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 22/31

CS 154

Formal Languages and Computability

Spring 2019

Agenda of Day 22

- Collecting Quiz 7
- Solution and Feedback of Quiz 6
- Summary of Lecture 18
- Lecture 19: Teaching ...
  - Regular Expressions (Part 1)

Solution and Feedback of Quiz 6 (Out of 20)

Section	Average	High Score	Low Score
01 (TR 3:00 PM)	18.55	20	15
02 (TR 4:30 PM)	17.86	20	8
03 (TR 6:00 PM)	18.29	20	13

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

Any Question?

Summary of Lecture 18: We learned ...

Basic Concepts of Computation

- The algorithm for a problem is ...
  - ... the structure of the TM that solves it.

- The program of a TM is ...
  - ... the transition function of the TM.

Any Question?

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:

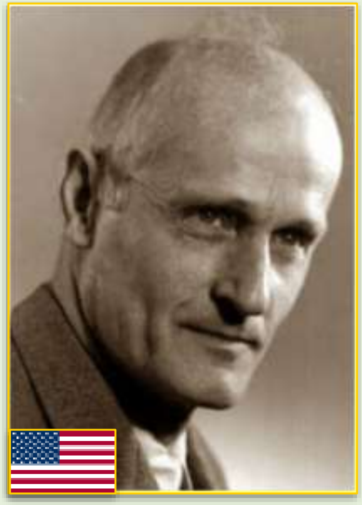
  Can REGEXs represent all formal languages?

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 its ingredients.

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

REGEXs Elements

- REGEXs have three 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 with one symbol as 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

- L = {a} . {b}
- r = a.b

REGEXs Examples

Union Operator: '+'

Example 3

- Given $L = \{ab, bb, ba\}$ over $\Sigma = \{a, b\}$
- $r = ?$

Solution

- $L = \{ab, bb, ba\} = \{ab\} \cup \{bb\} \cup \{ba\}$
- $r = a.b + b.b + b.a$

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 REGEXs.

A Side Note

Different Notations of a Language

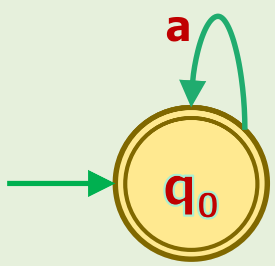
Set builder

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

Roster Method

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

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 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:

$$\left.\begin{array}{l} r_1 + r_2 \\ r_1 \,.\, r_2 \\ r_1^{\,*} \\ (r_1) \end{array}\right\} \quad \text{Regular Expressions}$$

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

REGEXs Validation

Example 8

- Is r a valid REGEX?
- $r = (a + bc)^* . (c + \phi)$

- Yes, because it has been derived from the rules.

Example 9

- Is r a valid REGEX?
- $r = (a + b +) . c$

- No, it cannot be derived by application of the 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:

   $r_1 + r_2$

   $r_1 . r_2$

   r_1^*

   (r_1)
3. A string is REGEX iff 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)

$= \{ba\}$ (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 $r = a + b$
- $L(r) = ?$

Solution

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

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

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

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

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

$= \{a, ba\}$ (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

$L(r) = L[(a + b)^*]$

$= [L(a + b)]^*$ (rule #7)

$= [L(a) \cup L(b)]^*$ (rule #4)

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

$= \{w : w \in \Sigma^*\}$ (any string over Σ)

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

Solution

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