Lecture 6:
The Myhill-Nerode Theorem and Streaming Algorithms
DFA Minimization Theorem:

For every regular language $A$, there is a unique (up to re-labeling of the states) minimal-state DFA $M^*$ such that $A = L(M^*)$. Furthermore, there is an efficient algorithm which, given any DFA $M$, will output this unique $M^*$.

If such algorithms existed for more general models of computation, that would be an engineering breakthrough!!
How could we show whether two regular expressions are equivalent?

**Claim:** There is an algorithm which given regular expressions $R$ and $R'$, determines whether $L(R) = L(R')$. 
The Myhill-Nerode Theorem:

For every language L:

Either there’s a DFA for L

or there’s a set of strings that “trick” every possible DFA trying to recognize L
In DFA Minimization, we defined an equivalence relation between states of a DFA. We can also define a similar equivalence relation over *strings* in a *language*:

Let \( L \subseteq \Sigma^* \) and \( x, y \in \Sigma^* \)

\[ x \equiv_L y \text{ means: for all } z \in \Sigma^*, \ xz \in L \iff yz \in L \]

**Def.** \( x \) and \( y \) are **indistinguishable to \( L \)** iff \( x \equiv_L y \)

**Claim:** \( \equiv_L \) (“\( L \)-equivalent”) is an equivalence relation
Let $L \subseteq \Sigma^*$ and $x, y \in \Sigma^*$. 

$x \equiv_l y$ means: for all $z \in \Sigma^*$, $xz \in L \iff yz \in L$ 

**Def.** $x$ and $y$ are indistinguishable to $L$ iff $x \equiv_l y$ 

**Claim:** $\equiv_l$ ("L-equivalent") is an equivalence relation.

**Reflexive:**

$x \equiv_l x$: for all $z \in \Sigma^*$, $xz \in L \iff xz \in L$ 

**Symmetric:**

$x \equiv_l y$: for all $z \in \Sigma^*$, $xz \in L \iff yz \in L$ 

Equivalent to: for all $z \in \Sigma^*$, $yz \in L \iff xz \in L$, $y \equiv_l x$ 

**Transitive:**

$x \equiv_l y$: for all $z \in \Sigma^*$, $xz \in L \iff yz \in L$ 

$y \equiv_l w$: for all $z \in \Sigma^*$, $yz \in L \iff wz \in L$ 

Implies for all $z \in \Sigma^*$, $xz \in L \iff wz \in L$, $x \equiv_l x$
Let $L \subseteq \Sigma^*$ and $x, y \in \Sigma^*$

$x \equiv_L y$ means: for all $z \in \Sigma^*$, $xz \in L \iff yz \in L$

Suppose we partition all strings in $\Sigma^*$ into equivalence classes under $\equiv_L$

The Myhill-Nerode Theorem:

If the number of parts is finite $\implies$ can construct a DFA!

If the number of parts is infinite $\implies$ there is no DFA!
Mapping strings to DFA states

Given DFA $M = (Q, \Sigma, \delta, q_0, F)$, we define a function $\Delta : \Sigma^* \to Q$ as follows:

$\Delta(\varepsilon) = q_0$

$\Delta(\sigma) = \delta(q_0, \sigma)$

$\Delta(\sigma_1 \cdots \sigma_k \sigma_{k+1}) = \delta(\Delta(\sigma_1 \cdots \sigma_k), \sigma_{k+1})$

$\Delta(w) = \text{the state of } M \text{ reached after reading in } w$

Note: $\Delta(w) \in F \iff M \text{ accepts } w$
The Myhill-Nerode Theorem:
A language $L$ is regular if and only if the number of equivalence classes of $\equiv_L$ is finite.

Proof ($\Rightarrow$) Let $M = (Q, \Sigma, \delta, q_0, F)$ be any DFA for $L$.

Define the relation: $x \approx_M y \iff \Delta(x) = \Delta(y)$

Claim: $\approx_M$ is an equivalence relation with $|Q|$ classes

Claim: If $x \approx_M y$ then $x \equiv_L y$

Proof: $x \approx_M y$ implies for all $z \in \Sigma^*$, $xz$ and $yz$ reach the same state of $M$. So $xz \in L \iff yz \in L$, and $x \equiv_L y$

Corollary: The number of $\equiv_L$ classes is at most the number of $\approx_M$ classes (which is $|Q|$)
The Myhill-Nerode Theorem:
A language $L$ is regular if and only if the number of equivalence classes of $\equiv_L$ is finite.

Claim: If $x \approx_M y$ then $x \equiv_L y$

Corollary: The number of $\equiv_L$ classes is at most the number of $\approx_M$ classes (which is $|Q|$)

Proof: Let $S = \{x_1, x_2, \ldots \}$ be distinct strings, one from every $\equiv_L$ class. $|S| = \text{number of } \equiv_L \text{ classes.}$

Thus for all $i \neq j, x_i \not\equiv_L x_j.$ By the claim: $x_i \approx_M x_j.$

So each $x_i \in S$ is in a distinct $\approx_M$ equivalence class. $\Rightarrow$ The number of $\approx_M$ classes is at least $|S|.$
Let $L \subseteq \Sigma^*$ and $x, y \in \Sigma^*$

$x \equiv_L y$ means: for all $z \in \Sigma^*$, $xz \in L \iff yz \in L$

$(\Leftarrow)$ If the number of equivalence classes of $\equiv_L$ is $k$
then there is a DFA for $L$ with $k$ states

**Idea:** Build a DFA whose *states* are
the *equivalence classes* of $\equiv_L$

Define a DFA $M$ where:

- $Q$ is the set of equivalence classes of $\equiv_L$
- $q_0 = [\varepsilon] = \{ y \mid y \equiv_L \varepsilon \}$
- for all $x \in \Sigma^*$, $\delta([x], \sigma) = [x \sigma]$ (well-defined??)
- $F = \{ [x] \mid x \in L \}$

**Claim:** $M$ accepts $x$ if and only if $x \in L$
Define a DFA $M$ where:

- $Q$ is the set of equivalence classes of $\equiv_L$
- $q_0 = [\varepsilon] = \{y \mid y \equiv_L \varepsilon\}$
- $\delta([x], \sigma) = [x \sigma]$
- $F = \{[x] \mid x \in L\}$

Claim: $M$ accepts $x$ if and only if $x \in L$

Proof: Let $M$ run on $x = x_1 \cdots x_n \in \Sigma^*$, for $x_i \in \Sigma$. $M$ starts in state $[\varepsilon]$, reads $x_1$ and moves to $[x_1]$, reads $x_2$ and moves to $[x_1 x_2]$, ..., and ends in state $[x_1 \cdots x_n]$. So, $M$ accepts $x_1 \cdots x_n \iff [x_1 \cdots x_n] \in F$

By definition of the set $F$, $[x_1 \cdots x_n] \in F \iff x \in L$
The Myhill-Nerode Theorem gives us a new way to prove that a given language is not regular:

$L$ is not regular

if and only if

there are infinitely many equiv. classes of $\equiv_L$

$L$ is not regular

if and only if

There are infinitely many strings $w_1, w_2, \ldots$ so that for all $w_i \neq w_j$, $w_i$ and $w_j$ are distinguishable to $L$:

there is a $z \in \Sigma^*$ such that

'exactly one' of $w_i z$ and $w_j z$ is in $L$
L is not regular \textit{if and only if}
There are infinitely many strings $w_1, w_2, \ldots$ so that for all $w_i \neq w_j$, $w_i$ and $w_j$ are distinguishable to L

To prove that L is regular, we have to show that a special finite object (DFA/NFA/regex) exists.

To prove that L is not regular, it is sufficient to show that a special infinite set of strings exists!

We can prove the nonexistence of a DFA/NFA/regex by proving the existence of this special string set!
Using **Myhill-Nerode** to prove non-regularity:

**Theorem:** \( L = \{0^n 1^n \mid n \geq 0\} \) is not regular.

**Proof:** Consider the infinite set of strings 
\[ S = \{0, 00, 000, \ldots, 0^n, \ldots\} \]

Claim: \( S \) is a distinguishing set for \( L \).

Take any pair \((0^m, 0^n)\) of distinct strings in \( S \)
Let \( z = 1^m \)
Then \( 0^m 1^m \) is in \( L \), but \( 0^n 1^m \) is *not* in \( L \)
So all pairs of strings in \( S \) are distinguishable to \( L \)

Hence there are infinitely many equivalence classes of \( \equiv_L \), and \( L \) is not regular!
Theorem: \( \text{PAL} = \{ x \, x^R \mid x \in \{0, 1\}^* \} \) is not regular.

Proof: Consider the infinite set of strings

\[ S = \{01^k0 \mid k \geq 1\} \]

Claim: \( S \) is a distinguishing set for \( L \).

Take any pair \((01^k0, 01^j0)\) of strings where \( j \neq k \)

Let \( z = 1^k0 \)

Then \( 01^k0 \, 1^k0 \) is in \( \text{PAL} \), but \( 01^j0 \, 1^k0 \) is \( \text{not} \) in \( \text{PAL} \)

So all pairs of strings in \( S \) are distinguishable to \( \text{PAL} \)

Hence there are infinitely many equivalence classes of \( \equiv_L \), and \( L \) is not regular
(by the Myhill-Nerode theorem)
Streaming Algorithms
Streaming Algorithms
Streaming Algorithms

Have three components

Initialize:

<variables and their assignments>

When next symbol seen is $\sigma$:

<pseudocode using $\sigma$ and vars>

When stream stops (end of string):

<accept/reject condition on vars>

(or: <pseudocode for output>)

Algorithm A computes $L \subseteq \Sigma^*$ if

A accepts the strings in $L$, rejects strings not in $L$
Streaming Algorithms

Streaming algorithms differ from DFAs in several significant ways:

1. Streaming algorithms could output more than one bit
2. The “memory” or “space” of a streaming algorithm can (slowly) increase as it reads longer strings
3. Could also make multiple passes over the input, could be randomized

Can recognize non-regular languages!
L = \{x \mid x \text{ has more 1's than 0's}\}

Initialize: C := 0 and B := 0

When next symbol seen is \(\sigma\):
- If (C = 0) then B := \(\sigma\), C := 1
- If (C \neq 0) and (B = \(\sigma\)) then C := C + 1
- If (C \neq 0) and (B \neq \(\sigma\)) then C := C - 1

When stream stops:
- \textit{accept} if B=1 and C > 0, else \textit{reject}

B = the majority bit
C = how many more times B appears

On all strings of length \(n\), the algorithm uses \((\log_2 n) + O(1)\) bits of space (to store B and C)
How to think of memory usage

The program is not considered as part of the memory

Space usage of A:

\[ S(n) = \text{maximum # of bits used to store vars in A, over all inputs of length up to } n \]
$L = \{0^n1^n \mid n \geq 0\}$

Initialize: $z := 0$, $s := \text{false}$, $\text{fail} := \text{false}$

When next symbol seen is $\sigma$:
- If (not $s$) and ($\sigma = 0$) then $z := z + 1$
- If (not $s$) and ($\sigma = 1$) then $s := \text{true}$; $z := z - 1$
- If ($s$) and ($\sigma = 0$) then $\text{fail} := \text{true}$
- If ($s$) and ($\sigma = 1$) then $z := z - 1$

When stream stops:
- $\text{accept}$ if and only if (not $\text{fail}$) and ($z = 0$)

$z$ = how many more times $0$ appears than $1$

$s$ = “Started reading $1$s yet?”

$\text{fail}$ = “Reject for certain?”

On all strings of length $n$, uses $(\log_2 n) + O(1)$ space
DFAs and Streaming

**Thm:** Let $L'$ be recognized by DFA $M$ with $\leq 2^p$ states. Then $L'$ is computable by a streaming algorithm $A$ using $\leq p$ bits of space.

**Proof Idea:** Define algorithm $A$ as follows.

- **Initialize:** Encode the start state of $M$ in memory.
- **When next symbol seen is $\sigma$:**
  - Update state of $M$ using $M$'s transition function
- **When stream stops:**
  - **Accept** if current state of $M$ is final, else **reject**
DFAs and Streaming

Thm: Let $L'$ be recognized by DFA $M$ with $\leq 2^p$ states. Then $L'$ is computable by a streaming algorithm $A$ using $\leq p$ bits of space.

Initialize: $B = 0$
When reading $\sigma$: Set $B := \sigma$
When stream stops: Accept iff $B = 1$
Uses 1 bit of space
DFAs and Streaming

For any $A \subseteq \Sigma^*$ define $A_n = \{x \in A \mid |x| \leq n\}$

Theorem: Let $L'$ be computable by streaming algorithm $A$ using $\leq S(n)$ bits of space on all strings of length up to $n$. Then for all $n$, there is a DFA $M$ with $< 2^{S(n)+1}$ states such that $L'_n = L(M)_n$

That is, for all streaming algorithms $A$ using $S(n)$ space, there’s a DFA $M$ of $< 2^{S(n)+1}$ states such that $A$ and $M$ agree on all strings of length up to $n$.

Note: $L'_n$ is always regular! (It’s finite!)
For any $A \subseteq \Sigma^*$ define $A_n = \{x \in A \mid |x| \leq n\}$

**Theorem:** Let $L'$ be computable by streaming algorithm $A$ using $\leq S(n)$ bits of space on all strings of length up to $n$. Then for all $n$, there is a DFA $M$ with $< 2^{S(n)+1}$ states such that $L'_n = L(M)_n$

**Proof Idea:**
- States of $M =$ at most $2^{S(n)+1} - 1$ possible memory configurations of $A$, over strings of length up to $n$
- Start state of $M =$ Initialized memory of $A$
- Transition function = Mimic how $A$ updates its memory
- Final states of $M =$ Subset of memory configurations in which $A$ would accept, if the string ended there
Streaming Lower Bounds via DFAs

For any $A \subseteq \Sigma^*$ define $A_n = \{x \in A \mid |x| \leq n\}$

**Theorem:** Let $L'$ be computable by streaming algorithm $A$ using $S(n)$ bits of space on all strings of length up to $n$. Then for all $n$, there is a DFA $M$ with $< 2^{S(n)+1}$ states such that $L'_n = L(M)_n$

**Corollary:** Suppose for some $n$, every DFA $M$ agreeing with $L'_n$ requires at least $Q(n) := 2^{S(n)+1}$ states. Then $L'$ is *not computable* by a streaming algorithm using $S(n) = \log_2(Q(n)/2) = \log_2(Q(n))-1$ space! That is, $L'$ requires at least $\log_2(Q(n))$ space for some $n$. 