Lecture 6:

- Regexes = NFAs
- Non-regular languages

Reading:

Sipser Ch 1.3
“Myhill-Nerode” note
Sipser Ch 1.4 (optional)

Mark Bun
February 10, 2021
Regular Expressions – Syntax

A regular expression $R$ is defined recursively using the following rules:

1. $\varepsilon$, $\emptyset$, and $a$ are regular expressions for every $a \in \Sigma$

2. If $R_1$ and $R_2$ are regular expressions, then so are
   
   \[ (R_1 \cup R_2), (R_1 \circ R_2), \text{ and } (R_1^*) \]

Examples: (over $\Sigma = \{a, b, c\}$) (with simplified notation)

\[
ab \quad ab^* c \cup (a^* b)^* \quad \emptyset
\]
Regular Expressions – Semantics

$L(R) = \text{the language a regular expression describes}$

1. $L(\emptyset) = \emptyset$
2. $L(\varepsilon) = \{\varepsilon\}$
3. $L(a) = \{a\} \text{ for every } a \in \Sigma$
4. $L((R_1 \cup R_2)) = L(R_1) \cup L(R_2)$
5. $L((R_1 \circ R_2)) = L(R_1) \circ L(R_2)$
6. $L\left(\left(R_1^*\right)\right) = (L(R_1))^*$

Example: $L(a^* b^*) = \{a^m b^n \mid m, n \geq 0\}$
Regular Expressions Describe Regular Languages

**Theorem:** A language $A$ is regular if and only if it is described by a regular expression

**Theorem 1:** Every regular expression has an equivalent NFA

**Theorem 2:** Every NFA has an equivalent regular expression
Regular expression -> NFA

Theorem 1: Every regex has an equivalent NFA
Proof: Induction on size of a regex

Base cases:

\[ R = \emptyset \]
\[
\begin{array}{c}
\text{recognize} \\
\emptyset
\end{array}
\]

\[ R = \varepsilon \]
\[
\begin{array}{c}
\text{recognize} \\
\varepsilon
\end{array}
\]

\[ R = a \]
\[
\begin{array}{c}
\text{recognize} \\
\gamma a \Gamma
\end{array}
\]

2/10/2021 CS332 - Theory of Computation
Regular expression -> NFA

**Theorem 1:** Every regex has an equivalent NFA

**Proof:** Induction on size of a regex

Inductive step:

\[ R = (R_1 \cup R_2) \]

\[ R = (R_1 R_2) \]

\[ R = (R_1^*) \]
Example

Convert \((1(0 \cup 1))^*\) to an NFA
Regular Expressions Describe Regular Languages

**Theorem:** A language $A$ is regular if and only if it is described by a regular expression

**Theorem 1:** Every regular expression has an equivalent NFA

**Theorem 2:** Every NFA has an equivalent regular expression
NFA -> Regular expression

Theorem 2: Every NFA has an equivalent regex

Proof idea: Simplify NFA by “ripping out” states one at a time and replacing with regexes
Generalized NFAs (GNFA)

- Every transition is labeled by a regex
- One start state with only outgoing transitions
- Only one accept state with only incoming transitions
- Start state and accept state are distinct
Generalized NFA Example

\[ R(q_s, q) = a^* b \]
\[ R(q_a, q) = \emptyset \]
\[ R(q, q_s) = \emptyset \]
Which of the following strings is accepted by this GNFA?

a) $aaa$

b) $aabb$

c) $bbb$

d) $bba$
NFA -> Regular expression

\[
\sum_{i=3}^{k+2} i^2 \approx O(k^3)
\]

\[
k + 2 \text{ states} \\
(\text{perform } (k+2)^2 \text{ replacements})
\]

\[
k + 1 \text{ states} \\
( (k+1)^2 \text{ replace moves})
\]

\[
2 \text{ states}
\]
NFA -> GNFA

- Add a new start state with no incoming arrows.
- Make a unique accept state with no outgoing arrows.
GNFA -> Regular expression

Idea: While the machine has more than 2 states, rip one out and relabel the arrows with regexes to account for the missing state
GNFA -> Regular expression

Idea: While the machine has more than 2 states, rip one out and relabel the arrows with regexes to account for the missing state

1) $a^*b(a \cup b)a$
2) $a^*b(a \cup b)^*a$
3) $a^*b \cup (a \cup b)va$
4) None

$aba \in L((a \cup b)^*)$
$aba \notin L(a^*b^*)$

$q_1 \xrightarrow{a^*b} q_2 \xrightarrow{a} q_3$
GNFA -> Regular expression

Idea: While the machine has more than 2 states, rip one out and relabel the arrows with regexes to account for the missing state.
GNFA -> Regular expression

Idea: While the machine has more than 2 states, rip one out and relabel the arrows with regexes to account for the missing state.
2/10/2021

### Theory of Computation

1) Minimize NFAs

2) Minimize state 2

3) Minimize state 3

4) Minimize

Final regex

\[(baa^*b \cup a^*b)^* \delta_a \]

\[(b(aa^* uc) \cup aa^*)^* \]

\[baa^* b \cup a^*b \]

\[b(\emptyset (aa^* uc) \cup aa^*) \]

\[b(aa^* uc) \cup aa^*b \]
Non-Regular Languages
Motivating Questions

• We’ve seen techniques for showing that languages are regular
  \( \text{NFA, NFA, regex} \)

• How can we tell if we’ve found the smallest DFA recognizing a language?

• Are all languages regular? How can we prove that a language is not regular?
An Example

Claim: Every DFA recognizing \( A \) needs at least 3 states

Proof: Let \( M \) be any DFA recognizing \( A \). Consider running \( M \) on each of \( x = \varepsilon, y = 0, w = 01 \)

1) \( p_2 \neq p_x \) and \( p_2 \neq p_y \) and \( p_2 \) accepts, \( p_1 \) and \( p_y \) are rejects

2) \( p_1 \neq p_y \)

Assume \( p_x = p_y \) (for contradiction) what happens when \( I \) run \( M \) on 1, 01
A General Technique

Definition: Strings $x$ and $y$ are distinguishable by $L$ if there exists $z$ such that exactly one of $xz$ or $yz$ is in $L$.

Ex. $x = \varepsilon$, $y = 0$

Definition: A set of strings $S$ is pairwise distinguishable by $L$ if every pair of distinct strings $x, y \in S$ is distinguishable by $L$.

Ex. $S = \{\varepsilon, 0, 01\}$
A General Technique

**Theorem:** If $S$ is pairwise distinguishable by $L$, then every DFA recognizing $L$ needs at least $|S|$ states.

**Proof:** Let $M$ be a DFA with $< |S|$ states. By the pigeonhole principle, there are $x, y \in S$ such that $M$ ends up in the same state on $x$ and $y$.
Back to Our Example

\[ A = \{ w \in \{0, 1\}^* \mid w \text{ ends with } 01 \} \]

**Theorem:** If \( S \) is pairwise distinguishable by \( L \), then every DFA recognizing \( L \) needs at least \(|S|\) states.

\[ S = \{ \varepsilon, 0, 01 \} \]

- \( \varepsilon, 0 \) \( \text{dit} \)
- \( 0, 01 \) \( \text{dit} \)
- \( \varepsilon, 01 \) \( \text{dit} \)

Then \( \implies S \text{ pairwise dit} \).

\[ \implies \text{Any DFA for } A \text{ needs } 3 \text{ states} \]

\[ S' = \{ \varepsilon, 1, 0, 01 \} \text{ also works} \]