BU CS 332 – Theory of Computation

Lecture 6:
• Regexes = NFAs
• Non-regular languages

Reading:
Sipser Ch 1.3
“Myhill-Nerode” note

Mark Bun
September 21, 2021

https://forms.gle/XT3v76KCagDQB5QL6

-Hw 2 due tonight
- Islam’s OH moved to
  3:30 - 4:30 Mrs B33
  (today only)
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)

\[
\begin{align*}
ab & \quad ab^* c \cup (a^* b)^* \quad \emptyset
\end{align*}
\]
Regular Expressions – Semantics

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

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

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

even length string; each odd position is 1

\[ (1(0 \cup 1))^* \]

Simplification
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 \((G\text{NFA})\)

- 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 these strings is accepted?

Which of the following strings is accepted by this GNFA?

a) $aaa$

b) $aabb$

c) $bbb$

d) $bba$ ✓
NFA -> Regular expression

- NFA $k$ states
- GNFA $k + 2$ states
- GNFA $k + 1$ states
- GNFA $2$ states
- Regex

$q_s \xrightarrow{r} q_o$
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

$q_1 \xrightarrow{a^* b} q_2 \xrightarrow{a} q_3$

$q_1 \xrightarrow{a^* b a} q_3$
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

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.
Step 0: Convert to GNF-A

Step 1: Rip out state 2

Step 2: Rip out state 3

Step 3: Rip out state 1

\[ \epsilon (baa* b U aa*b)^* (b(aa* e U e) U aa*) \]
\[ = (ba^* b U a^* a)^* (b a^* U aa^*) \]
\[ = (ba^* b U a^* a)^* (ba^* U va^*) \]
Non-Regular Languages
Motivating Questions

• We’ve seen techniques for showing that languages are regular
  - Construct DFA
  - Construct NFA
  - Construct 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

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

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

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

Let \( q_x = \text{state } M \text{ reads when reading } x \)

\( q_x = \) reading \( x \)

Let \( q_y = \) reading \( y \)

Claim: \( q_x, q_y, q_w \) are all distinct

\( q_w \neq q_x \), \( q_w \neq q_y \) because \( q_w \) is an accept, \( q_x \) \& \( q_y \) reject

\( q_x \neq q_y \): Assume for contr. that \( q_x = q_y \)

Let \( z = 1 \). Then what does \( M \) do on \( xz = 1 \), and \( yz = 01 \)? Should accept?
A General Technique

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

**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 \( L \).

**Example:** \( x = \varepsilon, \; y = 0 \)

\[ z = 1, \quad xz \notin A, \quad yz \in A \]

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

**Example:** \( S = \{ \varepsilon, 0, 01 \} \)

\begin{align*}
\varepsilon = \varepsilon, \; y = 0 & : \; z = 1 \\
\varepsilon = \varepsilon, \; y = 01 & : \; z = 0 \\
\varepsilon = 0, \; y = 01 & : \; z = 0
\end{align*}
A General Technique

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