San José State University
Department of Computer Science

Ahmad Yazdankhah
ahmad.yazdankhah@sjsu.edu
www.cs.sjsu.edu/~yazdankhah

Regular Expressions

(Part 1)

Lecture 19

Day 20/31

CS 154

Formal Languages and Computability

Fall 2019

Agenda of Day 20

- About Midterm 2 (Reminder 2)
- Solution and Feedback of Quiz 7
- Summary of Lecture 18
- A Few Slides from the Past (Added to the Lecture 18)
- Lecture 19: Teaching ...
  - Regular Expressions (Part 1)

About Midterm 2

Reminder 2

- Midterm #2 (aka Quiz++)
  - Date: Thursday 10/31
  - Value: 15%
  - Topics: Everything covered from the beginning of the semester
  - Type: Closed y ∈ Material

    Material = {Book, Notes, Electronic Devices, Chat, ... }

- The cutoff for this midterm is the end of lecture 18.

Study Guide

- I've announced the type and number of questions via Canvas.

Solution and Feedback of Quiz 7 (Out of 14)

Section	Average	High Score	Low Score
01 (TR 3:00 PM)	12.03	14	7
02 (TR 4:30 PM)	12.13	14	6
03 (TR 6:00 PM)	12.93	14	10

Summary of Lecture 18: We learned ...

Multi-Tape TM

- It did not add more power to standard TM.
- It facilitate the design process.

Nondeterministic TMs

- There are two possible violations in standard TMs:
  - λ-transition
  - When δ is multifunction
- Historically, λ-transitions was not defined in TMs.

Formal Definition

- A nondeterministic TM M is defined by the septuple:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$$

$$\delta: Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{L, R\}}$$

δ is total function.

- We concluded this fact that:
  - A nondeterministic TM is a collection of standard TMs.
  - Nondeterminism does not add power.
  - It just speed up the computation.

Any Question?

A Few Slides From the Past

Added to Lecture Notes 18

Objective of This Lecture

- So far, we've represented formal languages by sets.
- In this lecture, we are going to introduce an alternative mathematical tool for representing them.

- So, in a nutshell:
  - Regular expressions (REGEXs for short) are another mathematical way to represent formal languages.

- They have important practical applications in OS's like Linux/UNIX, and programming languages like Java.

- The question that raises here is:

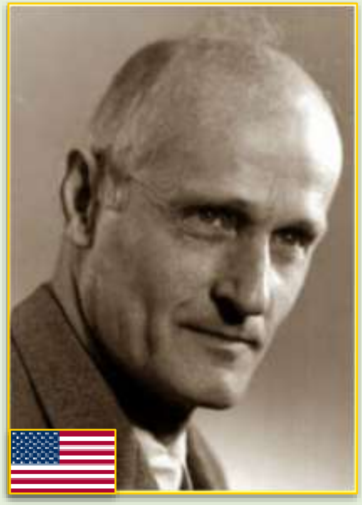
Can REGEXs represent all formal languages?

Regular Expressions (REGEXs)

REGEXs Ingredients

- REGEXs like anything else in this course, have a mathematical base.

- REGEXs was introduced by American mathematician, Stephen C. Kleene (1909-1994) in 1956.



- First, we introduce REGEXs' ingredients.

- REGEXs contain:
  1. Elements
  2. Rules (Formal Definition).

REGEXs Elements

- REGEXs have three types of elements:

1. The symbols of alphabet Σ (e.g. a, b, c, etc.), ϕ, and λ
   - ϕ and λ has special usage that will be covered shortly.
2. ( )
3. Operators:

   + (union)

   . (dot or concatenation)

   * (star-closure)

- Before defining REGEXs' rules, let's take some simple examples to have a taste of them!

REGEXs Examples

Example 1

- Given L = {a} over Σ = {a, b}
- Represent L by a set builder and a REGEX

Solution

- L = {x : x = a}
- r = a   (we'll use "r" as a shortcut for REGEX.)

- So, we just learned how to write the REGEX of all languages that have only one symbol as their string!

  – Theoretically, we can have infinite languages like this!

REGEXs Examples

Concatenation Operator: '.'

- We can concatenate REGEXs symbols: Σ, ϕ, λ

Example 2

- Given L = {ab} over Σ = {a, b}
- r = ?

Solution

- We can decompose {ab} as:
- L = {a} . {b}
- And we've learned in the previous example what the REGEXs of {a} and {b} were. So, ...
- r = a.b

REGEXs Examples

Union Operator: '+'

Example 3

- Given L = {ab, bb, ba} over Σ = {a, b}
- r = ?

Solution

- We can decompose {ab, bb, ba} as:
- L = {ab, bb, ba} = {ab} U {bb} U {ba}
- And we've learned in the previous example what the REGEXs of {ab}, {bb}, and {ba} were. So, ...
- r = a.b + b.b + b.a

REGEXs Examples

Star-Closure Operator: '*'

- Means "Zero or more concatenation"

Example 4

- Given $L = \{a^n : n \geq 0\}$ over $\Sigma = \{a\}$
- $r = ?$

Solution

- $L = \{\lambda, a, aa, aaa, ...\}$
- In formal languages terminology, L can also be represented as:
- $L = \{a\}^*$
- $r = a^*$

REGEXs Examples

Example 5

- Given $L = \{a^n : n \geq 1\}$ over $\Sigma = \{a\}$
- $r = ?$

Solution

- It means, we need at least one 'a'.
- $r = a.a^*$

- The strings of the language L has at least one a.
- So, we put the first 'a' to represent this fact.
- And we put a^* for zero or more a's.
- Note that we don't have expressions like a^+, a^2, a^3 in REGEX.

A Side Note

Different Notations of a Language

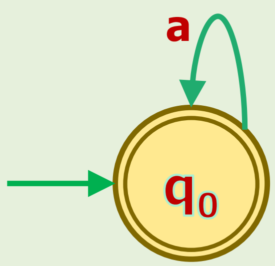
Set builder

$$L = \{a^n : n \geq 0\}$$

Roster Method

$$L = \{\lambda, a, aa, aaa, ...\}$$

NFA



REGEX

$$r = a^*$$

- Why should we study REGEXs?
- REGEXs represent formal languages in a more compact way.
- They are shorthand for set builder notations!
- They are easier to be implemented in computer.

Precedence of Operators

- For more complex REGEXs, there could be some ambiguity.

Example 6

- r = a + b . c
- We may interpret the above REGEX as one of these:

  r = ((a + b) . c)

  r = (a + (b . c))

- Which one is correct?
  - It depends on our definition of operators' precedence.

- So, to remove this ambiguity, we should define some "precedence rules".

Precedence of Operators

- The precedence from the highest to the lowest would be:
  1. Parentheses
  2. Star-closure
  3. Concatenation
  4. Union

Example 7

- $r = a \,.\, b^* + c$
- In fact, $r = ((a \,.\, (b)^*) + c)$
- That is very similar to elementary algebra!

- For simplicity, from now on, we might eliminate '.' (dot) operator.
- So, the above example can be shown as: $r = ab^* + c$

Formal Definition of REGEXs

Formal Definition of REGEXs

1. ϕ, λ, and symbols of Σ are all REGEXs.
   - These are called primitive REGEXs.

2. If r_1 and r_2 are REGEXs, then the following expressions are REGEXs too:

$$\left.\begin{array}{l} r_1 + r_2 \\ r_1 \,.\, r_2 \\ r_1^{\,*} \\ (r_1) \end{array}\right\} \text{Are REGEXs too}$$

3. A string is REGEX if it can be derived recursively from the primitive REGEXs by a finite number of applications of the above rules.

REGEXs Validation

Example 8

- Is r a valid REGEX?
- $r = (a + bc)^* \cdot (c + \phi)$
- Yes, because it has been derived from the REGEX rules.

Example 9

- Is r a valid REGEX?
- $r = (a + b +) \cdot c$
- No, it cannot be derived from the REGEX rules.

REGEX Definition

Repeated

1. ϕ, λ, and $a \in \Sigma$ are all REGEXs.
2. If r_1 and r_2 are REGEXs, then the following expressions are REGEXs too:

   $r_1 + r_2$

   $r_1 \cdot r_2$

   r_1^*

   (r_1)
3. A string is REGEX if it can be derived from the primitive REGEXs by a finite number of applications of the rule #2.

REGEXs - Languages Correspondence

Introduction

- The following REGEX is given:

$$r = a\,(a + b)^*$$

- How can we mathematically calculate what language it represents?

- In other words, how can we calculate L(r)?

$$L(r) = L(a\,(a + b)^*) = ?$$

- We need some mathematical rules!

REGEXs-Languages Correspondence Rules

- If r_1 and r_2 are REGEXs, then the following rules hold recursively:

1. $L(\phi) = \{ \}$
2. $L(\lambda) = \{\lambda\}$
3. $L(a) = \{a\}$ for all $a \in \Sigma$
4. $L(r_1 + r_2) = L(r_1) \cup L(r_2)$
5. $L(r_1 . r_2) = L(r_1) . L(r_2)$
6. $L((r_1)) = L(r_1)$
7. $L(r_1^*) = (L(r_1))^*$

1. ϕ
2. λ
3. $a \in \Sigma$
4. $r_1 + r_2$
5. $r_1 . r_2$
6. (r_1)
7. r_1^*

- The first 3 rules are the termination conditions for the recursion.
- The last 4 rules are used to reduce L(r) to simpler components recursively.

REGEX → Language Examples

REGEX → Language Examples

Example 10

- Given $r = b$
- $L(r) = ?$

Solution

- $L(r) = L(b) = \{b\}$

- We used rule #3.

1. $L(\phi) = \phi$
2. $L(\lambda) = \{\lambda\}$
3. $L(a) = \{a\}$ for all $a \in \Sigma$
4. $L(r_1 + r_2) = L(r_1) \cup L(r_2)$
5. $L(r_1 . r_2) = L(r_1) . L(r_2)$
6. $L((r_1)) = L(r_1)$
7. $L(r_1^*) = (L(r_1))^*$

REGEX → Language Examples

Example 11

- Given $r = b.a$
- $L(r) = ?$

Solution

$L(r) = L(b.a)$

$= L(b) . L(a)$ (rule #5)

$= \{b\} . \{a\}$ (rule #3)

$= \{ba\}$ (by concatenation of languages)

1. $L(\phi) = \phi$
2. $L(\lambda) = \{\lambda\}$
3. $L(a) = \{a\}$ for all $a \in \Sigma$
4. $L(r_1 + r_2) = L(r_1) \cup L(r_2)$
5. $L(r_1 . r_2) = L(r_1) . L(r_2)$
6. $L((r_1)) = L(r_1)$
7. $L(r_1^*) = (L(r_1))^*$

REGEX → Language Examples

Example 12

- Given $r = a + b$
- $L(r) = ?$

1. $L(\phi) = \phi$
2. $L(\lambda) = \{\lambda\}$
3. $L(a) = \{a\}$ for all $a \in \Sigma$
4. $L(r_1 + r_2) = L(r_1) \cup L(r_2)$
5. $L(r_1 . r_2) = L(r_1) . L(r_2)$
6. $L((r_1)) = L(r_1)$
7. $L(r_1^*) = (L(r_1))^*$

Solution

$L(r) = L(a + b)$

$= L(a) \cup L(b)$ (rule #4)

$= \{a\} \cup \{b\}$ (rule #3)

$= \{a, b\}$ (by union of languages)

REGEX → Language Examples

Example 13

- Given $r = a + b.a$
- $L(r) = ?$

1. $L(\phi) = \phi$
2. $L(\lambda) = \{\lambda\}$
3. $L(a) = \{a\}$ for all $a \in \Sigma$
4. $L(r_1 + r_2) = L(r_1) \cup L(r_2)$
5. $L(r_1 . r_2) = L(r_1) . L(r_2)$
6. $L((r_1)) = L(r_1)$
7. $L(r_1^*) = (L(r_1))^*$

Solution

$L(r) = L(a + b.a)$

$= L(a) \cup L(b.a)$ (rule #4)

$= L(a) \cup (L(b) . L(a))$ (rule #5)

$= \{a\} \cup (\{b\} . \{a\})$ (rule #3)

$= \{a\} \cup \{ba\}$ (by concatenation of languages)

$= \{a, ba\}$ (by union of two languages)

REGEX → Language Examples

Example 14

- Given $r = a^*$
- $L(r) = ?$

Solution

$L(r) = L(a^*)$

$= (L(a))^*$ (rule #7)

$= \{a\}^*$ (rule #3)

$= \{a^n : n \geq 0\}$

1. $L(\phi) = \phi$
2. $L(\lambda) = \{\lambda\}$
3. $L(a) = \{a\}$ for all $a \in \Sigma$
4. $L(r_1 + r_2) = L(r_1) \cup L(r_2)$
5. $L(r_1 . r_2) = L(r_1) . L(r_2)$
6. $L((r_1)) = L(r_1)$
7. $L(r_1^*) = (L(r_1))^*$

REGEX → Language Examples

Example 15

- Given $r = (a + b)^*$
- $L(r) = ?$

Solution

$$
\begin{aligned}
L(r) &= L[(a + b)^*] \\
&= [L(a + b)]^* && \text{(rule \#7)} \\
&= [L(a) \cup L(b)]^* && \text{(rule \#4)} \\
&= \{a, b\}^* && \text{(rule \#3)} \\
&= \{w : w \in \Sigma^*\} && \text{(any string over } \Sigma\text{)}
\end{aligned}
$$

1. $L(\phi) = \phi$
2. $L(\lambda) = \{\lambda\}$
3. $L(a) = \{a\}$ for all $a \in \Sigma$
4. $L(r_1 + r_2) = L(r_1) \cup L(r_2)$
5. $L(r_1 \,.\, r_2) = L(r_1) \,.\, L(r_2)$
6. $L((r_1)) = L(r_1)$
7. $L(r_1^*) = (L(r_1))^*$

REGEX → Language Summary

REGEX	Language
b	{b}
b.a	{ba}
a + b	{a, b}
a + b.a	{a, ba}
a*	$\{a^n : n \geq 0\}$
(a + b)*	{a, b}*

REGEX → Language Examples

Example 16

- Given $r = a\ (a + b)^*$
- $L(r) = ?$

Solution

REGEX → Language Examples

Example 17

- Given $r = (aa)^*$
- $L(r) = ?$

Solution

REGEX → Language Examples

Example 18

- Given $r = (bb)^* b$
- $L(r) = ?$

Solution

REGEX → Language Examples

Example 19

- Given r = $(aa)^* \, b \, (bb)^*$
- L(r) = ?

Solution

Associated Languages to REGEXs

Definition

- If REGEX r represents language L, then L is called the "associated language" to r and is denoted by L(r).

- As we saw in the previous slides ...

  If $r = (aa)^*$, then

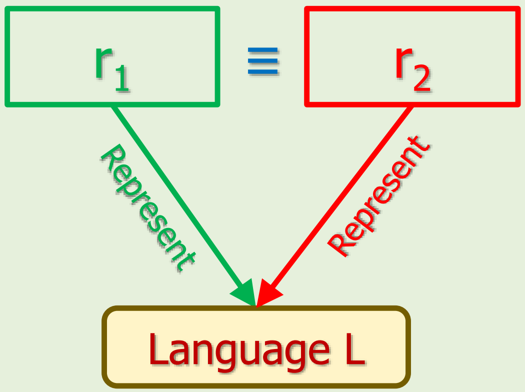
  $L(r) = \{a^{2n} : n \geq 0\}$

Equivalency of REGEXs

Definition

- Two regular expressions r_1 and r_2 are equivalent if both has the same associated language.

$$L(r_1) = L(r_2) \rightarrow r_1 \equiv r_2$$



Equivalency of REGEXs Example

Example 20

- Given r_1 and r_2 as:
- $r_1 = (a + b)^* a$
- $r_2 = (a + b)^* (a + b)^* a$
- Are r_1 and r_2 equivalent?

- Both of these REGEXs are expressing a language containing any string of 'a' and 'b' terminated by an 'a'.

- For a given language L, how many REGEXs we can make?
  - Infinite

REGEXs Identities

REGEXs Identities

- If r, s, and t are REGEXs, and $a, b \in \Sigma$, then:
  1. $r(s + t) = rs + rt$
  2. $(s + t)r = sr + tr$
  3. $(a^*)^* = a^*$
  4. $(a \ldots a)^* a = a (a \ldots a)^*$
  5. $a^* (a + b)^* = (a + b)^* a^* = (a + b)^*$
- We can use the seven mathematical rules mentioned earlier to prove the above identities.
- To prove them, we should show both sides represent the same language.
- For example, for the first one, we should show:

$$L(r(s + t)) = L(rs + rt)$$

REGEXs Identities Examples

Example 21

$a\ b^* + b\ b^*$

$= (a + b)\ b^*$

Example 22

$b^* + b^*\ a$

$= b^*\ (\lambda + a)$

- Note that we used the rule:

$$b^* = b^*\ \lambda$$

Example 23

$aaa^*\ bb^* + aa^*\ b\ bb^*$

$= (aaa^* + aa^*b)\ bb^*$

$= (aa^*\ a + aa^*b)\ bb^*$

$= aa^*\ (a + b)\ bb^*$

- Note that we used the identity:

$$(a \dots a)^*\ a = a\ (a \dots a)^*$$

- and changed aa^* to a^*a

Homework: Identities

- Given r = $(aa)^* (\lambda + ab) (bb)^*$
- $L(r) = ?$

References

1. Linz, Peter, "An Introduction to Formal Languages and Automata, 5th ed.," Jones & Bartlett Learning, LLC, Canada, 2012
2. Michael Sipser, "Introduction to the Theory of Computation, 3rd ed.," CENGAGE Learning, United States, 2013
   ISBN-13: 978-1133187790