Announcements:
- Pset 1 is on piazza (as of last night)
- If you don’t have piazza access but are registered for 6.045, send email to TAs with your mit.edu address!
Deterministic Finite Automata

Computation with finite memory
Non-Deterministic Finite Automata

Computation with finite memory

and “guessing”
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

$(R_1R_2)$, $(R_1 + R_2)$, 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:

$(R_1R_2)$ 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 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 $(0+1)^*0$
DFAs \equiv NFAs \equiv Regular Expressions!

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

We saw: L can be represented by some regexp
\implies L is regular

Every regexp can be converted into an NFA

Now we’ll show: L is regular
\implies 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*.
Generalized NFA (GNFA)

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

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 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

---

![Diagram](image_url)
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

\[ ab^*c \]
In general:

While the machine has more than 2 states:
In general:

$$R(q_1, q_2)R(q_2, q_2)^*R(q_2, q_3) + R(q_1, q_3)$$

While the machine has more than 2 states:
$R(q_0, q_3) = (a*b)(a+b)^*$

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, 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. $\delta(q,\sigma_i) = q'$

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

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

If #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 $\text{CONVERT}(G')$

Claim: $L(G') = L(G)$

[Sipser, p.73-74]

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

Proof by induction on $k$, the number of states in $G$

Base Case: $k = 2$ \hspace{5em} \text{CONVERT outputs } R(q_{\text{start}}, q_{\text{acc}})

Inductive Step:

Assume theorem is true for $k-1$ state GNFA$\text{s}$

Let $G$ have $k$ states. Let $G'$ be the $k-1$ state GNFA.

First show that $L(G) = L(G')$ \hspace{5em} [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 $L(R) = L(G)$. \hspace{5em} \text{QED}$
Convert the DFA to a regular expression
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

\((a + b)b^*b(bb^*b)^*\)

\(((a + b)b^*b(bb^*b)^*a)^* (\varepsilon + (a + b)b^*b(bb^*b)^*)\)
DEFINITION

DFAs

Regular Languages

NFAs

Regular Expressions

GNFAs
Some Languages Are Not Regular:

Limitations on DFAs/NFAs

a.k.a.

“Lower Bounds” on DFAs/NFAs
\[ \Sigma = \{0, 1\} \]

**Regular or Not?**

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

**NOT REGULAR!**

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

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

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

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

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

Claim:
A string \( w \) has equal occurrences of 01 and 10
\( \iff w \) starts and ends with the same bit.
A Language That Has No DFA

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

Proof Idea:
No DFA can “remember” the number of 0’s, if we feed it 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!
A Language That Has No DFA

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

Proof: By contradiction. Assume \( A \) is regular. Then \( A \) has a DFA \( D \) with \( P \) states, for some \( P > 0 \).

Consider feeding input \( w = 0^{P+1} \) to \( D \).

By pigeonhole principle, \textit{some state} \( q \) \textit{of} \( D \) \textit{is visited more than once while reading in} \( w \).

Therefore, \( D \) is in state \( q \) after reading \( 0^S \), \textit{and} is also in \( q \) after reading \( 0^R \), for some \( R < S \leq P+1 \).

What happens when \( D \) reads \( 1^S \) starting from state \( q \)?

\( D \) must \textit{accept}, because \( 0^S 1^S \) in \( A \).

\( D \) must \textit{reject}, because \( 0^R 1^S \) is not in \( A \). \textit{Contradiction!}
Another Language That Has No DFA

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

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

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

Contradiction!
The Pumping Lemma: Structure in Regular Languages

Let $L$ be any regular language

There is a positive integer $P$ s.t.

for all strings $w \in L$ with $|w| \geq P$

there are $x, y, z$ where $w = xyz$, and:

1. $|y| > 0$ (that is, $y \neq \varepsilon$)
2. $|xy| \leq P$
3. $xz \in L$ and $xyyz \in L$ (in fact, $xy\cdots yz \in L$)

Why is it called the pumping lemma? The word $xyz$ can be \textit{pumped} into a longer string $xyyz$...
Proof: Let M be a DFA that recognizes L

Let \( P \) be the number of states in M

Let \( w \) be a string where \( w \in L \) and \( |w| \geq P \)

We show: \( w = xyz \)

1. \( |y| > 0 \)
2. \( |xy| \leq P \)
3. \( xz \in L \) and \( xyyz \in L \)

Claim: There must exist \( j \) and \( k \) such that \( 0 \leq j < k \leq P \), and \( q_j = q_k \)
Proving a Language is NOT Regular

Let L be any language.

Suppose for every integer \( P \) there is a \( w \in L \) with \( |w| \geq P \) so that for all ways to write \( w = xyz \), where:

1. \( |y| > 0 \) (that is, \( y \neq \varepsilon \))
2. \( |xy| \leq P \)

Either \( xz \not\in L \) or \( xyyz \not\in L \)

THEN \( L \) is not regular!
Theorem: \( \text{EQ} = \{ w \mid w \text{ has equal number of 1s and 0s} \} \)

is not regular.

By contradiction. Assume \( \text{EQ} \) is regular.

1. Let \( P \) be as in pumping lemma.
2. Let \( w = 0^P1^P \) and note that \( w \in \text{EQ} \).
3. If \( \text{EQ} \) is regular, then there is a way to write \( w \) as \( w = xyz, \ |y| > 0, \ |xy| \leq P, \) and \( xyyz, xz \) is also in \( \text{EQ} \).

Claim: The string \( y \) must be all zeroes.

Why? Because \( |xy| \leq P \) and \( w = xyz = 0^P1^P \).

But then \( xyyz \) has more 0s than 1s! Contradiction!
Let’s prove that \( SQ = \{0^{n^2} \mid n \geq 0\} \) is not regular.

Assume \( SQ \) is regular. Let \( w = 0^{p^2} \).

If \( SQ \) is regular, then we can write \( w = xyz, \ |y| > 0, |xy| \leq P \), such that \( xz \) and \( xyyz \) are also in \( SQ \).

We have: \( xyyz = 0^{p^2+|y|} \).

Observe that \( 0 < |y| \leq P \).

So \( |xyyz| = p^2 + |y| \leq p^2 + P < p^2 + 2P + 1 = (p+1)^2 \).

And \( p^2 < |xyyz| < (p+1)^2 \).

Therefore \( |xyyz| \) is not a perfect square!

Hence \( 0^{p^2+|y|} = xyyz \notin SQ \), so our assumption must be false.

That is, \( SQ \) is not regular!