Nondeterministic
Finite Automata

4 February 2020
Assignment 1

Due today
Corrections due Thursday

Office hours
Where are we?
Formal language theory

An *alphabet*, denoted $\Sigma$, is a finite, non-empty set of symbols 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 $\Sigma$* if it 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.
A sample DFA

\[ L = \{ w \in \{0, 1\}^* \mid w \text{ contains } 11 \text{ as a substring} \} \]
Exercise

Build an automaton to recognize the set of strings that end with “ing”.
Exercise

Build an automaton to recognize the set of strings that start and end with the same symbol.
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 (or initial state).
- $F \subseteq Q$: Set of zero or more final states (or accepting states).
Formal DFAs revisited
Tabular DFAs
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>
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>→</td>
<td>q₀</td>
<td>q₀</td>
</tr>
<tr>
<td></td>
<td>q₁</td>
<td>q₀</td>
</tr>
<tr>
<td>*</td>
<td>q₂</td>
<td>q₂</td>
</tr>
</tbody>
</table>
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):
    state = "q0"
    for char in word:
        state = transition_table[state][char]
    return state in accept_states
```
Extended transition function
Intuitively, a finite automaton accepts a string \( w = a_1a_2\ldots a_n \) iff there is a path in the transition diagram that:

1. Begins at the start state
2. Ends at an accepting state
3. Has sequence of labels \( a_1, a_2, \ldots, a_n \).
Formally, we extend transition function $\delta$ to $\hat{\delta}(q, w)$ where $w$ can be any string of input symbols:

**Basis:** $\hat{\delta}(q, \varepsilon) = q$

On no input, the DFA doesn’t go anywhere

**Induction:** $\hat{\delta}(q, w\alpha) = \delta(\hat{\delta}(q, w), \alpha)$, where $w$ is a string and $\alpha$ is a single symbol. See where the DFA goes on $w$, then look for the transition on the last symbol from that state.
\( \hat{\delta} \) represents paths.

So, if

\[ w = a_1 a_2 \ldots a_n \]

and

\[ \delta(p_i, a_i) = p_{i+1} \]

for all \( i = 0, 1, \ldots, n-1 \), then

\[ \hat{\delta}(p_0, w) = p_n. \]
A finite automaton \( M = (Q, \Sigma, \delta, q_0, F) \) accepts string \( w \) if \( \hat{\delta}(q_0, w) \in F \).

A finite automaton \( M \) recognizes the language \( L(M) = \{ w \mid \hat{\delta}(q_0, w) \in F \} \).
Regular languages
Regular languages

**DEFINITION.** A language $L$ is called a *regular language* if there exists a DFA $D$ such that $L(D) = L$.

If $L$ is a language and $L(D) = L$, we say that $D$ *recognizes* the language $L$. 
The complement of a language

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

Formally:

$$\bar{L} = \Sigma^* - L$$
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:

\[
\overline{L} = \Sigma^* - L
\]
Complementing regular languages

$L = \{ w \in \{0, 1\}^* \mid w \text{ contains 11 as a substring} \}$
Complementing regular languages

\[ L = \{ w \in \{0, 1\}^* \mid w \text{ contains } 11 \text{ as a substring} \} \]

\[ \bar{L} = \{ w \in \{0, 1\}^* \mid w \text{ does not contain } 11 \text{ as a substring} \} \]
Complementing regular languages

\[ L = \{ w \in \{0, 1\}^* \mid w \text{ contains } 11 \text{ as a substring} \} \]

\[ \bar{L} = \{ w \in \{0, 1\}^* \mid w \text{ does not contain } 11 \text{ as a substring} \} \]
\[ L = \{ w \in \{a, *, /\}^* \mid w \text{ represents a C-style comment} \} \]
\[ \overline{L} = \{ w \in \{a, *, /\}^* \mid w \text{ doesn’t represent a C-style comment} \} \]
\[ \bar{L} = \{ w \in \{a, *, /\}^* \mid w \text{ doesn't represent a C-style comment} \} \]
Closure properties

THEOREM. If $L$ is a regular language, then $\bar{L}$ is also a regular language.

As a result, we say that the regular languages are closed under complementation.
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.

Are the nonregular languages closed under complementation? Why or why not?
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

start

$q_0$ 1 $q_1$ 1 $q_2$

$q_3$

0, 1

0, 1

0 1 0 1 1
A simple NFA
$\begin{align*}
&\text{start} \\
&\text{q}_0 \quad 1 \quad \text{q}_1 \\
&\quad \text{1} \\
&\quad \text{q}_2 \\
&\text{q}_3 \\
&\text{0, 1} \\
&\text{0} \\
&\text{0, 1} \\
&\text{0, 1} \\
&\text{0} \\
&\text{1} \\
\end{align*}$
Illustration by Gemma Correll
A more complex NFA

\begin{align*}
q_0 & \xrightarrow{\emptyset, 1} q_0 \\
q_0 & \xrightarrow{1} q_1 \\
q_1 & \xrightarrow{1} q_2
\end{align*}
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.
The diagram shows a finite automaton with states $q_0$, $q_1$, and $q_2$. The transitions are:

- From $q_0$, on input $0$ or $1$, the automaton moves to $q_0$.
- From $q_0$, on input $1$, the automaton moves to $q_1$.
- From $q_1$, on input $1$, the automaton moves to $q_2$.

The automaton starts at state $q_0$. The sequence of inputs and states is given by: $01011$. The automaton reaches the accepting state $q_2$.
0, 1

start

$q_0$ 1 $q_1$ 1 $q_2$

0 1 0 1 1
The automaton has three states: $q_0$, $q_1$, and $q_2$. The transitions are as follows:

- From $q_0$, on input 0 or 1, it stays at $q_0$.
- From $q_0$, on input 1, it moves to $q_1$.
- From $q_1$, on input 1, it moves to $q_2$.
- From $q_1$, on input 1, it stays at $q_1$.

The automaton starts at state $q_0$. It transitions to $q_1$ on input 1, and from $q_1$, it can stay at $q_1$ or move to $q_2$ on input 1. The input sequence is 0 1 0 1 1.
Nowhere to go!
Illustration by
Gemma Correll
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?

$L = \{w \in \Sigma^* | \text{ab is a suffix of } w\}$
Let $\Sigma = \{a, b\}$

What's the language of each NFA?
Let $\Sigma = \{a, b\}$

What’s the language of each NFA?

$L = \emptyset$
Let $\Sigma = \{a, b\}$

What's the language of each NFA?

$L = \emptyset$  \hspace{2cm}  $L = \{\varepsilon\}$
Let $\Sigma = \{a, b\}$

What’s the language of each NFA?

$L = \emptyset$  
$L = \{\varepsilon\}$  
$L = \Sigma^*$
Let $\Sigma = \{a, b\}$

What’s the language of each NFA?

$L = \emptyset$ 
$L = \{\varepsilon\}$ 
$L = \Sigma^*$
For DFAs, you must read a symbol in order for the machine to make a move.

NFAs can move without consuming an input symbol – an $\epsilon$-transition. An NFA can follow any number of $\epsilon$-transitions at any time without consuming any input.
Example

\[0 \quad 0 \quad 1\]

\[
\begin{array}{ccc}
\text{start} & \rightarrow & q \\
& & \\
1 & \rightarrow & r \\
& & \\
0 & \rightarrow & r \\
& & \\
\varepsilon & \rightarrow & r \\
& & \\
& & \rightarrow s \\
1 & \rightarrow & s \\
\end{array}
\]
Example

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

\[
\begin{array}{c}
\varepsilon \\
\varepsilon \\
\end{array}
\]

\[
\begin{array}{c}
start \\
r \\
s \\
q \\
\end{array}
\]

\[
\begin{array}{cc}
1 & \\
0 & \\
0 & 1 \\
1 & \\
\end{array}
\]

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

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

\begin{tikzpicture}

\node[state, initial] (q) at (0, 0) {$q$};
\node[state] (r) at (4, 0) {$r$};
\node[state, accepting] (s) at (8, 0) {$s$};

\draw[->] (q) edge node {$1$} (r);
\draw[->] (r) edge[loop right] node {$\varepsilon$} (r);
\draw[->] (q) edge[loop below] node {$\varepsilon$} (q);
\draw[->] (r) edge node {$\varepsilon$} (s);
\draw[->] (r) edge node {$1$} (s);
\draw[->] (q) edge node {$0$} (r);
\end{tikzpicture}
Example

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

- **State Diagram**
  - **Start State**: q
  - Edges:
    - \( q \rightarrow r \) on input 1
    - \( q \rightarrow q \) on input 0
    - \( r \rightarrow s \) on input \( \varepsilon \)
    - \( r \rightarrow r \) on input \( \varepsilon \)
    - \( s \rightarrow s \) on input \( \varepsilon \)


Example
Example

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

\[
\begin{array}{ccc}
q & r & s \\
0 & 1 & \varepsilon \\
\varepsilon & \varepsilon & 1
\end{array}
\]
NFAs are not required to follow \( \varepsilon \)-transitions; they’re just another choice of path for the computation.
Acknowledgments

This lecture incorporates material from:

Nancy Ide
Keith Schwarz
Michael Sipser