## Computer Theory

Background

Automata Theory is a branch of theoretical computer science that deals with the study of abstract machines and their computational capabilities.

**Finite Automata (FA)** and **Pushdown Automata (PDA)** both types of automata, but they have key differences in terms of their computational power.

|  |  |
| --- | --- |
| **Finite Automata (FA)** | **Pushdown Automata (PDA)** |
| simplest form of automata | have an additional component called a stack.  A stack allows for recognition of patterns that involve nested structures |
| limited to recognizing regular languages | recognize context-free languages, which are a more powerful class of languages compared to regular languages. |
| finite set of states, an input alphabet, transition rules, an initial state, and a set of accepting (or final) states |  |
| limited in terms of memory; they can only recognize regular languages. | can accept languages that can be described by context-free grammars and are capable of recognizing nested structures, such as matching parentheses. |
| Regular languages can be described by regular expressions, and finite automata are equivalent in power to regular expressions. |  |

Pushdown Automata Theory

|  |  |
| --- | --- |
| Section 1 | Context-Free Grammars  Grammatical Format  Pushdown Automata  CFG = PDA |
| Section 2 | Non-Context-Free Languages  Context-Free Languages  Decidability |
| Section 3 | Turing Machines  Post Machines  Minsky's Theorem  Variations on t he TM |
| Section 4 | TM Languages  The Chomsky Hierarchy  Computers |

**Lesson 1**

Context-Free Grammars (CFG)

Context-Free Grammars (CFG) are a formal way of describing the syntax or structure of languages.

They play a crucial role in the analysis and design of programming languages, compilers, and various tools in computer science.

A context-free grammar, CFG, is a collection of three things: **T-N-P**

**Terminals:** An alphabet of letters that appear in the final strings of the language (words of a language)

*(designated by lowercase letters)*

**Non-terminals:** Symbols that can be replaced by other symbols according to the rules.

*(designated by capital letters)*

Start Symbol: for “start here”. Specifies where the generation of strings begins.

**Production Rules:** Define how non-terminals can be replaced by sequences of terminals and/or other non-terminals.

A finite set of productions of the form

One Nonterminal finite string of terminals

and/or

One Nonterminal choice of terminals

A language generated by a

CFG is called a context-free language, abbreviated **CFL.**

Example 1:

*Define production rules for the Non-terminal AE*

**AE rules:**

*Rule 1 Any number is in the set AE.*

*Rule 2 If x and y are in AE, then so are:*

**(x)**

**-(x)**

**(x + y)**

**(x - y)**

**(x \* y)**

**(x/y)**

**(x\*\*y)**

Example 2:

*Identify the Terminals, Non-terminals and production rules of AE*

# mathematica

# Terminal(s): An alphabet ∑/Set of letters

# +

# -

# \*

# /

# \*\*

# (

# )

# any-number

# Non-terminal(s): Are defined by a set of rules

# Start

# AE

# Production Rules:

# Define non-terminals using of terminals and/or non-terminals

Start -> (AE)

AE -> (AE + AE)

AE -> (AE - AE)

AE -> (AE \* AE)

AE -> (AE / AE)

AE -> (AE \*\* AE)

AE -> (AE)

AE -> - (AE)

AE -> any-number

Example 3:

*Use Example 2 with expansion on ANY-NUMBER*

We could also convert the terminal ANY-NUMBER to a Non-terminal using a set of rules:

# Terminal(s):

# O 1 2 3 4 5 6 7 8 9

# Non-terminal(s)

# ANY-NUMBER

# FIRST-DIGIT

# OTHER-DIGIT

# Production Rules:

Rule 1 ANY-NUMBER -> FIRST-DIGIT

Rule 2 FIRST-DIGIT -> FIRST-DIGIT OTHER-DIGIT

Rule 3 FIRST-DIGIT -> 1 2 3 4 5 6 7 8 9

Rule 4 OTHER-DIGIT -> O 1 2 3 4 5 6 7 8 9

Example 4:

*Use Example 2 + Example 3 to produce the number 1066*

# Terminal(s):

# O 1 2 3 4 5 6 7 8 9

# Non-terminal(s)

# ANY-NUMBER

# FIRST-DIGIT

# OTHER-DIGIT

# Production Rules:

Rule 1 ANY-NUMBER => FIRST-DIGIT

Rule 2 => FIRST-DIGIT OTHER-DIGIT

Rule 2 => FIRST-DIGIT OTHER-DIGIT OTHER-DIGIT

Rule 2 => FIRST-DIGIT OTHER-DIGIT OTHER-DIGIT OTHER-DIGIT

Rule 3 and 4 => 1066

A context-free grammar, CFG, is a collection of three things:

1 . An alphabet of letters called Terminals from which we are going to make strings that will be the words of a language.

2. A set of symbols called Non-terminals, one of which is the symbol S, standing for "start here."

3. A finite set of productions of the form

One Non-terminal finite string of Terminals

and/or

One Non-terminal choice of Terminals

or

One Non-terminal the empty string.

We require that at least one production

has the Non-terminal S as its left side.

Example

Let the only terminal be a and the productions be:

# Terminal(s):

# a

# Non-terminal(s)

# S

# Production Rules:

PROD 1 S -> aS

PROD 2 S ->

The null/empty string

“can be replaced by”

(The language generated by this CFG is exactly a\*)

If we apply production 1 six times and then apply production 2,

we generate the following:

S => aS

=> aaS

=> aaaS

=> aaaaS

=> aaaaaS

=> aaaaaaS

=> aaaaaaA

= aaaaaa

**Working strings**

("unfinished stages" are strings of terminals and non-terminals)

“can develop into”

ASS1

[Question 1]

Consider the following CFG:

Prove that this generates the language defined by the regular expression *a*∗*ba*

Given

Let the language generated by the CFG be

Let the language defined by the regular expression be

To show , we must prove that

|  |  |
| --- | --- |
| 1. | *Every string generated by the CFG is also in the language* |
| 2. | *Every string in the language can be generated by the CFG is also* |

*1.*

**Terminal(s):**

,

**Non-terminal(s)**

**Production Rules:**

PROD 1: *will generate words with arbitrary number of a’s*

*…*

PROD 2: *will generate just the word ba*

Therefore, any string generated by will be in the form

*2.*

can be generated with the production PROD 1:

can be generated with the production PROD 2:

Therefore, any string generated by can be generated by

Thus,

Syntax trees

Also known as parse trees or derivation trees, are graphical representations of the syntactic structure of a string generated by a context-free grammar (CFG)

|  |  |
| --- | --- |
| node | represents a symbol in the grammar |
| edges | represent the derivation steps from one symbol to another |

ASS1

[Question 3]

Investigate each of the CFGs provided and decide whether the word *abba* is generated by the given CFGs.

In the case where *abba* is not generated a brief discussion why a particular CFG does not generate *abba*.

If abba is indeed generated, then draw the corresponding syntax tree illustrating the generation of *abba*.

1. CFG 1:

b

a

S

S

b

a

|  |  |
| --- | --- |
| **Start Symbol (S)** | - root of the tree |
| **Derivation Steps** | - represent derivation step in the production rules of the CFG. |
| **Terminal Symbols** | ,  - represented as leaf nodes in the tree. |
| **Non-terminal Symbols** | - represented as internal nodes in the tree. |
| **Derivation Path** | - path from the root (start symbol) to the leaves (terminal symbols)  - represents the derivation path that generates the input string. |

PROD 1:

The first 'a' is generated by the production rule S -> aSb.

The second 'b' is generated by the production rule S -> aSb.

PROD 2:

The 'ab' in the middle is generated by the production rule S -> ab.

|  |  |  |
| --- | --- | --- |
| Production Rule |  |  |
| PROD 1: | a | Start with the start symbol S and apply the production rule S → aSb. |
| PROD 1: | b | Now, we're at a non-terminal symbol S.  Apply the production rule S → aSb. |
| PROD 1: | b | We still have the non-terminal symbol S.  Apply the production rule S → aSb again. |
| PROD 2: | a | Finally, we've exhausted the non-terminal symbols.  Now, apply the production rule S → ab. |