Lecture 3: Nondeterminism and Regular Expressions
Announcements:

- Pset 0 is out, due tomorrow 11:59pm
- Latex source of hw on piazza
- Pset 1 coming out tomorrow
- No class next Tuesday (...because next week Monday classes will be on Tuesday)
Deterministic Finite Automata

Computation with finite memory
Non-Deterministic Finite Automata

Computation with finite memory and magical guessing
Non-deterministic Finite Automata (NFA)

This NFA recognizes: \( \{ w \mid w \text{ contains } 100 \} \)

An NFA accepts string \( x \) if there is some path reading in \( x \) that reaches some accept state from some start state.
Theorem: For every NFA $N$, there is a DFA $M$ such that $L(M) = L(N)$

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

Corollary: $A$ is regular iff $A^R$ is regular

left-to-right DFAs $\equiv$ right-to-left DFAs
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 NFA $N$ accepts, we could do the computation of $N$ 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 \( M = (Q', \Sigma, \delta', q_0', F') \)

\[
Q' = 2^Q
\]

\[
\delta' : Q' \times \Sigma \rightarrow Q'
\]

For \( S \in Q', \sigma \in \Sigma: \quad \delta'(S,\sigma) = \bigcup_{q \in S} \varepsilon(\delta(q,\sigma))
\]

\[
q_0' = \varepsilon(Q_0)
\]

\[
F' = \{ S \in Q' \mid f \in S \text{ for some } f \in F \}
\]

* For \( S \subseteq Q \), the \( \varepsilon \)-closure of \( S \) is

\[
\varepsilon(S) = \{ r \in Q \text{ reachable from some } q \in S \text{ by taking zero or more } \varepsilon\text{-transitions} \} 
\]
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 Sketch?

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?
Some Operations on Languages

- **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:** \( \overline{A} = \{ w \in \Sigma^* \mid w \notin A \} \)
- **Reverse:** \( A^R = \{ w_1 \ldots w_k \mid w_k \ldots w_1 \in A, w_i \in \Sigma \} \)
- **Concatenation:** \( A \cdot B = \{ vw \mid v \in A \text{ and } w \in B \} \)
- **Star:** \( A^* = \{ s_1 \ldots s_k \mid k \geq 0 \text{ and each } s_i \in A \} \)

\( A^* = \text{set of all strings over alphabet } A \)
Regular Languages are closed under concatenation

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

Given DFAs \( M_1 \) for \( A \) and \( M_2 \) for \( B \), connect the accept states of \( M_1 \) to the start state state of \( M_2 \)
Regular Languages are closed under concatenation

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

Given DFAs \( M_1 \) for \( A \) and \( M_2 \) for \( B \), connect the accept states of \( M_1 \) to the start state 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

We construct an NFA \( N \) that recognizes \( L(M)^* \)
Formally, the construction is:

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

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

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

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

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

How would we *prove* that the NFA construction works?

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

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

Let \( w = w_1 \cdots w_k \) be in \( L(M)^* \) where \( w_1, \ldots, w_k \in L(M) \)

We show: \( N \) accepts \( w \) by induction on \( k \)

Base Cases:

- \( k = 0 \) \hspace{2cm} \( (w = \varepsilon) \)
- \( k = 1 \) \hspace{2cm} \( (w \in L(M) \text{ and } L(M) \subseteq L(N)) \)

Inductive Step: Let \( k \geq 1 \) be an integer

I.H. \( N \) accepts all strings \( v = v_1 \cdots v_k \in L(M)^* \), \( v_i \in L(M) \)

Let \( u = u_1 \cdots u_k u_{k+1} \in L(M)^* \), \( u_j \in L(M) \)

\( N \) accepts \( u_1 \cdots u_k \) (by I.H.) and \( M \) accepts \( u_{k+1} \)

imply that \( N \) also accepts \( u \)

(since \( N \) has \( \varepsilon \)-transitions from final states to start state of \( M \!))
Let $w$ be accepted by $N$; we want to show $w \in L(M)^*$.

If $w = \epsilon$, then $w \in L(M)^*$.

I.H. $N$ accepts $u$ and takes $k$ \(\epsilon\)-transitions, so $u \in L(M)^*$.

Let $w$ be accepted by $N$ with $k+1$ \(\epsilon\)-transitions.

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

N accepts $u$, so $u \in L(M)^*$, By I.H.

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

$v \in L(M)$
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:** \( \overline{A} = \{ w \in \Sigma^* \mid w \notin A \} \)

**Reverse:** \( A^R = \{ w_1 \ldots w_k \mid w_k \ldots w_1 \in A, w_i \in \Sigma \} \)

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

**Star:** \( A^* = \{ s_1 \ldots s_k \mid k \geq 0 \text{ and each } s_i \in A \} \)
Regular Expressions: Computation as Description

A different way of thinking about computation:

What is the complexity of describing the strings in the language?
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

Examples: $\varepsilon$, 0, $(1)^*$, $(0+1)^*$, $((((0)^1)^1) + (10))$
Precedence Order: $*$

then $\cdot$

then $+$

Example: $R_1 \ast R_2 + R_3 = ( ( R_1 \ast ) \cdot R_2 ) + R_3$
Definition: Regexps Describe Languages

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

The regexp $\epsilon$ represents $\{\epsilon\}$

The regexp $\emptyset$ represents $\emptyset$

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

$(R_1 R_2)$ represents $L_1 \cdot L_2$

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

$(R_1)^*$ represents $L_1^*$

Example: $(10 + 0^*1)$ represents $\{10\} \cup \{0^k1 \mid k \geq 0\}$
Regexp's Describe 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 $w \in L(R)$.

Examples: 0, 010, and 01010 match $(01)^*0$

110101110101100 matches $(0+1)^*0$
Assume $\Sigma = \{0,1\}$

\[
\{ w \mid w \text{ has exactly a single 1} \}
\]

0*10*

\[
\{ w \mid w \text{ contains 001} \}
\]

$(0+1)^*001(0+1)^*$
What language does the regexp $\emptyset^*$ represent?

$\{\varepsilon\}$
Assume $\Sigma = \{0,1\}$

$\{ w \mid w \text{ has length } \geq 3 \text{ and its 3rd symbol is } 0 \}$

$(0+1)(0+1)0(0+1)^*$
Assume \( \Sigma = \{0,1\} \)

\[
\{ w \mid w = \varepsilon \text{ or every odd position in } w \text{ is a } 1 \}
= (1(0 + 1))^{*}(1 + \varepsilon)
\]

How expressive are regular expressions?
During the “nerve net” hype in the 1950s...
DFAs $\equiv$ NFAs $\equiv$ Regular Expressions!

$L$ can be represented by some regexp
$\iff$ $L$ is regular
L can be represented by some regexp

⇒ L is regular
Induction Step: Suppose every regexp of length $< k$ represents some regular language.

Consider a regexp $R$ of length $k > 1$

Three possibilities for $R$:

$$R = R_1 + R_2$$

$$R = R_1 R_2$$

$$R = (R_1)^*$$
Induction Step: Suppose every regexp of length $< k$ represents some regular language.

Consider a regexp $R$ of length $k > 1$

Three possibilities for $R$:

- $R = R_1 + R_2$ By induction, $R_1$ and $R_2$ represent some regular languages, $L_1$ and $L_2$
- $R = R_1 R_2$ But $L(R) = L(R_1 + R_2) = L_1 \cup L_2$
- $R = (R_1)^*$ so $L(R)$ is regular, by the union theorem!
**Induction Step:** Suppose every regexp of length $< k$ represents some regular language.

Consider a regexp $R$ of length $k > 1$

Three possibilities for $R$:

- $R = R_1 + R_2$  
  By induction, $R_1$ and $R_2$ represent some regular languages, $L_1$ and $L_2$

- $R = R_1 R_2$  
  But $L(R) = L(R_1 \cdot R_2) = L_1 \cdot L_2$

- $R = (R_1)^*$  
  Thus $L(R)$ is regular because regular languages are closed under concatenation
Induction Step: Suppose every regexp of length $< k$ represents some regular language.

Consider a regexp $R$ of length $k > 1$

Three possibilities for $R$:

1. $R = R_1 + R_2$
   - By induction, $R_1$ represents a regular language $L_1$
   - Thus $L(R) = L(R_1^*) = L_1^*$

2. $R = R_1 R_2$
   - But $L(R) = L(R_1^*) = L_1^*$

3. $R = (R_1)^*$
   - Thus $L(R)$ is regular because regular languages are closed under star
**Induction Step:** Suppose every regexp of length \( < k \) represents some regular language.

Consider a regexp \( R \) of length \( k > 1 \)

Three possibilities for \( R \):

- \( R = R_1 + R_2 \)  
  By induction, \( R_1 \) represents a regular language \( L_1 \)

- \( R = R_1 R_2 \)  
  But \( L(R) = L(R_1^*) = L_1^* \)

- \( R = (R_1)^* \)  
  Thus \( L(R) \) is regular because regular languages are closed under star

Therefore: If \( L \) is represented by a regexp, then \( L \) is regular!
Give an NFA that accepts the language represented by \((1(0 + 1))^*\)

Regular expression: \((1(0+1))^*\)
Generalized NFAs (GNFA)

L can be represented by a regexp

L is a regular language

Idea: Transform a DFA for L into a regular expression by *removing states* and re-labeling the arcs with *regular expressions*

Rather than reading in just 0 or 1 letters from the string on an arc, we can read in *entire substrings*
Generalized NFA (GNFA)

This GNFA recognizes $L(a^*b(cb)^*a)$

Accept string $x \iff$ there is some path of regexps $R_1, \ldots, R_k$ from start state to final state such that $x$ matches $R_1 \cdots R_k$

Is $aaabcbcba$ accepted or rejected?
Is $bba$ accepted or rejected?
Is $bcba$ accepted or rejected?
Generalized NFA (GNFA)

This GNFA recognizes $L(a^*b(cb)^*a)$

Accept string $x \iff$ there is some path of regexps $R_1, \ldots, R_k$ from start state to final state such that $x$ matches $R_1 \cdots R_k$

Every NFA is also a GNFA.
Every regexp can be converted into a GNFA with just two states!
Add unique start and accept states

Goal: Replace DFA with a single regexp $R$

Then, $L(R) = L(DFA)$
While the machine has more than 2 states:

Pick an internal state, **rip it out and re-label the arrows with regexps**, to account for paths through the missing state.
While the machine has more than 2 states:

Pick an internal state, rip it out and re-label the arrows with regexps, to account for paths through the missing state.

\[
\text{ab}^*c
\]
In general:

While the machine has more than 2 states:

\[ R(q_1, q_2) \rightarrow R(q_2, q_3) \rightarrow R(q_1, q_3) \]
While the machine has more than 2 states:

In general:

\[ R(q_1, q_2)R(q_2, q_2)^*R(q_2, q_3) + R(q_1, q_3) \]
$R(q_0, q_3) = (a*b)(a+b)^*$
represents $L(N)$
\[ R(q_0, q_3) = (a \cdot b)(a+b)^* \]

represents \( L(N) \)
\( R(q_0, q_3) = (a*b)(a+b)^* \)

represents \( L(N) \)
Formally: Given a DFA $M$, add $q_{\text{start}}$ and $q_{\text{acc}}$ to create $G$

For all $q, q' \in Q$, define $R(q, q') = \sigma_1 + \cdots + \sigma_k$ s.t. $\delta(q, \sigma_i) = q'$

**CONVERT(G):** *(Takes a GNFA, outputs a regexp)*

If #states = 2  \[\text{return } R(q_{\text{start}}, q_{\text{acc}})\]

If #states > 2

\[\text{pick } q_{\text{rip}} \in Q \text{ different from } q_{\text{start}} \text{ and } q_{\text{acc}}\]

\[\text{define } Q' = Q - \{q_{\text{rip}}\}\]

\[\text{define } R' \text{ on } Q' - \{q_{\text{acc}}\} \times Q' - \{q_{\text{start}}\} \text{ as:}\]

\[R'(q_i, q_j) = R(q_i, q_{\text{rip}})R(q_{\text{rip}}, q_{\text{rip}})^*R(q_{\text{rip}}, q_j) + R(q_i, q_j)\]

\[\text{return } \text{CONVERT}(G')\]

**Theorem:** Let $R = \text{CONVERT}(G)$.
Then $L(R) = L(M)$.

**Claim:** $L(G') = L(G)$ *[Sipser, p.73-74]*