CS 154

Finite Automata vs Regular Expressions, Non-Regular Languages
Deterministic Finite Automata

Computation with finite memory
Non-Deterministic Finite Automata

Computation with finite memory

and “guessing”
Regular Languages are closed under all of the following operations:

- **Union:** $A \cup B = \{ w | w \in A \text{ or } w \in B \}$
- **Intersection:** $A \cap B = \{ w | w \in A \text{ and } w \in B \}$
- **Complement:** $\neg A = \{ w \in \Sigma^* | w \notin A \}$
- **Reverse:** $A^R = \{ w_1 \ldots w_k | w_k \ldots w_1 \in A \}$
- **Concatenation:** $A \cdot B = \{ vw | v \in A \text{ and } w \in B \}$
- **Star:** $A^* = \{ w_1 \ldots w_k | k \geq 0 \text{ and each } w_i \in A \}$
Regular Expressions

Computation as simple, logical description

A totally different way of thinking about computation:

*What is the complexity of describing the strings in the language?*
Inductive Definition of Regexp

Let Σ be an alphabet. We define the regular expressions over Σ inductively:

For all σ ∈ Σ, σ is a regexp

ε is a regexp

∅ is a regexp

If R₁ and R₂ are both regexps, then

(R₁R₂), (R₁ + R₂), and (R₁)* are regexps
Precedence Order: $\ast$

then $\cdot$

then $+$

Example: $R_1 \ast R_2 + R_3 = ((R_1 \ast) \cdot R_2) + R_3$
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:

\[ (R_1R_2) \text{ represents } L_1 \cdot L_2 \]
\[ (R_1 + R_2) \text{ represents } L_1 \cup L_2 \]
\[ (R_1)^* \text{ represents } L_1^* \]

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

Example: $01010$ matches the regexp $(01)^*0$.
Assume $\Sigma = \{0,1\}$

\{ $w$ | $w$ has exactly a single 1 \} = 0^*10^*$

\{ $w$ | $w$ 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 $\Sigma = \{0,1\}$

\[
\{ w \mid \text{every odd position in } w \text{ is a } 1 \}
\]

\[(1(0 + 1))^{*}(1 + \varepsilon)\]
DFAs $\equiv$ NFAs $\equiv$ Regular Expressions!

$L$ can be represented by some regexp

$\iff$ L is regular
L can be represented by some regexp
\Rightarrow \text{ L is regular}
Given any regexp R, we will construct an NFA N s.t. N accepts \textit{exactly} the strings accepted by R.

Proof by induction on the \textit{length} of the regexp R:

\textbf{Base Cases (R has length 1):}

- $R = \sigma$
- $R = \epsilon$
- $R = \emptyset$
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 \(R\):

\[
R = R_1 + R_2 \quad \text{By induction, } R_1 \text{ and } R_2 \text{ represent some regular languages, } L_1 \text{ and } L_2
\]

\[
R = R_1 R_2 \quad \text{But } L(R) = L(R_1 + R_2) = L_1 \cup L_2
\]

\[
R = (R_1)^* \quad \text{so } L(R) \text{ is regular, by the union theorem!}
\]
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 $L(R) = L(R_1 \cdot R_2) = L_1 \cdot L_2$

- $R = (R_1)^*$  
  so $L(R)$ is regular by the concatenation 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 \( L(R) = L(R_1^*) = L_1^* \)

- \( R = (R_1)^* \)  
  so \( L(R) \) is regular, by the star theorem
**Induction Step:** Suppose every regexp of length \(< k\) represents some regular language.

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

Three possibilities for \(R\):

\[
R = R_1 + R_2 \quad \text{By induction, } R_1 \text{ and } R_2 \text{ represent some regular languages, } L_1 \text{ and } L_2
\]

\[
R = R_1 R_2 \quad \text{But } L(R) = L(R_1^*) = L_1^*
\]

\[
R = (R_1)^* \quad \text{so } L(R) \text{ is regular, by the star theorem}
\]

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

\[ \iff L \text{ is a regular language} \]

**Idea:** Transform an NFA 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 a step, we can read in entire substrings
A GNFA is a 5-tuple $G = (Q, \Sigma, R, q_{\text{start}}, q_{\text{accept}})$

$Q, \Sigma$ are states and alphabet

$R : (Q - \{q_{\text{accept}}\}) \times (Q - \{q_{\text{start}}\}) \rightarrow \mathcal{R}$

is the transition function

$q_{\text{start}} \in Q$ is the start state

$q_{\text{accept}} \in Q$ is the (unique) accept state

$\mathcal{R} = \text{set of all regular expressions over } \Sigma$
A GNFA is a 5-tuple $G = (Q, \Sigma, R, q_{\text{start}}, q_{\text{accept}})$

Let $w \in \Sigma^*$ and let $G$ be a GNFA. 

**G accepts $w$** if $w$ can be written as $w = w_1 \cdots w_k$ where $w_i \in \Sigma^*$ and there is a sequence $r_0, r_1, \ldots, r_k \in Q$ such that

- $r_0 = q_{\text{start}}$
- $w_i$ matches $R(r_{i-1}, r_i)$ for all $i = 1, \ldots, k$, and
- $r_k = q_{\text{accept}}$

$L(G)$ = set of all strings that $G$ accepts
= “the language recognized by $G$”
This GNFA recognizes $L(a^*b(cb)^*a)$

Is aaabcbcbaba accepted or rejected?
Is bba accepted or rejected?
Is bcba accepted or rejected?
Add unique start and accept states
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

\[ \begin{array}{c}
\text{NFA} \\
\end{array} \]
While the machine has more than 2 states:

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

In general:

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

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 $R(q, q')$ to be $\sigma$ if $\delta(q, \sigma) = q'$, else $\emptyset$

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

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

If #states > 2

select $q_{\text{rip}} \in Q$ 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: $L(G') = L(G)$
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{1cm} \text{CONVERT outputs } R(q_{\text{start}}, q_{\text{acc}})$ \hspace{1cm} ✓

Inductive Step:

Assume theorem is true for $k-1$ state GNFFAs

Let $G$ have $k$ states. Let $G'$ be the $k-1$ state GNFA obtained by ripping out a state.

We already claimed $L(G) = L(G')$ \hspace{1cm} \cite{Sipser, p.73--74}

$G'$ has $k-1$ states, so by induction,

$L(G') = L(\text{CONVERT}(G')) = L(R)$

Therefore $L(R) = L(G)$. \hspace{1cm} QED
\( bb + (a + ba)b^*a \)

\( q_1 \)

\( \varepsilon \)

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

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

\[ q_1 \rightarrow a, b \rightarrow q_2 \rightarrow b \]

\[ q_1 \rightarrow a \rightarrow q_3 \rightarrow b \]

\[ q_2 \rightarrow b \rightarrow q_2 \]
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

\[(a + b)b*b(bb*b)*\]

\[(a + b)b*b(bb*b)*a\]

\[((a + b)b*b(bb*b)*a)^*(\epsilon + (a + b)b*b(bb*b)*)\]
DEFINITION

DFAs \rightarrow \leftarrow NFAs

Regular Languages

Regular Expressions
Some Languages Are Not Regular:

Limitations on DFAs
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!**
\{ 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 } w \text{ starts with a } 0 \text{ and ends with a } 0, \text{ or } w \text{ 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.
The Pumping Lemma: Structure in Regular Languages

Let $L$ be a regular language

Then there is a positive integer $P$ s.t.

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

there is a way to write $w = xyz$, where:

1. $|y| > 0$ (that is, $y \neq \varepsilon$)
2. $|xy| \leq P$
3. For all $i \geq 0$, $xy^iz \in L$

Why is it called the pumping lemma? The word $w$ gets *pumped* into longer and longer strings...
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. $xy^iz \in L$ for all $i \geq 0$

There must exist $j$ and $k$ such that $0 \leq j < k \leq P$, and $q_j = q_k$