Lecture 4:
More on Regexps,
Non-Regular Languages
Announcements:
- Pset 1 is on piazza (as of last night)
- No class next Tuesday
- Come to office hours?
Deterministic Finite Automata

Computation with finite memory
Non-Deterministic Finite Automata

Computation with finite memory

and magical guessing
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: \( \overline{A} = \{ w \in \Sigma^* \mid w \not\in A \} \)

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?*

*DFAs find “patterns” in strings; how to describe them?*
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, $(1)^*$, $(0+1)^*$, $(((0)*1)*1) + (10)$
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:

$(R_1 R_2)$ 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 $\{10\} \cup \{0^k1 \mid k \geq 0\}$
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 $w \in L(R)$.

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

$110101110101100$ matches $(0+1)^*0$

$L((0+1)^*0) = \{ w \in \{0,1\}^* \mid w \text{ ends in a } 0 \}$
DFAs $\equiv$ NFAs $\equiv$ Regular Expressions!

L can be represented by some regexp
\[ \iff \quad \text{L is regular} \]

We saw: L can be represented by some regexp
\[ \implies \quad \text{L is regular} \]

Every regexp can be converted into an NFA

Now we’ll show: L is regular
\[ \implies \quad \text{L can be represented by some regexp} \]

Every DFA can be converted into a regexp
Generalized NFAs (GNFA)

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

Rather than reading in just 0 or 1 letters from the string on an arc, we can read in *entire substrings*.
This GNFA recognizes $L(a^*b(cb)^*a)$, the set of strings matched by $a^*b(cb)^*a$
Add unique start and accept states

Goal: Replace $\text{DFA}$ with a single regexp $R$

Then, $L(R) = L(\text{DFA})$
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.
While the machine has more than 2 states:

In general:
In general:

While the machine has more than 2 states:

\[ R(q_1, q_2)R(q_2, q_2)^*R(q_2, q_3) + R(q_1, q_3) \]
\[ R(q_0, q_3) = (a^*b)(a+b)^* \]

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

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

$\text{CONVERT}(G)$: (Takes a GNFA, outputs a regexp)

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

If $\#\text{states} > 2$

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

define $Q' = Q - \{q_{\text{rip}}\}$

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

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

return $\text{CONVERT}(G')$

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

Claim: $L(G') = L(G)$  
[Sipser, p.73-74]
Theorem: Let $R = \text{CONVERT}(G)$. Then $L(R) = L(G)$.
Proof by induction on $k$, the number of states in $G$

Base Case: $k = 2$ \quad \text{CONVERT outputs } R(q_{\text{start}}, q_{\text{acc}}) \checkmark

Inductive Step:

Assume theorem is true for $k-1$ state GNFAs
Let $G$ have $k$ states. Let $G'$ be the $k-1$ state GNFA.
First show that $L(G) = L(G')$ \cite{Sipser, p.73--74}
$G'$ has $k-1$ states, so by induction,
$L(G') = L(\text{CONVERT}(G')) = L(\text{CONVERT}(G)) = L(R)$ by I.H.

Therefore $L(R) = L(G)$. \quad \text{QED}
Convert to a regular expression
\( q_1 \) \( \longrightarrow \) \( q_2 \) with transitions:

- \( \varepsilon \) on \( q_1 \) to \( q_2 \)
- \( bb \) on \( q_1 \) to \( q_1 \)
- \( a \) on \( q_1 \) to back to \( q_1 \)
- \( a + ba \) on \( q_1 \) to \( q_2 \)
- \( b \) on \( q_2 \) to \( q_2 \)
- \( \varepsilon \) on \( q_2 \) to \( q_2 \)
\[ bb + (a + ba)b^*a \]

\[ (bb + (a + ba)b^*a)^* (b + (a + ba)b^*) \]
Convert the NFA to a regular expression
Convert the NFA to a regular expression
Convert the NFA to a regular expression

\((a + b)b^*b\)
Convert the NFA to a regular expression

\[((a + b)b^*b(bb^*b)^*)^*(\varepsilon + (a + b)b^*b(bb^*b)^*)\]
Many Languages Are Not Regular:

Limitations on DFAs/NFAs

a.k.a.

“Lower Bounds” on DFAs/NFAs
Regular or Not? 

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

C = \{ w \mid w \text{ has equal number of 1s and 0s}\} 

NOT REGULAR!

D = \{ w \mid w \text{ has equal number of occurrences of 01 and 10}\} 

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

\[ D = \{ \text{w} \mid \text{w has equal number of occurrences of 01 and 10}\} \]

\[ = \{ \text{w} \mid \text{w} = 1, \text{w} = 0, \text{or } \text{w} = \varepsilon, \text{ or } \]
\[ \text{w starts with a 0 and ends with a 0, or } \]
\[ \text{w starts with a 1 and ends with a 1 } \} \]

\[ 1 + 0 + \varepsilon + 0(0+1)*0 + 1(0+1)*1 \]

Claim:
A string w has equal occurrences of 01 and 10
\[\Leftrightarrow\] w starts and ends with the same bit!
A Language With No DFA

Theorem: $A = \{0^n1^n \mid n \geq 0\}$ is not regular

Big Idea:
No DFA can “remember” the number of 0’s, if it reads more 0’s than its number of states.

In that case, the DFA can’t accurately compare the number of 0’s to the number of 1s!
Theorem: $A = \{ 0^n 1^n \mid n \geq 0 \}$ is not regular

Proof: By contradiction. Assume $A$ is regular. Then $A$ has a DFA $M$ with $Q$ states, for some $Q > 0$. Suppose we run $M$ on the input $w = 0^{Q+1}$. By the pigeonhole principle, some state $q$ of $M$ is visited more than once while reading in $w$.

Therefore, $M$ is in state $q$ after reading $0^S$, and $M$ is in state $q$ after reading $0^R$, for some $R < S \leq Q+1$. What happens when $M$ reads $1^S$ starting from state $q$?

M must accept, because $0^S 1^S$ in $A$. **Contradiction!**

AND $M$ must reject, because $0^R 1^S$ is not in $A$. 😞
Counting: Hard With Finite Brain

Thm: \( EQ = \{w \mid w \text{ has an equal number of 0s and 1s}\} \) is not regular

Proof: By contradiction. Assume \( EQ \) is regular.

Observation: \( EQ \cap L(0^*1^*) = \{0^n1^n \mid n \geq 0\} \)

If \( EQ \) is regular and \( L(0^*1^*) \) is regular then \( EQ \cap L(0^*1^*) \) is regular.
(Regular Languages are closed under intersection!)

But \( \{0^n1^n \mid n \geq 0\} \) is not regular!

\textit{Contradiction!}
Palindromes: Hard With Finite Brain

Theorem: PAL = \{w \mid w = w^R\} is not regular

Proof: By contradiction. Assume PAL is regular.
Then PAL has a DFA M with Q states, for some Q > 0.

Run M on the input \( w = 10^{Q+1} \)
By the pigeonhole principle, some state \( q \) of M is
visited more than once, while reading in the 0’s of \( w \).

Therefore, M is in state \( q \) after reading \( 10^S \),
and is also in \( q \) after reading \( 10^R \), for some \( R < S \leq Q+1 \).

What happens when M reads \( 10^S1 \) starting from state \( q \)?
M must accept, because \( 10^S10^S1 \) is in PAL. Contradiction!
AND M must reject, because \( 10^R10^S1 \) is not...
How to Make a DFA Lose Its Mind

Want to show: Language $L$ is not regular

Proof: By contradiction. Assume $L$ is regular.
So $L$ has a DFA $M$ with $Q$ states, for some $Q > 0$.

YOU: Cleverly pick strings $x$, $y$ where $|y| > Q$

Run $M$ on $xy$. *Pigeons tell us: Some state $q$ of $M$ is visited more than once, while reading in $y$.*
Therefore, $M$ is in state $q$ after reading $xy'$, and is in $q$ after reading $xy''$, for two prefixes $y'$ and $y''$ of $y$.

YOU: Cleverly pick string $z$ so that *exactly one* of $xy'z$ and $xy''z$ is in $L$

But $M$ will give the same output on both! *Contradiction!*
Minimizing DFAs
Does this DFA have a minimal number of states?

NO
Is this minimal?

How can we tell in general?
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!!*
In general, there isn’t a uniquely minimal NFA
Distinguishing states with strings

For a DFA $M = (Q, \Sigma, \delta, q_0, F)$, and $q \in Q$, let $M_q$ be the DFA equal to $(Q, \Sigma, \delta, q, F)$

Def. $w \in \Sigma^*$ distinguishes states $p$ and $q$ if:

$M_p$ accepts $w \iff M_q$ rejects $w$

OR
Distinguishing states with strings

For a DFA $M = (Q, \Sigma, \delta, q_0, F)$, and $q \in Q$, let $M_q$ be the DFA equal to $(Q, \Sigma, \delta, q, F)$.

Def. $w \in \Sigma^*$ distinguishes states $p$ and $q$ if: $M_p$ and $M_q$ have different outputs on input $w$
Distinguishing two states

Def. \( w \in \Sigma^* \) *distinguishes* states \( p \) and \( q \) iff \( M_p \) and \( M_q \) have *different outputs* on \( w \)

Here... read this

I’m in \( p \) or \( q \), but which?

How?

Ok, I’m *accepting*!
Must have been \( p \)

Ok, I’m *rejecting*!
Must have been \( q \)