Course: Automata Theory

Lecture 11: Regular Languages and Context Free Languages

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 11: Regular Languages and Context Free Languages

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

- Describe Regular and Context Free Languages
- Derive languages using Parse Trees
- Formally describe Context Free Languages

Context Free Grammar (CFG)

- Recall that Push down Stack Machines (PDA) recognize Context Free Languages and these are **built upon** Context Free Grammar.
- In a natural language like English, understanding a sentence begins with understanding the grammatical structure, meaning that one has to know how it is **derived from the grammar rules of the language.**

Formal Definition of a Context Free Grammar (CFG)

- A Context Free Grammar is a **4 Tuple** {**V, T, S, P**}, where: -
- **1) V** is a finite set of *Variables*
- 2) **T** is a finite set of *Terminals*
- 3) **S** is the *Start Variable*
- **4) P** is a finite set of *production rules*

Context Free Grammar (CFG)

- Suppose we have a Context Free Grammar e.g. the one specifying the syntax of a programming language.
- If a string is derivable from this grammar it is often useful to know a **derivation** because that is what allows us to interpret the strings correctly.
- A natural way to represent these derivations is by use of <u>derivation trees</u>.

Derivation Trees:

- A natural way of exhibiting any structure is by deriving it using a **derivation tree**.
- The root of the tree represents the variable with which we begin the derivation (**Start Variable**).
- Another name for a derivation tree is a **parse tree**.

Derivation Trees:

- Each *interior node* of the tree corresponds to the *variables* of the grammar
- The *children* of the tree correspond to the *symbols* in the entire string appearing on the right side of the production rules.

Derivation of Languages:

- This refers to the sequence of rules that produce the finished string of terminals from the start symbol. This is called **derivation or production**.
- The language of CFG is the set of terminal symbols which we can derive using specified production rules.
- The symbol → is used within the production rules to mean "Could take the value".

Example 1:

- Given the terminal a
- Non-terminal S
- Production rules S→aS, S→∧ where
 ∧ (empty string), the derivation for
 aaaa could be given as

- $S \rightarrow aaS$
- S →aaaS
- S →aaaaS
- S →aaaa∧→aaaa

Parse Tree

• The process of deriving a string is called as **derivation** and the geometrical representation of a derivation is called as a **parse tree** or **derivation tree**.



Leftmost Derivation-

- This the process of deriving a string by expanding the leftmost nonterminal at each step.
- The geometrical representation of leftmost derivation is called as a **leftmost derivation tree**.

Rightmost Derivation-

- This is the process of deriving a string by expanding the rightmost nonterminal at each step.
- The geometrical representation of rightmost derivation is called as a **rightmost derivation tree**.

Yield of a Parse Tree-

- Concatenating the leaves of a parse tree from the left produces a string of terminals.
- This string of terminals is called as **yield of a parse tree**.

Properties of a Parse Tree-

- *The root node* of a parse tree is the *start symbol* of the grammar.
- Each *leaf node* of a parse tree represents a *terminal symbol*.
- Each *interior node* of a parse tree represents a *non-terminal symbol*.
- Parse tree is independent of the order in which the productions are used during derivations.

Example 2:

- Given the terminal a;
- Non terminal S
- Productions: $S \rightarrow aS/Sa/a$
- Derive the Tree for the string "aa" **Solution**
- The word "aa" can be generated using two parse trees: -



Example 3:

- Consider a context free grammar that defines the language **pal of all palindromes** over the alphabet ∑ {a, b} using the following production rules:-
 - (i.) $(S \rightarrow \wedge)$ S could take the null value
 - (ii.) $(S \rightarrow a/b) S$ could take the value a or b
 - (iii.) $(S \rightarrow aSa/bSb) S$ could take the form aSa or bSb
- From these rules we can write S→aSa→abSba→ab∧ba →abba (which are all palindromes!)
- In an expression such as aSa or bSb, the two alternatives are aSa & bSb and not a & b directly.



Three steps are used to derive string **abba**.

- Rule three is used *twice*; and rule one; *once*, using the → notation.
- The rules can be written as: -
- S→a/b/∧
- S→aSa/bSb
- S→aSa→bSb→ab∧ba=abba
 - When we concatenate the **leaves** of any derivation tree, we get a string which is known as the yield of the tree. This tree has 4 yields.

Question:

Using the above production rules, derive the following strings: -S

- baab i.
- $(S \rightarrow \land)$
- $(S \rightarrow a/b)$
- (S \rightarrow aSa/bSb)

b b а а

Λ

ii. aaaa

- (S→∧) (S→a/b)
- (S→aSa/bSb)



iii. bbbb

- (S→∧)
- (S→a/b)
- (S→aSa/bSb)



iv. abaaba

- (S→∧) (S→a/b)
- (S→aSa/bSb)



<u>Example 4:</u>

- Given a CFG with the following definitions
- Terminals a,b
- Non-terminals: S, A
- Production Rules
 - S→AAA/AA
 - A \rightarrow AA/aA/Ab/a/b
- The string "abaaba" has the derivation tree:



а

Grammar Ambiguity

- For ambiguous grammars, Leftmost derivation and Rightmost derivation represents different parse trees.
- For unambiguous grammars, Leftmost derivation and Rightmost derivation represents the same parse tree.

- To check whether the given grammar is ambiguous or not we check whether we have two different parse trees for the same string derivation-
- Consider the string w generated by the given grammar-
- w = abba
 - i. $S \rightarrow SS$
 - ii. $S \rightarrow a$
 - iii. $S \rightarrow b$

• We realize that two different parse trees exist for string w, therefore the given grammar is ambiguous.



Parse tree-01

Parse tree-02

Example 5:

Given the string "a+a+a" with the following production rules: -

(i.) $S \rightarrow S+S$ (ii.) $S \rightarrow a$

1. Left-most derivation tree for some string

This is the process that looks at the string from Left to Right following the production rules provided.

The left most derivation for string "a+a+a" takes the format:

• $S \rightarrow S + S \rightarrow a + S \rightarrow a + S + S \rightarrow a + a + S \rightarrow a + a + a = a + (a + a)$



2. Right-most derivation for some string

This process looks at the string from Right to left. The right most derivation for string "a+a+a" takes the format: -

i. $S \rightarrow S + S$

ii. S→a

• $S \rightarrow S + S \rightarrow S + S \rightarrow a + S \rightarrow a + a + S \rightarrow a + a + a = (a+a)+a)$



We can exhibit this structure by using left - a+(a+a) and right - (a+a) + a derivation trees as follows: -



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)