San José State University
Department of Computer Science

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

Regular Expressions

(Part 2)

Lecture 19

Day 22/31

CS 154

Formal Languages and Computability

Spring 2018

Agenda of Day 22

- Collecting Quiz 7
- About Midterm 2
- Solution and Feedback of Quiz 6
- Summary of Lecture 18
- Lecture 18: Teaching ...
  - Regular Expression (Part 2)

About Midterm 2

Reminder 2

- Midterm #2 (aka Quiz++)
  - Date: Thursday 04/12
  - Value: 15%
  - Topics: Everything covered from the beginning of the semester
  - Type: Closed $y \in$ Material

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

- The cutoff for midterm #2 is the end of this lecture.

Solution and Feedback of Quiz 6 (Out of 25)

Metrics	Section 1	Section 2	Section 3
Average	23	20	22
High Score	25	25	25
Low Score	19	7	6

Summary of Lecture 18: We learned ...

Nondeterministic TMs

- There are two possible violations in standard TMs:
  - λ-transition
  - When δ is multifunction
- If we put λ in the condition place, we make a λ-transition.

- In practice, the following λ-transition is used:

Formal Definition

- A nondeterministic TM M is defined by the septuple:

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

$$\delta: Q \times (\Gamma \cup \{\lambda\}) \rightarrow 2^{Q \times (\Gamma \cup \{\lambda\}) \times \{L, R, S\}}$$

δ might be total xor partial.

- 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?

Summary of Lecture 18: We learned ...

Regular Expressions (REGEXs)

- REGEXs are another way to represent formal languages.
- We like REGEXs because ...
  - ... they represent formal languages in a more compact way.
  - They are shorthand for some formal languages.
  - They have practical applications in OS's and programming languages.
- We don't have a standard REGEX.
- This course introduces the mathematical base of them.

- The elements of REGEXs are:
  - ϕ, λ, Σ
  - ()
  - Operators:
    + (union)
    . (dot or concatenation)
    * (star-closure)
- Every REGEX represents a language.
- Does every language have a REGEX?
  - We don't know yet but we'll shortly!
- Associated language to a REGEX is ...
  - ... the language that it represents.

Any Question?

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 iff it can be derived from the primitive REGEXs by a finite number of applications of the rule #2.

REGEXs Validation

Example 9

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

Example 10

- 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 is representing?

- 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

Example 11

- 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 12

- 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 13

- 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)

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

REGEX → Language Examples

Example 14

- 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 15

- 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 16

- 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 17

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

Solution

REGEX → Language Examples

Example 18

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

Solution

Homework

- Find a REGEX for the following languages.
  1. $L(r) = \{w \in \{a, b\}^* : w \text{ contains no } a\}$
  2. $L(r) = \{w \in \{a, b\}^* : w \text{ contains exactly two } a\text{'s}\}$
  3. $L(r) = \{a^{2n} : n \geq 0\}$ over $\Sigma = \{a\}$
  4. $L(r) = \{a^{2n+1} : n \geq 0\}$ over $\Sigma = \{a\}$

REGEX → Language Examples

Example 19

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

Solution

REGEX → Language Examples

Example 20

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

Solution

REGEX → Language Examples

Example 21

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

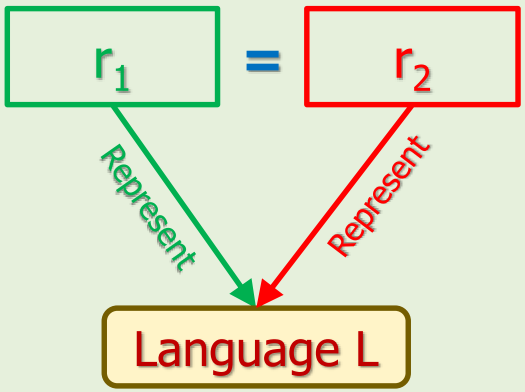
Solution

Equivalency of REGEXs

Definition

- Two regular expressions r_1 and r_2 are equivalent iff both represent the same language.

$$r_1 = r_2 \leftrightarrow L(r_1) = L(r_2)$$



Equivalency of REGEXs

Example 22

- 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 REGEX we can make?
  - Infinite

Identities

Identities

- If r, s, and t are REGEXs, and a, b ∈ Σ, 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 before to prove the above identities.
- Obviously, we should show both sides represent the same language.
- For example, for the first one, we should show:

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

Identities Examples

Example 23

$a\, b^* + bb^*$

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

Example 24

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

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

Example 25

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

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

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

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

Homework

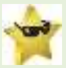
- Given $r = (aa)^* (\lambda + ab) (bb)^*$
- $L(r) = ?$

Homework

- Find a REGEX for the following languages.
  1. $L(r) = \{w \in \{a, b\}^* : w$ contains at least two a's$\}$
  2. $L(r) = \{w \in \{a, b\}^* : w$ begins with an 'a' and ends with a 'b'$\}$
  3. $L(r) = \{w \in \{a, b\}^* : w$ begins and ends with the same symbol$\}$

More Complex Languages

Example 26



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

- Struggling?
- Let's forget about this, and take another example!

Example 27

- L = $\{ww^R : w \in \Sigma^*\}$ over $\Sigma = \{a, b\}$
- r = ?
- After some struggling, we realize that we cannot represent such languages by REGEXs! Why?

REGEXs and Regular Languages

- The following theorem shows that REGEXs are another way to represent regular languages.

Theorem

- A language is regular iff at least one REGEX represents it.

Set of Languages Described by REGEXs	=	Set of Regular Languages

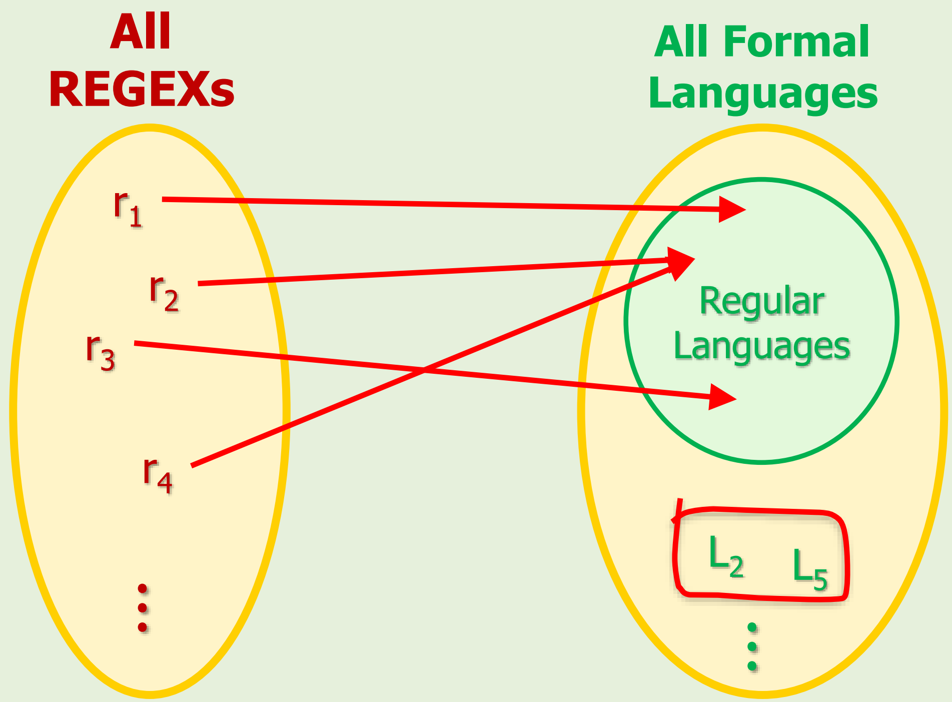
REGEXs and Languages Association

Revisited

- We've already known that "every REGEX represents a language".
- Now we know that:

  Those languages are regular.

  And there is no association between non-regular Languages and REGEXs.



What is the Next Step?

- We started this topic to look for a compact way to represent formal languages.
- We introduced REGEXs and experienced their usefulness.
- But the theorem showed their limitations.
  - REGEXs are just for regular languages.

- So, the next step would be looking for ...
  a practical compact way to represent non-regular languages.

Homework

- Fill out the following tables.
- For example, ϕ + a = ϕ + a = a or a . a = aa
  – Note that '+' and '.' are binary operations and need two operands but '*' is unary operation and needs one operand.

+	∅	λ	a
∅			a
λ			
a	a		

.	∅	λ	a
∅			
λ			
a			aa

∅*	
λ*	
a*	a*

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