CS 154

Finite Automata,
Nondeterminism,
Regular Expressions
The DFA **accepts** a string if the process ends in a double circle.

Read string left to right

```
0111
```
A DFA is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$

$Q$ is the set of states (finite)

$\Sigma$ is the alphabet (finite)

$\delta : Q \times \Sigma \rightarrow Q$ is the transition function

$q_0 \in Q$ is the start state

$F \subseteq Q$ is the set of accept/final states

$L(M) = \text{set of all strings that } M \text{ accepts} = \text{“the language recognized by } M\text{”}$
A DFA is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$

$L(M) =$ set of all strings that $M$ accepts
$= “\text{the language recognized by } M”$

**Definition:** A language $L$ is **regular** if it is recognized by a DFA; that is, there is a DFA $M$ where $L = L(M)$.
Union Theorem for Regular Languages

The union of two regular languages is also a regular language

Intersection Theorem for Regular Languages

The intersection of two regular languages is also a regular language
Complement Theorem for Regular Languages

The complement of a regular language is also a regular language
The Reverse of a Language

Reverse of L:
\[ L^R = \{ w_1 \ldots w_k \mid w_k \ldots w_1 \in L, w_i \in \Sigma \} \]

If L is recognized by the usual kind of DFA, then \( L^R \) is recognized by a DFA that reads its strings from right to left!

Question: If L is regular, then is \( L^R \) also regular?

Can every “Right-to-Left” DFA be replaced by a normal “Left-to-Right” DFA?
Suppose our machine reads strings from right to left...
Then \( L(M) = \{w \mid w \text{ ends with a } 1\} \). Is this regular?
Reversing DFAs

Assume $L$ is a regular language. Let $M$ be a DFA that recognizes $L$.

We’ll build a machine $M^R$ that accepts $L^R$.

If $M$ accepts $w$, then $w$ describes a directed path in $M$ from start to an accept.

**First Attempt:** Try to define $M^R$ as $M$ with the arrows reversed, turn start state into a final state, turn final states into starts.
Problem: $M^R$ IS NOT ALWAYS A DFA!

It could have many start states

Some states may have *more than one* outgoing edge, or none at all!
Non-deterministic Finite Automata (NFA)

What happens with 100?

We will say this new machine accepts a string $x$ if there is some path reading in $x$ that reaches some accept state from some start state.
Non-deterministic Finite Automata (NFA)

Then, this machine recognizes: \( \{ w \mid w \text{ contains 100} \} \)

We will say this new machine \textit{accepts} a string \( x \) if \textit{there is some path reading in} \( x \) \textit{that reaches some accept state from some start state.}
At each state, we can have any number of out arrows for a letter $\sigma \in \Sigma$, including $\varepsilon$.

Set of strings accepted by this NFA $= \{w \mid w \text{ contains a } 0\}$
Multiple Start States

We allow *multiple* start states for NFAs, and Sipser allows only one.

Can easily convert NFA with many start states into one with a single start state:
A *non-deterministic* finite automaton (NFA) is a 5-tuple \( N = (Q, \Sigma, \delta, Q_0, F) \) where

- \( Q \) is the set of states
- \( \Sigma \) is the alphabet
- \( \delta : Q \times \Sigma_\varepsilon \rightarrow 2^Q \) is the transition function
- \( Q_0 \subseteq Q \) is the set of start states
- \( F \subseteq Q \) is the set of accept states

\( 2^Q \) is the set of all possible subsets of \( Q \)

\( \Sigma_\varepsilon = \Sigma \cup \{\varepsilon\} \)
**Def.** Let $w \in \Sigma^*$. Let $N$ be an NFA. $N$ accepts $w$ if there’s a sequence of states $r_0, r_1, \ldots, r_k \in Q$ and $w$ can be written as $w_1 \ldots w_k$ with $w_i \in \Sigma \cup \{\varepsilon\}$ such that

1. $r_0 \in Q_0$
2. $r_{i+1} \in \delta(r_i, w_{i+1})$ for all $i = 0, \ldots, k-1$, and
3. $r_n \in F$

$L(N) = \text{the language recognized by } N$

$= \text{set of all strings machine } N \text{ accepts}$

A language $L'$ is **recognized** by an NFA $N$ if $L' = L(N)$. 
\[ L(N) = \{1,00,01\} \]

\[ N = (Q, \Sigma, \delta, Q_0, F) \]

\[ Q = \{q_1, q_2, q_3, q_4\} \]

\[ \Sigma = \{0,1\} \]

\[ Q_0 = \{q_1, q_2\} \]

\[ F = \{q_4\} \]

\[ \delta(q_2,1) = \{q_4\} \]

\[ \delta(q_3,1) = \emptyset \]

\[ \delta(q_1,0) = \{q_3\} \]
Are these equally powerful???
NFAs are generally simpler than DFAs

A DFA recognizing the language $\{1\}$

An NFA recognizing the language $\{1\}$
Every NFA can be perfectly simulated by some DFA!

**Theorem:** For every NFA N, there is a DFA M such that $L(M) = L(N)$

**Corollary:** A language $L$ is regular if and only if $L$ is recognized by an NFA

**Corollary:** $L$ is regular iff $L^R$ is regular
From NFAs to DFAs

Input: NFA $N = (Q, \Sigma, \delta, Q_0, F)$

Output: DFA $M = (Q', \Sigma, \delta', q_0', F')$

To learn if an NFA accepts, we could do the computation in parallel, maintaining the set of all possible states that can be reached.

Idea:
Set $Q' = 2^Q$
From NFAs to DFAs: Subset Construction

Input: NFA $N = (Q, \Sigma, \delta, Q_0, F)$

Output: DFA $M = (Q', \Sigma, \delta', q_0', F')$

$$Q' = 2^Q$$

$$\delta' : Q' \times \Sigma \rightarrow Q'$$

$$\delta'(R, \sigma) = \bigcup_{r \in R} \varepsilon(\delta(r, \sigma))$$

$$q_0' = \varepsilon(Q_0)$$

$$F' = \{ R \in Q' \mid f \in R \text{ for some } f \in F \}$$

For $S \subseteq Q$, the $\varepsilon$-closure of $S$ is

$$\varepsilon(S) = \{ q \in Q \text{ reachable from some } s \in S \text{ by taking 0 or more } \varepsilon \text{ transitions} \}$$
Example of the \( \varepsilon \)-closure

\[ \varepsilon(\{q_0\}) = \{q_0, q_1, q_2\} \]
\[ \varepsilon(\{q_1\}) = \{q_1, q_2\} \]
\[ \varepsilon(\{q_2\}) = \{q_2\} \]
Given: NFA $N = (\{1,2,3\}, \{a,b\}, \delta, \{1\}, \{1\})$

Construct: Equivalent DFA $M$

$M = (2^{\{1,2,3\}}, \{a,b\}, \delta', \{1,3\}, ...)$

$\varepsilon(\{1\}) = \{1,3\}$

$\{1\}, \{1,2\} \ ?$
Reverse Theorem for Regular Languages

The reverse of a regular language is also a regular language

If a language can be recognized by a DFA that reads strings from **right to left**, then there is an “normal” DFA that accepts the same language

Proof?

Given a DFA for a language L, “reverse” its arrows and flip its start and accept states, getting an NFA. Convert that NFA back to a DFA.
Using NFAs in place of DFAs can make proofs about regular languages much easier!

Remember this on homework/exams!
Union Theorem using NFAs?
Regular Languages are closed under concatenation

**Concatenation:** $A \cdot B = \{ vw \mid v \in A \text{ and } w \in B \}$

Given DFAs $M_1$ and $M_2$, connect the accept states of $M_1$ to the start states of $M_2$

$L(N) = L(M_1) \cdot L(M_2)$
Regular Languages are closed under star

$$A^* = \{ s_1 \ldots s_k \mid k \geq 0 \text{ and each } s_i \in A \}$$

Let $M$ be a DFA, and let $L = L(M)$

We can construct an NFA $N$ that recognizes $L^*$
Formally, the construction is:

**Input:** DFA $M = (Q, \Sigma, \delta, q_1, F)$

**Output:** NFA $N = (Q', \Sigma, \delta', \{q_0\}, F')$

- $Q' = Q \cup \{q_0\}$
- $F' = F \cup \{q_0\}$

$$
\delta'(q, a) = \begin{cases} 
\{\delta(q, a)\} & \text{if } q \in Q \text{ and } a \neq \varepsilon \\
\{q_1\} & \text{if } q \in F \text{ and } a = \varepsilon \\
\emptyset & \text{if } q = q_0 \text{ and } a = \varepsilon \\
\emptyset & \text{if } q = q_0 \text{ and } a \neq \varepsilon \\
\emptyset & \text{else}
\end{cases}
$$
Regular Languages are closed under star

How would we prove that this NFA construction works?

Want to show: \( L(N) = L^* \)

1. \( L(N) \supseteq L^* \)

2. \( L(N) \subseteq L^* \)
1. $L(N) \supseteq L^*$

Assume $w = w_1 \ldots w_k$ is in $L^*$ where $w_1, \ldots, w_k \in L$

We show $N$ accepts $w$ by induction on $k$

Base Cases:
- $k = 0$  
  $(w = \varepsilon)$
- $k = 1$  
  $(w \in L)$

Inductive Step:

Assume $N$ accepts all strings $v = v_1 \ldots v_k \in L^*$, $v_i \in L$

Let $u = u_1 \ldots u_k u_{k+1} \in L^*$, $u_j \in L$

Since $N$ accepts $u_1 \ldots u_k$ (by induction) and $M$ accepts $u_{k+1}$, $N$ also accepts $u$ (by construction)
2. \( L(N) \subseteq L^* \)

Assume \( w \) is accepted by \( N \); we want to show \( w \in L^* \)

If \( w = \varepsilon \), then \( w \in L^* \)

I.H. \( N \) accepts \( u \) and takes at most \( k \) \( \varepsilon \)-transitions \( \Rightarrow u \in L^* \)

Let \( w \) be accepted by \( N \) with \( k+1 \).

Write \( w \) as \( w = uv \), where \( v \) is the substring read after the last \( \varepsilon \)-transition

\( u \in L(N) \), so \( u \in L^* \)

By I.H.

\( w = uv \in L^* \)

\( v \in L \)
Regular Languages are closed under all of the following operations:

→ **Union:** $A \cup B = \{ w \mid w \in A \text{ or } w \in B \}$

→ **Intersection:** $A \cap B = \{ w \mid w \in A \text{ and } w \in B \}$

→ **Complement:** $\neg A = \{ w \in \Sigma^* \mid w \notin A \}$

→ **Reverse:** $A^R = \{ w_1 \ldots w_k \mid w_k \ldots w_1 \in A \}$

→ **Concatenation:** $A \cdot B = \{ vw \mid v \in A \text{ and } w \in B \}$

→ **Star:** $A^* = \{ w_1 \ldots w_k \mid k \geq 0 \text{ and each } w_i \in A \}$
Homework 1 is coming out today... watch for it!
Regular Expressions
Inductive Definition of Regexp

Let $\Sigma$ be an alphabet. We define the regular expressions over $\Sigma$ inductively:

For all $\sigma \in \Sigma$, $\sigma$ is a regexp

$\varepsilon$ is a regexp

$\emptyset$ is a regexp

If $R_1$ and $R_2$ are both regexps, then

$(R_1 R_2)$, $(R_1 + R_2)$, and $(R_1)^*$ are regexps
**Precedence Order:**

Example:

\[ R_1 \times R_2 + R_3 = ((R_1 \times R_2) \cdot R_3) + R_3 \]
Definition: Regexps Represent Languages

The regexp $\sigma \in \Sigma$ represents the language $\{\sigma\}$

The regexp $\varepsilon$ represents $\{\varepsilon\}$

The regexp $\emptyset$ represents $\emptyset$

If $R_1$ and $R_2$ are regular expressions representing $L_1$ and $L_2$ then:

$(R_1R_2)$ represents $L_1 \cdot L_2$

$(R_1 + R_2)$ represents $L_1 \cup L_2$

$(R_1)^*$ represents $L_1^*$
Regexps Represent Languages

For every regexp $R$, define $L(R)$ to be the language that $R$ represents.

A string $w \in \Sigma^*$ is accepted by $R$ (or, $w$ matches $R$) if $w \in L(R)$.

Example: $01010$ matches the regexp $(01)^*0$.
end