Lecture 6:

- Regexes = NFAs
- Non-regular languages

Reading:
Sipser Ch 1.3
“Myhill-Nerode” note

Mark Bun
September 22, 2022
Regular Expressions – Syntax

A regular expression \( R \) is defined recursively using the following rules:

1. \( \varepsilon, \emptyset, \text{ and } a \) are regular expressions for every \( a \in \Sigma \)

2. If \( R_1 \) and \( R_2 \) are regular expressions, then so are \( (R_1 \cup R_2), (R_1 \circ R_2), \) and \( (R_1^*) \)

Examples: (over \( \Sigma = \{a, b, c\} \)) (with simplified notation)
\[
ab \quad \quad ab^*c \cup (a^*b)^* \quad \emptyset
\]
Regular Expressions – Semantics

$L(R)$ = the language a regular expression describes

1. $L(\emptyset) = \emptyset$
2. $L(\varepsilon) = \{\varepsilon\}$
3. $L(a) = \{a\}$ 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. $L((R_1^*)) = (L(R_1))^*$

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:

\[ R = \emptyset \]

\[ R = \varepsilon \]

\[ R = a \]
Regular expression -> NFA

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

What should the inductive hypothesis be?

a) Suppose some regular expression of length \( k \) can be converted to an NFA

b) Suppose every regular expression of length \( k \) can be converted to an NFA

c) Suppose every regular expression of length at most \( k \) can be converted to an NFA

d) None of the above
Regular expression $\rightarrow$ 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

- 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

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

- a) $aaa$
- b) $aabb$
- c) $bbb$
- d) $bba$
NFA -> Regular expression

$k$ states

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

$k + 2$ states

$k + 1$ states

2 states

$k$ states

GNFA

GNFA

GNFA

GNFA

Regex
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

a) $a^*b(a \cup b)a$

b) $a^*b(a \cup b)^*a$

c) $a^*b \cup (a \cup b) \cup a$

d) None of the above
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.

$q_1$ $R_1$ $R_2$ $R_3$ $R_4$ $q_2$ $q_3$

$q_1$ $R_4$ $q_3$

$q_1$ $R_3$ $q_3$
Non-Regular Languages
Motivating Questions

• We’ve seen techniques for showing that languages are regular

• 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

$$A = \{ w \in \{0, 1\}^* | \text{w ends with 01} \}$$

Claim: *Every* DFA recognizing $A$ needs at least 3 states

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

Definition: Strings $x$ and $y$ are **distinguishable** by $L$ if there exists a string $z$ 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. $S = \{\varepsilon, 0, 01\}$

$A = \{ w \in \{0, 1\}^* \mid w \text{ ends with } 01 \}$
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 same state on $x$ and $y$
Back to Our Example

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

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

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

\[ B = \{ w \in \{0, 1\}^* \mid |w| = 2 \} \]

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

\[ S = \{ \} \]
Distinguishing Extension

Which of the following is a distinguishing extension for $x = 0$ and $y = 00$ for language $B = \{ w \in \{0, 1\}^* \mid |w| = 2 \}$?

a) $z = \varepsilon$

b) $z = 0$

c) $z = 1$

d) $z = 00$