Course: Automata Theory

Lecture 10: Alphabets and Formal Language Definition

Lecturer: Martha Gichuki

Course description

- The course begins with an introduction to logic and formal grammar where learners will do a recap on sets, logic and truth tables, sequences, relations and functions
- A coverage of finite state machines, Push Down automata and Turing Machines (The Church's thesis) will culminate the study of various models of computation.
- Formal language and grammar will then follow to enable learners differentiate regular and context free languages.
- An evaluation of the computability and complexity of practical computational problems which are the foundations of automata theory will then be done and the outcome will be problem description.

Learning outcomes:

Lecture 10: Alphabets and Formal Language Definition

At the end of the lecture the learner will be able to:

i. Describe alphabets and formal languages
ii. Build formal language from alphabets and symbols
iii. Describe various Regular operations

Alphabets & Languages

- A language is simply **a set of strings** involving symbols from some **alphabet**(s).
- This general definition allows languages to be random assortments of unrelated strings as well as coherent highly structured languages such as natural languages or high-level programming languages.

- Consider the case of high-level programming languages such as C or Pascal.
- The most reasonable way to think **of a language as a set of strings** is perhaps to take the **strings to be complete legal programs** written in that language.

Computer Program Compiling Process

- When we compile programs, we usually ask a **compiler** to check the **syntax of the programs or sub-programs rather than individual statements.**
- If a language incorporates the rules of syntax or grammar that characterize the language then the compiler reports **no errors else a list of errors** together with the lines affected is **reported**

- Therefore an **alphabet is a finite set of symbols**.
- In the case of a common language like English, the alphabet contains **26 letters**; **UPPER CASE and lower case as well as blanks and various punctuation symbols**.
- In the case of a programming language, **ten numeric digits** are added.

- If ∑ is an alphabet, a string over ∑ is simply **some number possibly zero, of elements of alphabets placed in order**.
- If ∑ = {a,b} is an alphabet, some of the strings over the alphabet are:- {a, aba, aabba }
- The **null string** denoted by **Λ** is the **string with no symbols**.
- If **x is a string** of an alphabet, the **length of x** is the **number of symbols in the string** and we denote this number by |X|.

- A language is simply a set of strings that are made up of **symbols** from **some alphabet**.
- When we talk of the language of a machine, we mean that the machine **recognizes** that particular language.

Languages are defined by Machines

- A Finite State Machine (FSM) **accepts or rejects strings.**
- Therefore a Finite State Machine **defines a language**.
- We say that machine M **recognizes language A if A contains all the strings that Machine M accepts.**

- Languages are sets of strings, therefore, new languages can be constructed using set operations.
- For **any two languages of an alphabet**, their **union, intersection and difference** are also the languages of the alphabet.

Regular Expressions:

- **Regular expressions** were designed to **represent regular languages with a mathematical tool**.
- This representation involves a combination of strings of symbols from some alphabet $\sum$.
- Recall that **a language is regular if some Finite Automaton recognizes it.**

Regular Operations on Regular Languages

- In the previous sections we introduced and defined finite automata and regular languages.
- We now begin to **investigate their properties**.
- This will help us **develop a tool box of techniques** to use **when designing automata to recognize particular languages**.

Regular Operations on Regular Languages….

- The toolbox also will include **ways of proving that certain other languages are non-regular** (i.e. beyond the capability of finite automata).
- In arithmetic, the **basic objects are numbers and the tools are operations** for manipulating them.
- e.g. the arithmetic expression: 2 + 6 = 8 (2 & 6 are objects, while + is the tool that manipulates the two objects).

Regular Operations on Regular Languages…

- In the theory of computation, the **objects are languages** and the **tools include operations** specifically designed for manipulating them.
- We define **three operations on languages called the regular operations** and we shall use them to **study properties of regular languages.**

Definition of a language M

- Let L(M) = be "Language L accepted by Machine M"
- ∀ w, w∈L (M) then M accepts w
- A language is **regular if some DFA recognizes it**. DFA recognizes regular languages.
- A regular language is one that can be constructed from the **big Three Operations** i.e.

i). Union

ii). Concatenation

iii). Star / Kleene Closure

Use of Regular expressions to design regular languages

- $L \subseteq \sum^*$ is regular iff (if and only if) $\exists$(***there exists***) a DFA M, | (***such that***) L= L(M).

- $L \subseteq \sum^*$ is regular iff $\exists$ | L= L(M)

- Regular expressions are used to design regular languages

Example 1: Regular Operations

- Let L = {a, b} be a regular language over $\sum$ = {a, b}.
- If both ***a*** and ***b*** are regular languages and they are expressions in L, then L is a regular language.
- Expressions are like the grammar of the language

Example 1: Regular Operations….

- Let A and B above be languages.
- We define the three regular operations

1) **union,**
2) **concatenation and**
3) **star** as follows: -

1. Union Operation

- The operation simply takes all the strings in both A and B and lumps them together into one language.
- $A \cup B = \{x | x \in A \text{ or } x \in B\}$

2. Concatenation Operation

- The operation is a little trickier.
- It **attaches a string from A in front of a string from B in all possible ways** to get the strings in the new language.
- A **o** B= {xy|x ∈A and y∈B}
- B **o** A= {yx|y ∈B and x∈A}

3. Star Operation

- The operation is a bit different from the other two because **it applies to a single language rather than two**.
- That is, the **star operation is a unary operation and not** a binary operation.
- It works by attaching any number of strings in A together to get a string in the new language.
    - $A^* = \{x_1x_2x_3\ldots x_k$ where $k \geq 0$ and $x_i \in A\}$
    - $B^* = \{y_1y_2y_3\ldots y_k$ where $k \geq 0$ and $y_i \in B\}$
- Because "any number" includes 0 as a possibility, the empty string $\wedge$ is always a member of A*, no matter what A is.

Example 2:

- Let the alphabet $\sum$ be the standard 26 letters {a, b, c…z}.
- If A= {good, bad} and B= {boy, girl}, then define the three operations for the language A i.e. union, concatenation and star

1. Union Operation

- A= {good, bad} and B= {boy, girl},
- The union operation lumps elements together
- A∪B = {good, bad, boy, girl}

2. Concatenation Operation

- Concatenation operation **attaches a string from A in front of a string from B in all possible ways** to get the strings in the new language.
- A **o** B= {xylx ∈A and y∈B}
- B **o** A= {yxly ∈B and x∈A}

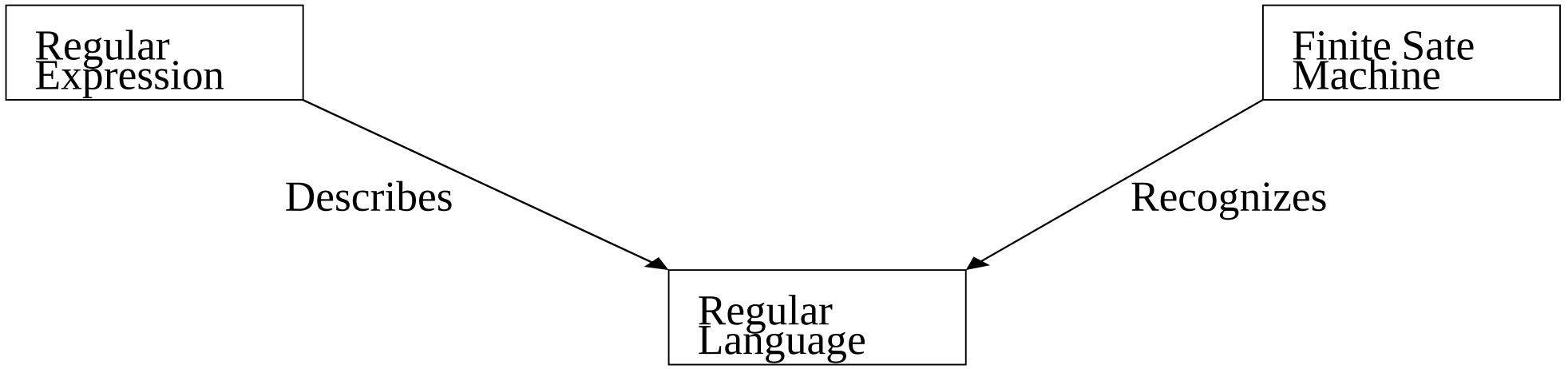
- A **o** B = {goodboy, goodgirl, badboy, badgirl}

3. Star Operation

- The Star operation attaches any number of strings(none included) in A together to get a string in the new language.
  - $A^* = \{x_1x_2x_3\ldots x_k$ where $k \geq 0$ and $x_i \in A\}$
  - $B^* = \{y_1y_2y_3\ldots y_k$ where $k \geq 0$ and $y_i \in B\}$
- A^* = {∧, good, bad, goodgood, goodbad, badgood, badbad, goodgoodgood, goodgoodbad, goodbadgood, goodbadbad…}
- **Question:** What is B^* =?

Equivalence with Finite Automata

- Regular expressions and finite automata are equivalent in their descriptive power.
- Any regular expression can be converted into a finite automaton that recognizes the language it describes and vice versa.
- **a language is regular if some Finite Automaton recognizes it.**
- A **language is regular if and only if some regular expression describes it.**



Proofing that a language is regular

- To proof that a language is regular, we need to construct a DFA for it.
- You can always construct some DFA M that accepts the same language as a given NFA N.
- Both of these automata can accept the same language.

Describing Languages

- We can use ***regular operations (union, concatenation and star) to build expressions that describe languages***.
- **Example: -** Given $(0 \cup 1)^* = \{\{0\}, \{1\}\}$, we can perform the following operations on this language as follows:

i). **Concatenating** we get $\{\in, 0, 1, 01, 10, 11, \ldots\}$ describes a certain NFA

ii). $\Sigma^*\mathbf{1}$ – Defines all strings **ending with a 1**

iii). $\mathbf{0}\Sigma^*$ - Defines all strings **beginning with a 0**

iv). $\{\Sigma^* \mathbf{1} \cup \mathbf{0}\Sigma^*\}$ – Defines a **union** of the above two

References

- Rowan G. & John T., (2009), *Discrete Mathematics: Proofs, Structures and Applications*, CRC Press, ISBN: 9781439812808.
- W. D. Wallis (2003), *A Beginners Guide to Discrete Mathematics*, Springer Science & Business Media, ISBN: 978-0817642693.
- Introduction to the theory of computation (3rd ed.), Michael, S. Boston, Cengage Learning. ISBN-13: 978-1133187790, (2012).
- Introduction to languages and the theory of computation (3rd ed.), Martin, J., New York: McGraw-Hill. ISBN-13: 978-0072322002, (2002)