center}
Perfect positive guessing
Perfect positive guessing

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

Input sequence: \texttt{a b a b a}
Perfect positive guessing

\[ a, b \]

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

\[ a, b, a, b, a \]
Perfect positive guessing

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

Start: \[q_0\] a, b

Symbols: a, b

States: q_0, q_1, q_2, q_3

Transition:
- q_0: a → q_1
- q_1: b → q_2
- q_2: a → q_3

Input sequence: ababa
Perfect positive guessing

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

\[ a, b \]
Perfect positive guessing

\[
\begin{align*}
\text{start} & \quad q_0 \quad a, b \\
& \quad q_1 \quad a \quad b \\
& \quad q_2 \quad a \\
& \quad q_3 \\
\end{align*}
\]
Perfect positive guessing

- States: $q_0, q_1, q_2, q_3$
- Transitions: $a, b$
- Start state: $q_0$
- Accepting state: $q_3$

Input sequence: $a b a b a$
Perfect positive guessing

a, b

start

q₀ → a → q₁ → b → q₂ → a → q₃

a b a b a
Perfect positive guessing

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

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

Next state: \texttt{q3}
Perfect positive guessing

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

a b a b a
Perfect positive guessing

Illustration by Gemma Correll
NFAs have a “Liquid Luck” potion
Perfect positive guessing

We can think of nondeterministic machines as having *magic powers* that enable them to guess the correct choice of moves to make.

If there is at least one choice leading to an accepting state for the input, the machine will guess it.

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

There’s no known way to physically model this intuition for nondeterminism; we have left reality.
Massive parallelism
Massive parallelism

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

    \path [->]
    (q0) edge node {a} (q1)
    (q1) edge node {b} (q2)
    (q2) edge node {a} (q3)
    (q0) edge [loop above] node {a, b} (q0);
\end{tikzpicture}
\end{figure}

\begin{itemize}
    \item a
    \item b
    \item a
    \item b
    \item a
\end{itemize}
Massive parallelism

\[ a, b \]

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

\[ a \; b \; a \; b \; a \]
Massive parallelism

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

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

\[ a, b \]

\[ \text{start} \]

\[ q_0 \]

\[ q_1 \]

\[ q_2 \]

\[ q_3 \]

\[ a \]

\[ b \]

\[ a \]
Massive parallelism
Massive parallelism

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

\text{a b a b a}
Massive parallelism

- States: $q_0$, $q_1$, $q_2$, $q_3$
- Transitions: $a$, $b$
- Initial state: $q_0$
- Accepting state: $q_3$

Input: $a b a b a$
Massive parallelism

\begin{align*}
\text{a, b} & \quad \text{a, b} \\
\text{a} & \quad \text{b} \\
\text{a} & \quad \text{a} \\
\end{align*}
Massive parallelism

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

Input sequence: a b a b a
Massive parallelism
Massive parallelism

- Start state: $q_0$
- Transitions:
  - $a$: $q_0 \rightarrow q_1$
  - $b$: $q_1 \rightarrow q_2$
- Accepting state: $q_3$

Input sequence: $a b a b a$
Massive parallelism

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

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

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

\[ \text{a, b} \]

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

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

\[ a, b \]

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

\[ a \ b \ a \ b \ a \]
Massive parallelism
Massive parallelism
Massive parallelism
Massive parallelism
Massive parallelism
Massive parallelism

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

\begin{array}{cccccc}
a & b & a & b & a
\end{array}
Massive parallelism

a, b

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

a b a b a
Massive parallelism

q0 \rightarrow a \rightarrow q1 \rightarrow b \rightarrow q2 \rightarrow a \rightarrow q3

\text{start} \rightarrow a, b

a b a b a
Massive parallelism
Massive parallelism

Graph showing a transition diagram with states labeled $q_0$, $q_1$, $q_2$, and $q_3$. The transitions are labeled with $a$, $b$, and arrows indicating the direction of transitions.

Input sequence: $a, b, a, b, a$
Massive parallelism

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

\(a\ b\ a\ b\ a\)

Massive parallelism
Massive parallelism

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

Input sequence: a b a b a
Massive parallelism

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

\[ a, b \]

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

\[ a \ b \ a \ b \ a \]
Massive parallelism

The diagram shows a state machine with states $q_0, q_1, q_2,$ and $q_3$. The machine starts at $q_0$ and transitions to $q_1$ upon reading 'a', then to $q_2$ upon reading 'b', and finally to $q_3$ upon reading 'a' again. The states $q_0$ and $q_3$ are connected by a loop labeled 'a, b', indicating that the machine can stay in $q_0$ or transition to $q_3$ upon reading 'a' or 'b'. The sequence 'a b a b a' is shown to demonstrate the path through the states.
Massive parallelism

1. Start at state $q_0$.
2. Transition to state $q_1$ on input $a$.
3. Transition to state $q_2$ on input $b$.
4. Transition back to state $q_0$ on input $a$.
5. Transition to state $q_3$ on input $a$.

Input sequence: $a b a b a$
Massive parallelism

```
a b a b a
```
Massive parallelism

\[ a, b \]

\[ \text{start} \]

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

\[ a \ b \ a \ b \ a \]
Massive parallelism
One of the states we’re in is an accepting state, so there is a path where the NFA accepts the input string.
One of the states we’re in is an accepting state, so there is a path where the NFA accepts the input string.
The future was and is massive parallelism.
Massive parallelism

An NFA can also 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.
Two roads diverged in a wood, and I –
both of them, at the same time, like a boss
I took the one less traveled by,

And that has made all the difference.

Robert Frost
Perfect guessing is a helpful way to think about how to design a machine to recognize a language.

Massive parallelism is a great way to test machines, and it has nice theoretical implications.
Language of an NFA:

An NFA accepts an input string $w$ if any path from the start state to an accepting state is labeled $w$. 
Designing NFAs
Embrace the nondeterminism.

A good approach is **guess-and-check**:

Is there some information you’d like to have?

Have the machine *nondeterministically guess* that information.

Then have it *deterministically check* that the choice was right, i.e., filter out the bad guesses.

The **guess** phase corresponds to trying lots of different options.

The **check** phase corresponds to filtering out bad guesses or wrong options.
$L = \{ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \}
\[ L = \{ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \} \]
$L = \{w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101\}$
\[ L = \{ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \} \]
$$L = \{w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101\}$$

Nondeterministically guess when the end of the string is coming up. Deterministically check whether you were correct.
$L = \{ w \in \{0, 1\}^* \mid w \text{ ends in 010 or 101} \}$
$L = \{ w \in \{0, 1\}^* | w \text{ ends in } 010 \text{ or } 101 \}$
\[ L = \{w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101\} \]

NFA diagram:
- Start state:
- Transitions:
  - \(0\) to \(1\)
  - \(1\) to \(0\)
  - \(0\) to \(1\)
- Accepting states:

Example input string: \(10101010\)
\[ L = \{ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \} \]
\[ L = \{ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \} \]
\[ L = \{ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \} \]
$L = \{ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \}$
$L = \{ w \in \{0, 1\}^* \mid w \text{ ends in 010 or 101} \}$
\[ L = \{w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101\} \]
$$L = \{ w \in \{0, 1\}^* \mid w \text{ ends in 010 or 101} \}$$
\[ L = \{ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \} \]
\[ L = \{ w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101 \} \]
\[ L = \{w \in \{0, 1\}^* \mid w \text{ ends in } 010 \text{ or } 101\} \]
\[ L = \{ w \in \{a, b, c\}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w \} \]
$L = \{ w \in \{ a, b, c \}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w \}$
\[ L = \{ w \in \{a, b, c\}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w \} \]
$L = \{ w \in \{a, b, c\}^* \mid \text{at least one of } a, b, \text{ or } c \text{ is not in } w \}$

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

Nondeterministically **guess** which character is missing.
Deterministically **check** whether that character is indeed missing.
Next time

Has nondeterminism made our finite automata more powerful? How do NFAs compare to DFAs?
Acknowledgments

This lecture incorporates material from:

Nancy Ide, Vassar College
Keith Schwarz, Stanford University
Michael Sipser, Introduction to the Theory of Computation