Lecture 6: The Myhill-Nerode Theorem and Streaming Algorithms
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
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 $x, y \in \Sigma^*$

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

Say $x$ and $y$ are indistinguishable to $L$ iff $x \equiv_L y$

Claim: $\equiv_L$ is an equivalence relation

Proof is easy!
The Myhill-Nerode Theorem:

A language $L$ is regular if and only if the number of equivalence classes of $\equiv_L$ is finite.

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

Define the relation: $x \sim_M y \iff \Delta(q_0, x) = \Delta(q_0, y)$

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

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

Proof: $x \sim_M y$ 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: Number of equiv. classes of $\equiv_L$ is at most the number of equiv. classes of $\sim_M$ (which is $|Q|$)
Let $L \subset \Sigma^*$ and $x, y \in \Sigma^*$

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

$(\Leftarrow)$ 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 $\equiv_L$
- $q_0 = [\varepsilon] = \{ y \mid y \equiv L \varepsilon \}$
- $\delta([x], \sigma) = [x \sigma]$ (well-defined??)
- $F = \{ [x] \mid x \in L \}$

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

Proof: Consider $M$ running on $x_1 \ldots x_n \in \Sigma^*$, where each $x_i \in \Sigma$.

$M$ starts in state $[\varepsilon]$, reads $x_1$ and moves to $[x_1]$, then $[x_1 x_2]$, ...

... and ends in state $[x_1 \ldots x_n]$.

So, $M$ accepts $x_1 \ldots x_n \iff [x_1 \ldots x_n] \in F$

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

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

**Proof:** Consider the infinite set of strings
\[
S = \{0, 00, 000, \ldots, 0^n, \ldots\}
\]
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!
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:
- accept if B=1 and C > 0, else reject

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

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

The program is not part of the memory.

Space usage of A:

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

Initialize: z := 0, s := false, fail := false

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

When stream stops:
- **accept** if and only if (not fail) and (\(z = 0\))

\(z\) = how many more times 0 appears than 1
\(s\) = “Started reading 1s yet?”
\(fail\) = “Reject for certain?”

On all strings of length \(n\), algorithm uses \((\log_2 n) + k\) bits of memory
**Theorem:** Suppose a language $L'$ can be recognized by a 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

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

Theorem: Suppose $L'$ is computable by a streaming algorithm $A$ using $\leq f(n)$ bits of space, on all strings of length up to $n$.

Then for all $n$, there is a DFA $M$ with $< 2^{f(n)+1}$ states such that $L'_n = L(M)_n$

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

For any $L \subseteq \Sigma^*$ define $L_n = \{x \in L \mid |x| \leq n\}$
DFAs and Streaming

For any \( L \subseteq \Sigma^* \) define \( L_n = \{ x \in L \mid |x| \leq n \} \)

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

**Proof Idea:** States of \( M = \) all \( 2^{f(n)+1} - 1 \) 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
Example: \( L = \{ x \mid x \text{ has more 1's than 0's} \} \)

Initialize: \( C := 0 \) and \( B := 0 \)
When the next symbol \( \sigma \) is read,
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 the stream stops,
  accept if \( B=1 \) and \( C > 0 \), else reject

Example: A DFA that agrees with \( L \) on all strings of length \( \leq 2 \)
DFAs and Streaming

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

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

**Corollary:** Suppose for all $n$, every DFA $M$ with $L'_n = L(M)_n$ needs $\geq 2^{f(n)+1}$ states. Then $L'$ is *not computable* by a streaming algorithm that uses $f(n)$ bits of space!
L = \{ x \mid x \text{ has more 1's than 0's} \}

Is there a streaming algorithm for L using much \textit{less than} \((\log_2 n)\) space?

**Theorem:** Every streaming algorithm for L needs at least \((\log_2 n)-2\) bits of space

We will use:

- Myhill-Nerode Theorem
- The connection between DFAs and streaming
L = \{x \mid x \text{ has more } 1\text{'s than } 0\text{'s}\}

**Theorem:** Every streaming algorithm for L requires at least \((\log_2 n)-2\) bits of space

**Proof Idea:** Let n be even, and \(L_n = \{x \in L \mid |x| \leq n\}\)

We will give a set \(S_n\) of \(\frac{n}{2} + 1\) strings such that each pair in \(S_n\) is distinguishable to \(L_n\)

**Myhill-Nerode Thm** \(\Rightarrow\) Every DFA recognizing \(L_n\) needs at least \(\frac{n}{2}+1\) states

\(\Rightarrow\) Every streaming algorithm for L needs at least \((\log n)-2\) bits of memory on strings of length n
**Theorem:** Every streaming algorithm for \( L \) requires at least \((\log_2 n) - 2\) bits of space.

Suppose we partition all strings into their equivalence classes under \( \equiv_{L_n} \).

But the number of states in a DFA recognizing \( L_n \) is at least the number of equivalence classes under \( \equiv_{L_n} \).
L = \{x \mid x \text{ has more 1's than 0's}\}

Theorem: Every streaming algorithm for L requires at least \((\log_2 n) - 2\) bits of space

Proof (Slide 1): Let \(S_n = \{0^{n/2-i}1^i \mid i = 0, \ldots, n/2\}\)
Let \(x = 0^{n/2-k}1^k\) and \(y = 0^{n/2-j}1^j\) be from \(S_n\), with \(k > j\)

Claim: \(z = 0^{k-1}1^{n/2-(k-1)}\) distinguishes \(x\) and \(y\) in \(L_n\)

\(xz\) has \(n/2-1\) zeroes and \(n/2+1\) ones \(\Rightarrow xz \in L_n\)

\(yz\) has \(n/2+(k-j-1)\) zeroes and \(n/2-(k-j-1)\) ones
But \(k-j-1 \geq 0\) ... so \(yz \not\in L_n\)

So the string \(z\) distinguishes \(x\) and \(y\), and \(x \not\equiv_{L_n} y\)