San José State University
Department of Computer Science

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

Other Models of TMs

Lecture 18 -1
Day 21/31

CS 154
Formal Languages and Computability
Spring 2018

Agenda of Day 21

- About Midterm 2
- Summary of Lecture 17
- Quiz 7
- Lecture 18: Teaching ...
  - Other Models of TMs (Part 2)
  - Regular Expression (Part 1)

About Midterm 2

Reminder 1

- 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.

Summary of Lecture 17: We learned ...

TMs as Transducer

- Transducer is a device that converts an "input" to an "output".
- We model a transducer by a ...
  - ... function.
- TMs can work in transducers mode.
  - Input is all or part of the nonblank symbols on the tape at the initial time.
  - Output is all or part of the tape's content when the machine halts.
- We learned how JFLAP shows the output.

- A function is called "Turing-computable" if ...
  - ... there exists a Turing machine that implements it.
- We learned how to break a complex problem into smaller ones and how to combine TMs to make a bigger one.

Any Question

Summary of Lecture 17: We learned ...

Other Models of TMs

- We tried to figure out whether we can get more power by adding some capabilities to standard TMs.
- With any changes in standard TMs, we created a new class of automata.
- The changes we made:
  - TMs with stay-option ...
  - TMs with multidimensional-tape ...
  - TMs with multi-tape ...
- Were the new classes more powerful than the standard TM?

- We mentioned several theorems stating that the new classes were equivalent to standard TMs.

Any Question?

NAME	Alan M. Turing		
SUBJECT	CS 154	TEST NO.	7
DATE	04/05/2018	PERIOD	1 , 2 , 3

TEST RECORD	
PART 1	123
PART 2	
TOTAL	

Your list # goes here!

Quiz 7

No Scantron

Take-Home Exam

Nondeterministic TMs

Determinism in Standard TMs

1. Determinism = during any timeframe, there is no more than one transition.
2. Any violation of determinism, will make a machine nondeterministic.

Recap

- What could be those violations in standard TMs?
  - λ-transition
  - When δ is multifunction
- Let's deal with each one in detail!

λ-Transitions

1. λ-transition in automata theory means:

   The machine may "unconditionally" transit.

2. Therefore, if we put λ in the condition places, we make a λ-transition.

Recap

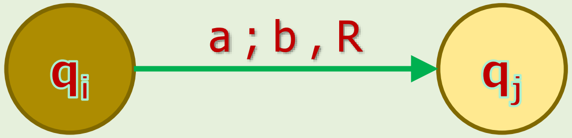
- This is our knowledge so far:

Automata Class	Transition Condition
DFA/NFA	Input Symbol
NPDA	Input Symbol + Top of stack
TM	Input Symbol

λ-Transitions

- For example, in the following transition, condition for transition is:

input symbol = 'a'



- So, if we put λ in the condition place, we make a λ-transition.

λ-Transitions

Definition

- For TMs, a transition is called λ-transition iff input part of the label is λ.

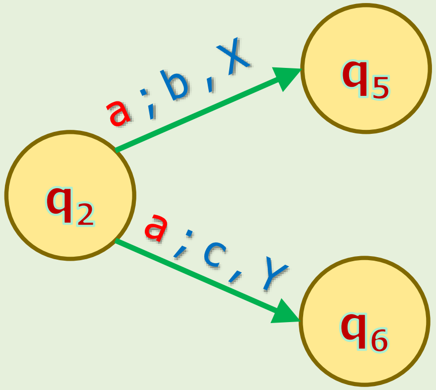
- But in practice, the following λ-transition is used:

- The machine does not need to do anything if it jumps to q_j.

Nondeterministic TMs: Multifunction Examples

Example 4

- Two or more transitions with the same input symbol



Nondeterministic TMs

Formal Definition

- A nondeterministic TM M is defined by the septuple:

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

- Where:
  - ... (same as standard TM elements)

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

δ might be total xor partial.

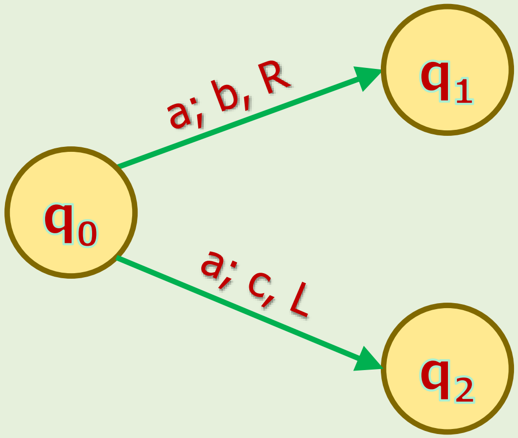
Nondeterministic TMs Transition Function Example

Example 5

- Draw the transition graph of the following sub-rule:

$$\delta\ (q_0, a) = \{(q_1, b, R), (q_2, c, L)\}$$

Solution



Nondeterministic TMs vs Standard TMs

Nondeterministic TMs vs Standard TMs

Theorem

- The class of nondeterministic TMs is equivalent to the class of standard TMs.
- We need to prove two things:
  1. Nondeterministic TMs simulate standard TMs.
  2. Standard TMs simulate nondeterministic TMs.

Proof of 1

- Let's assume we've constructed a standard TM for an arbitrary language L.
- Can we always construct a nondeterministic TM for L? How?
- Yes, just use a similar algorithm we used for DFAs and NFAs.

Nondeterministic TMs vs Standard TMs

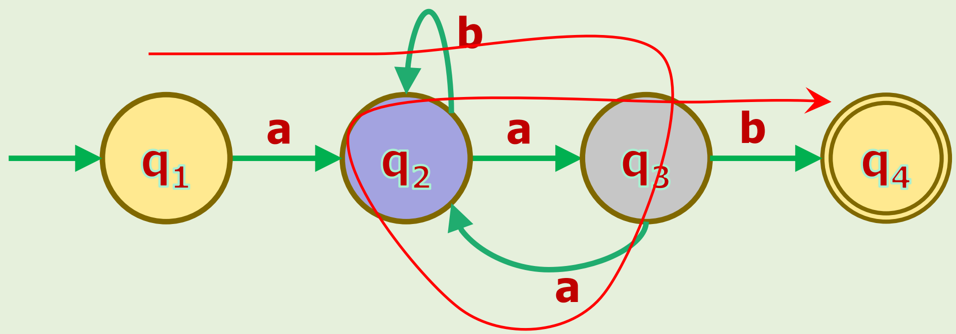
Proof of 2

- Mathematical proof of this part is not so easy but we can understand it intuitively.
- We'll explain it through an example.
- But first, we need some background.
- Next slide refreshes your knowledge about one-dimensional projection.

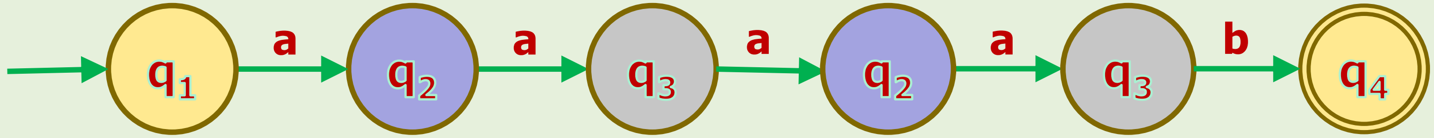
One-Dimensional Projection of a Walk

Recap

- As we learned before, we can represent a walk by one-dimensional projection.
- As an example, look at the string (walk) w = aaaab in the following NFA:

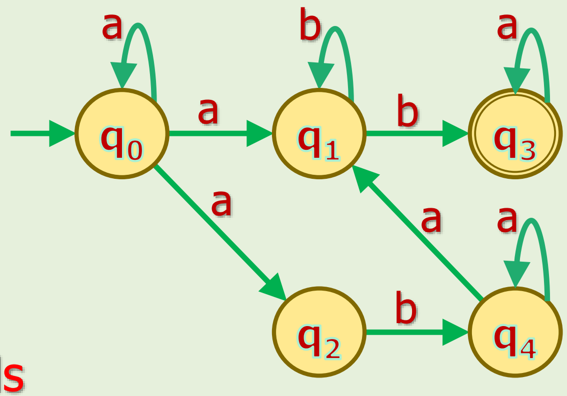


- This walk can be shown as:



Nondeterministic TMs vs Standard TMs

- **Proof of 2 (cont'd)**
- The following transition graph is an example of a nondeterministic TM.

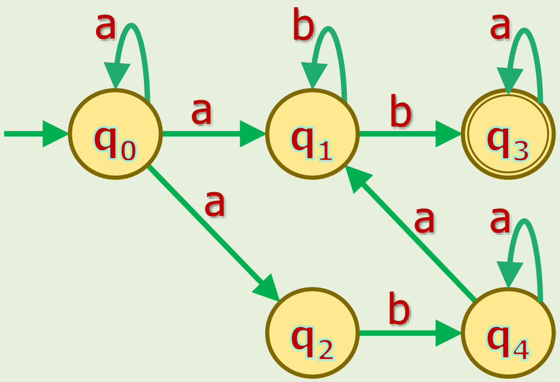


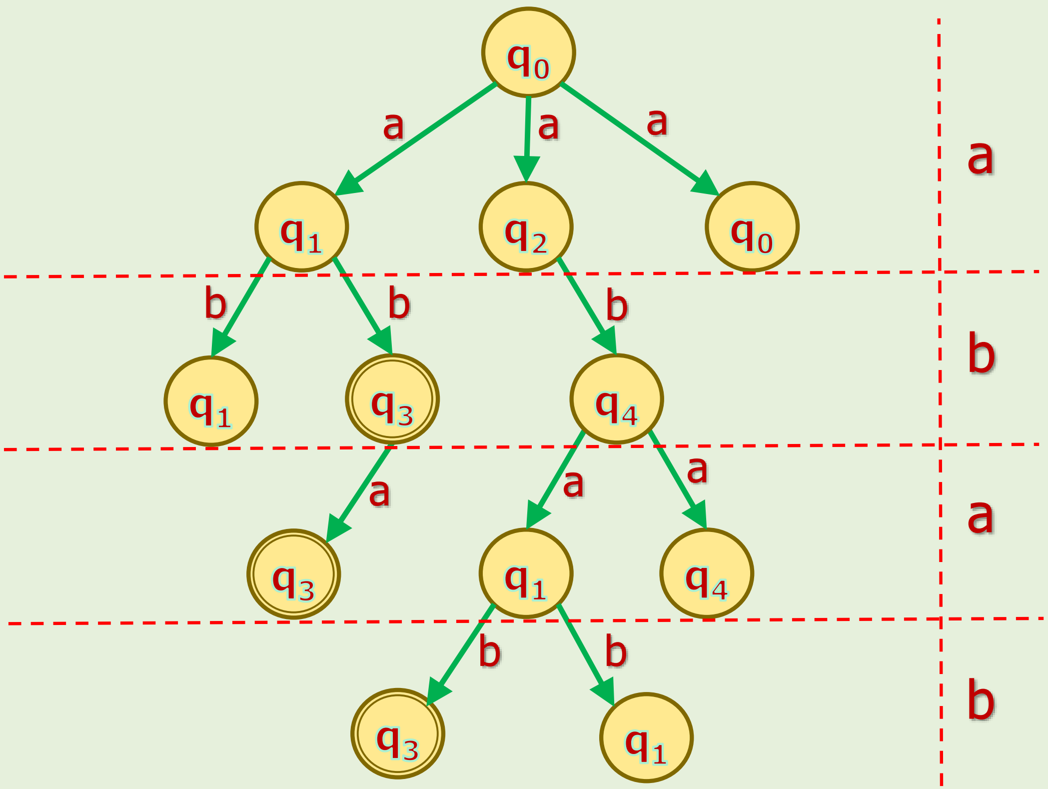
  - For simplicity, we showed only the input symbols of the labels.
  - It looks like an NFA, but we won't lose the generality of the point.

- If we input w = abab into this TM, overall 6 processes will be initiated.
- We usually prefer to organize them as a tree.

Nondeterministic TMs vs Standard TMs

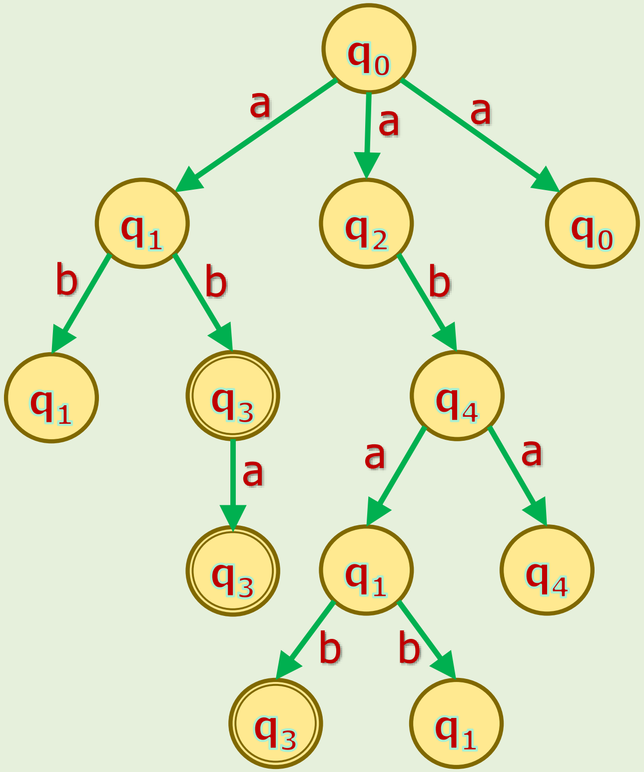
- **Proof of 2 (cont'd)**
- All processes for the string w = abab are:





Nondeterministic TMs vs Standard TMs

- **Proof of 2 (cont'd)**
- Every walk from q_0 to a leaf is a process that is a standard TM.
  - every leaf is either accepting or rejecting state.



- Therefore, here is the important point:

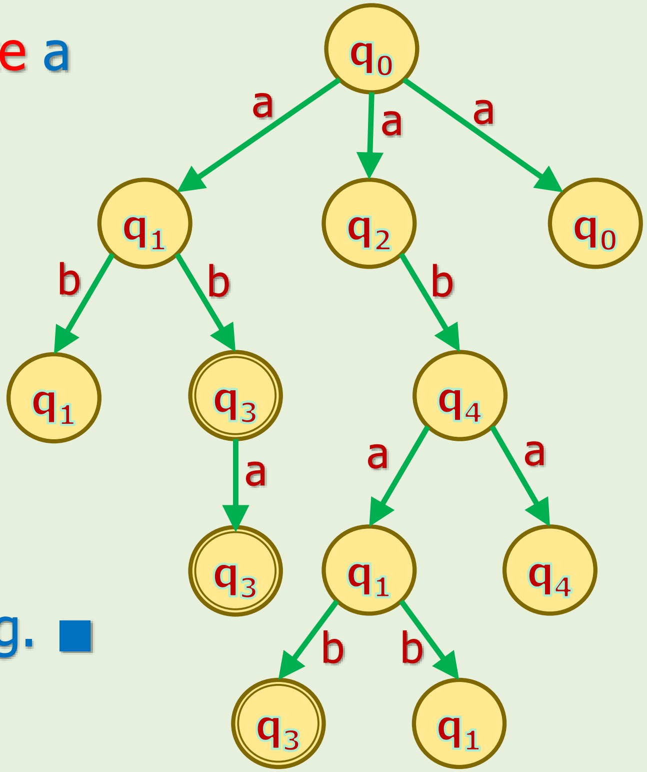
A nondeterministic TM
is a collection of standard TMs.

Nondeterministic TMs vs Standard TMs

Proof of 2 (cont'd)

- In fact, nondeterminism is a determinism backtracking algorithm.

- It means, a deterministic TM can simulate a nondeterministic TM if it can handle the bookkeeping of the backtracking.



- Your term project will show that standard TM can handle this bookkeeping. ■

Nondeterministic TMs: Notes

1. Nondeterminism does not add any power to the automata.
   - It just speeds up the computation.
2. We are always looking for more power and speed is not our concern.
   - "Speed" will be a matter of concern when we will be talking about "complexity theory".
3. Quantum computing tries to implement nondeterminism!
   - It does NOT add any power to computing.

Basic Concepts of Computation

Definition of Algorithm

Definition

- An algorithm for a problem L (= language) is equivalent to design a TM that solves L (= accept the language).

- Therefore, we define the TM structure as the "algorithm" for solving that problem.

Definition of Program

- A sub-rule defines how a machine acts in one transition for a specific state.
- The transition function defines all possible transitions of the machine for all possible situations.

- What is the "program" of a TM?

Definition

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

San José State University
Department of Computer Science

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

Regular Expressions

(Part 1)

Lecture 18 -2

Day 21/31

CS 154

Formal Languages and Computability

Spring 2018

Regular Expressions

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 to represent 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?

Do We Have Standard REGEXs?

- In computer science, we do NOT have a standard REGEX!
- Every OS and every programming language has its own REGEX.

- Of course, there are some common alphabet and rules between all of them.
  - So, you should learn each one based on their alphabet and rules.

- But the basic idea is the same.
  - In fact, they have implemented their REGEXs based on the REGEX we'll introduce in this course.

Regular Expressions (REGEXs)

REGEXs Elements

- REGEXs like everything else in this course, has a mathematical base.
- We need to introduce REGEXs'
  1. Elements
  2. Rules (Formal Definition).

- REGEXs have three elements:
  1. ϕ, λ, and the symbols of alphabet Σ (e.g. a, b, c)

     ϕ and λ has special usage that will be covered shortly.
  2. ( )
  3. Operators:

     \+ (union)

     . (dot or concatenation)

     * (star-closure)

REGEXs Examples

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

Example 1

- Given $L = \{a\}$ over $\Sigma = \{a, b\}$
- 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!
  – Infinite languages!

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 Σ = {a, b}
- r = ?

Solution

- L = {ab, bb, ba} = {ab} U {bb} U {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 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

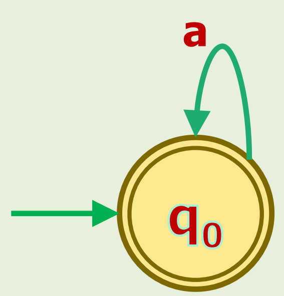
Set builder

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

Roster Method

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

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 lowest would be:
  - Parentheses
  - Star-closure
  - Concatenation
  - 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: $r = ab^* + c$

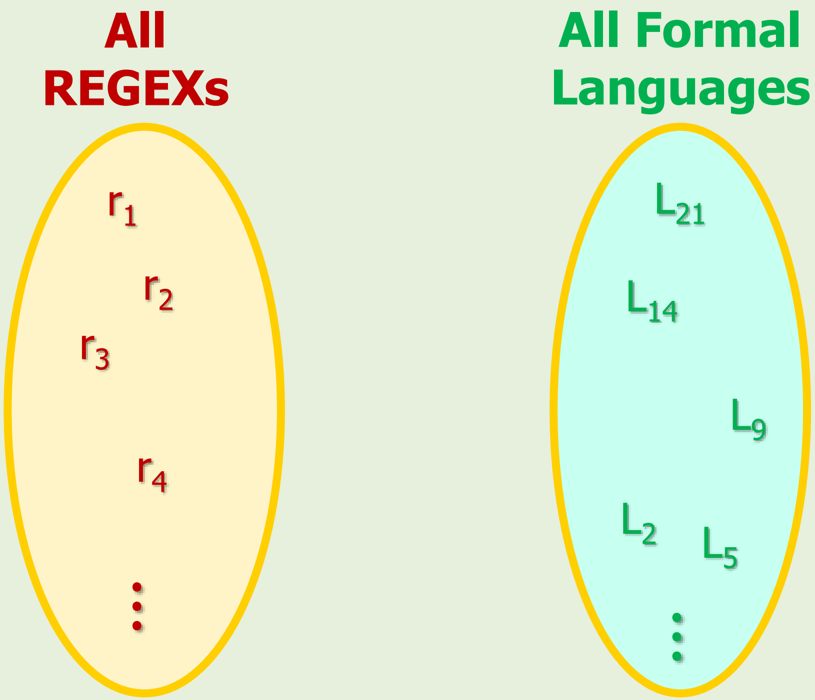
REGEXs and Languages Association

REGEXs and Languages Association

- What is the relationship between:

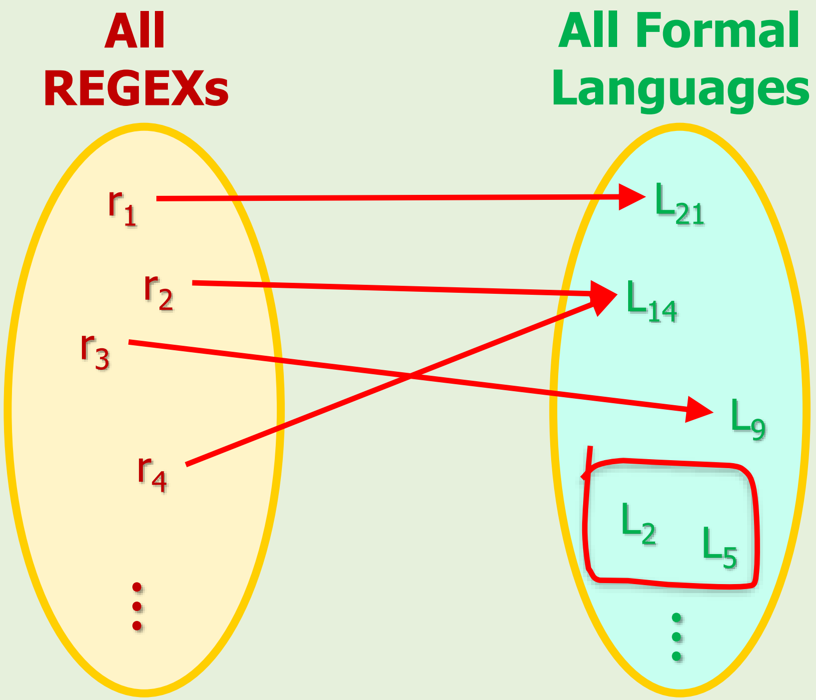
  the set of REGEXs, and

  the set of all formal languages?



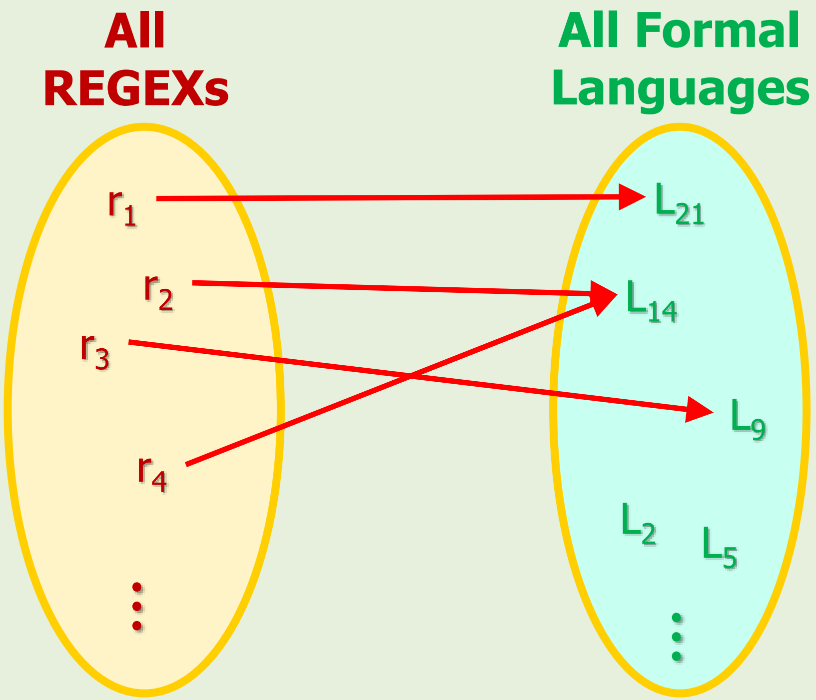
REGEXs and Languages Association

- We know that "every REGEX represents a language".
- BUT we don't know yet whether we can represent every language, by a REGEX or not!
  - Our knowledge is not enough yet.



REGEXs and Languages Association

- Can we consider this relationship as a function?
  - Yes, the definition of the function can be: $L : r \rightarrow L(r)$
- What type of function is this?
  - Total function!



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).

Example 8

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

- We saw before that a^* represented $L = \{a^n : n \geq 0\}$
- So, $L(r) = \{ba^n : n \geq 0\}$

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