Tyler Cullen
2/11/2023
CSC 344 | Programming Languages
BNF Assignment 1

Abstract

This assignment is about Backus-Naur Form Grammar (BNF). During this assignment I was asked to compose six different BNF grammars for the different languages given. This also included drawing BNF parse trees and describing them in English in a straightforward, compelling manner. This assignment will expand my knowledge on BNF grammar and parse trees so I can further understand how languages and functions are described.

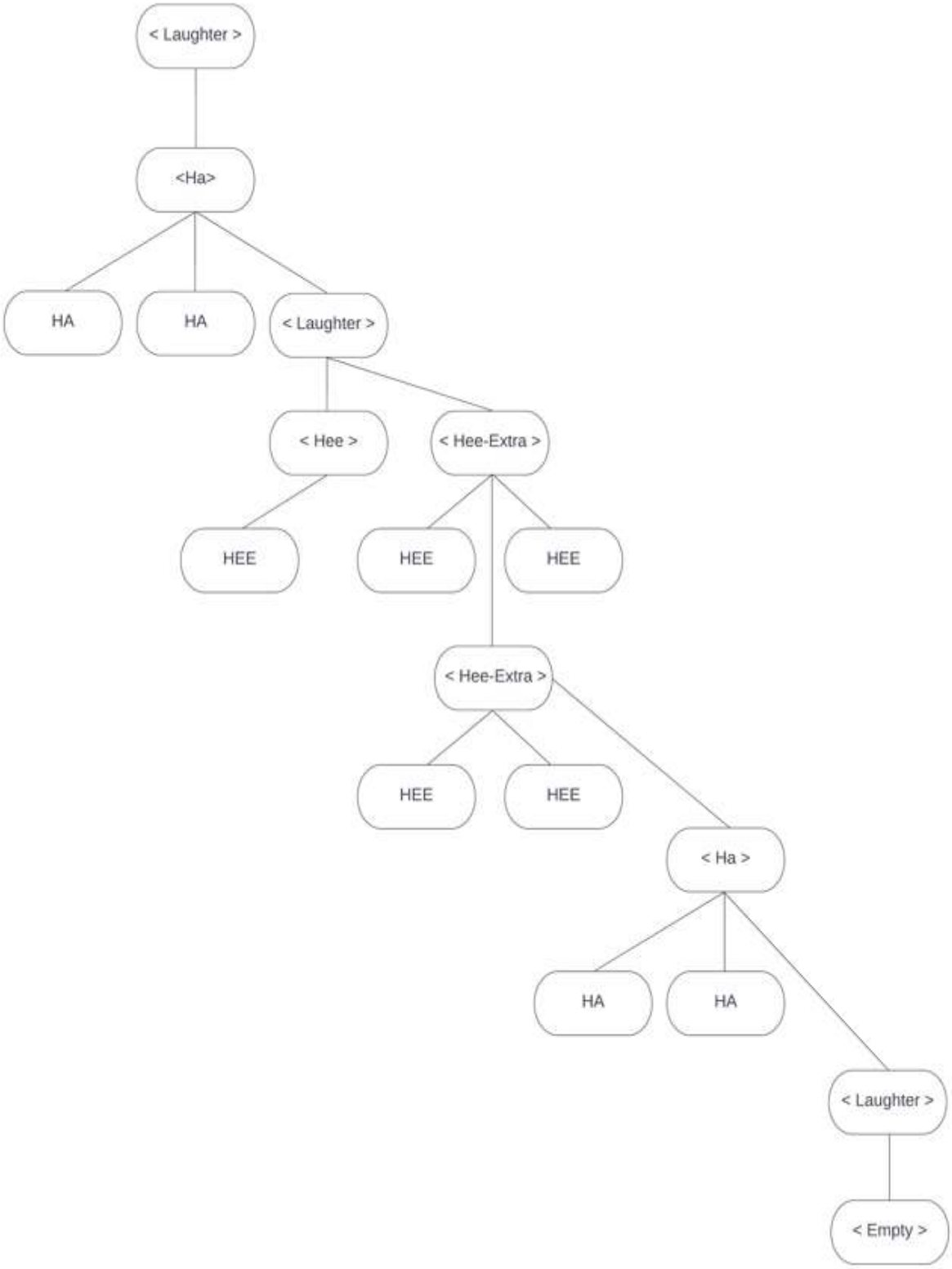
Problem 1 - Laughter

BNF Grammar:
Tokens: { HEE, HA }
Nonterminals = **{** Laughter, Ha, Hee, Hee-Extra }
Productions = The following set of rules:

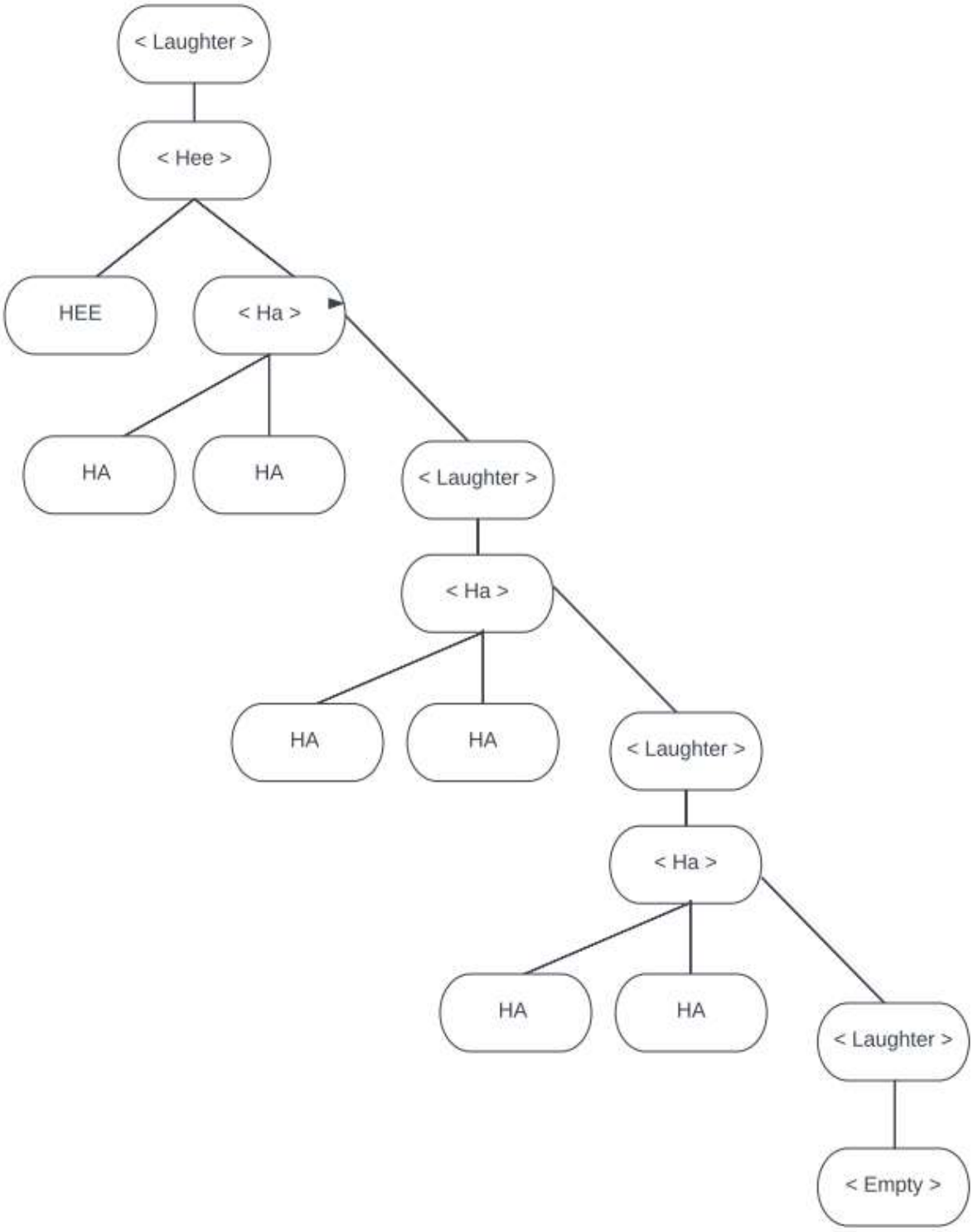
< **Laughter** > **::** = < Ha > | < Hee > | < Empty >
< **Ha** > **::** = HA HA <Laughter>
< **Hee** > **::** = HEE < Ha > | HEE HEE < Hee-Extra > | < Empty >
< **Hee-Extra** > **::** = Hee Hee < Ha > | < Laughter>

Start Symbol: < Laughter >

Parse Tree Problem - : HA HA HEE HEE HEE HEE HEE HA HA



Parse Tree - HEE HA HA HA HA HA HA



Problem 2 - Special Quaternary Numbers

Tokens: {0, 1, 2, 3}

Non-Terminals: { Quart_Num**,** Zero, No-Zero, One, No-One, Two, No-Two, Three, No-Three **}**

Productions = The following set of rules

Quart_Num : : = < Zero > | < One > | < Two > | < Three >

Zero :: = 0 | < No-Zero >

No-Zero :: = < One > | < Two > | < Three > | < Empty >

One :: = 1 | < No-One >

No-One :: = < Zero > | < Two > | < Three > | < Empty >

Two :: = 2 | < No-Two >

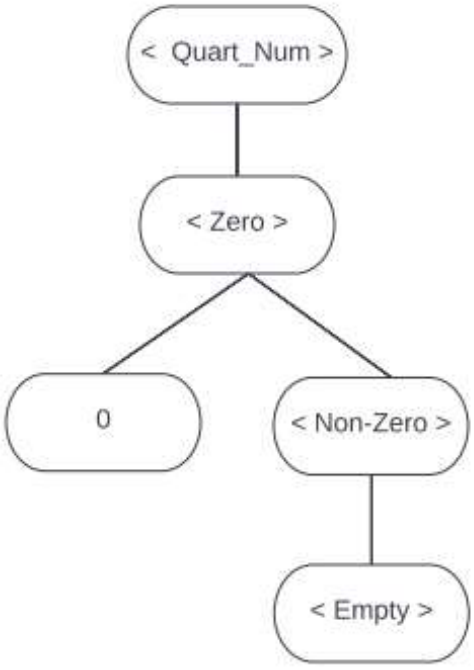
No-Two :: = < Zero > | < One > | < Three > | < Empty >

Three :: = 3 | < Non-Three >

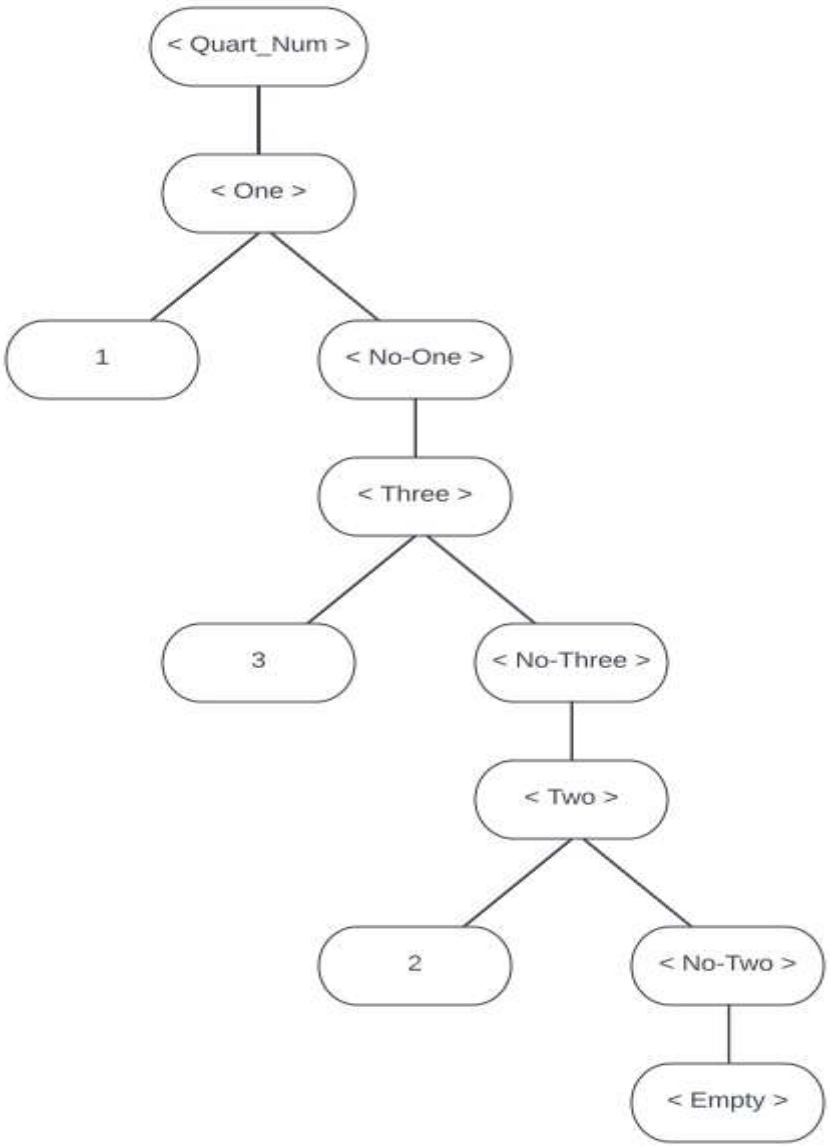
No-Three :: = < Zero > | < One > | < Two > | < Empty >

Starting Symbol: < Quart_Num >

Parse Tree - 0



Parse Tree - 132



Task 4 - Explain why 1223 is not in the language:

The reason 1223 is not in the language is because in the **production** set of rules for state **< Two >** it indicates the only other option is **< No-Two >** which does not allow for another 2 to be adjacent to each other. Therefore 1223 is not in the language.

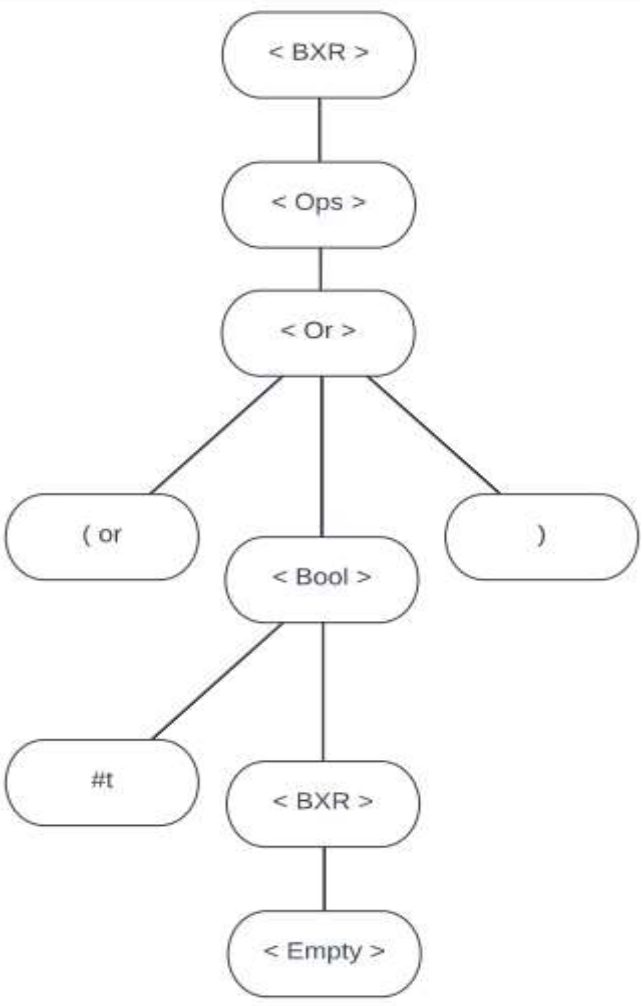
Problem 3 - BXR

Non-Terminals: { BXR, Ops, And, Or, Not, Bool **}**
Productions = The Following Set of Rules

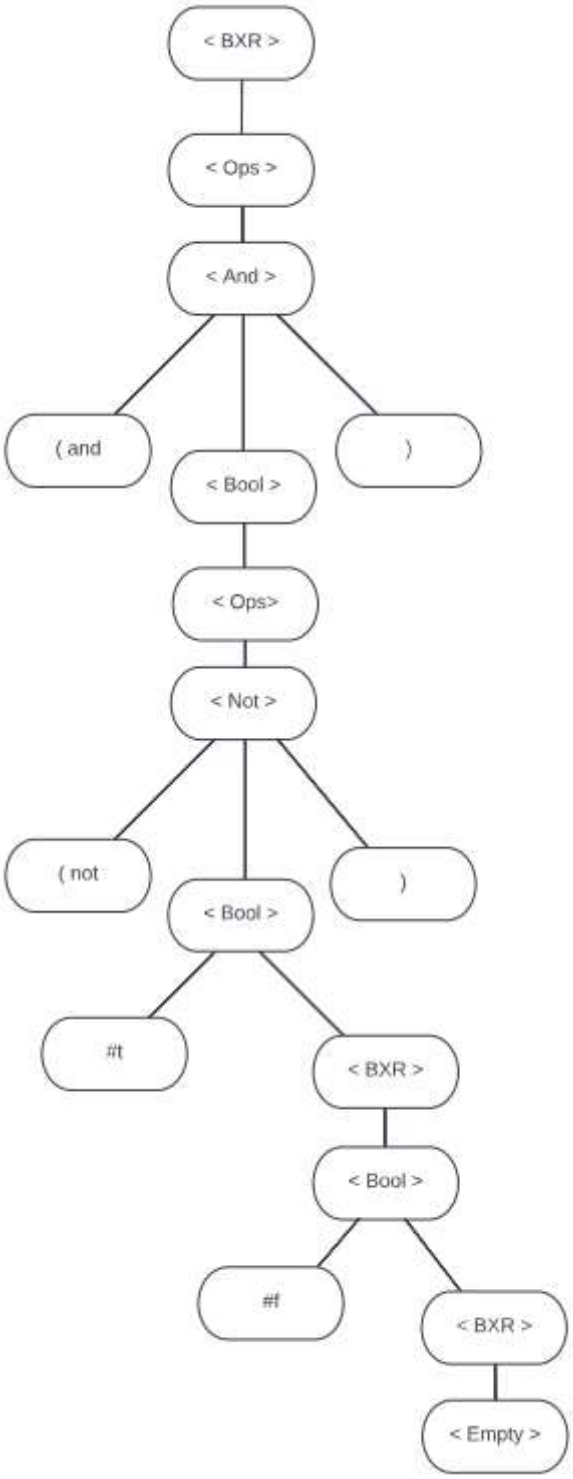
< **BXR** > **::=** < Ops > | < Bool > | Empty
< **Ops** > **::=** < And > | < Or > | < Not >
< **Not** >**::=** (not < Bool >)
< **And** >**::=** **(** and < Bool >) | (and)
< **Or** > **::=** (or < Bool >) | (or)
< **Bool** > **::=** #t | #f | < BXR > | < Ops >

Starting Symbol: < BRX >

Parse Tree - (or #t)



Parse Tree - (and (not #t) #f)



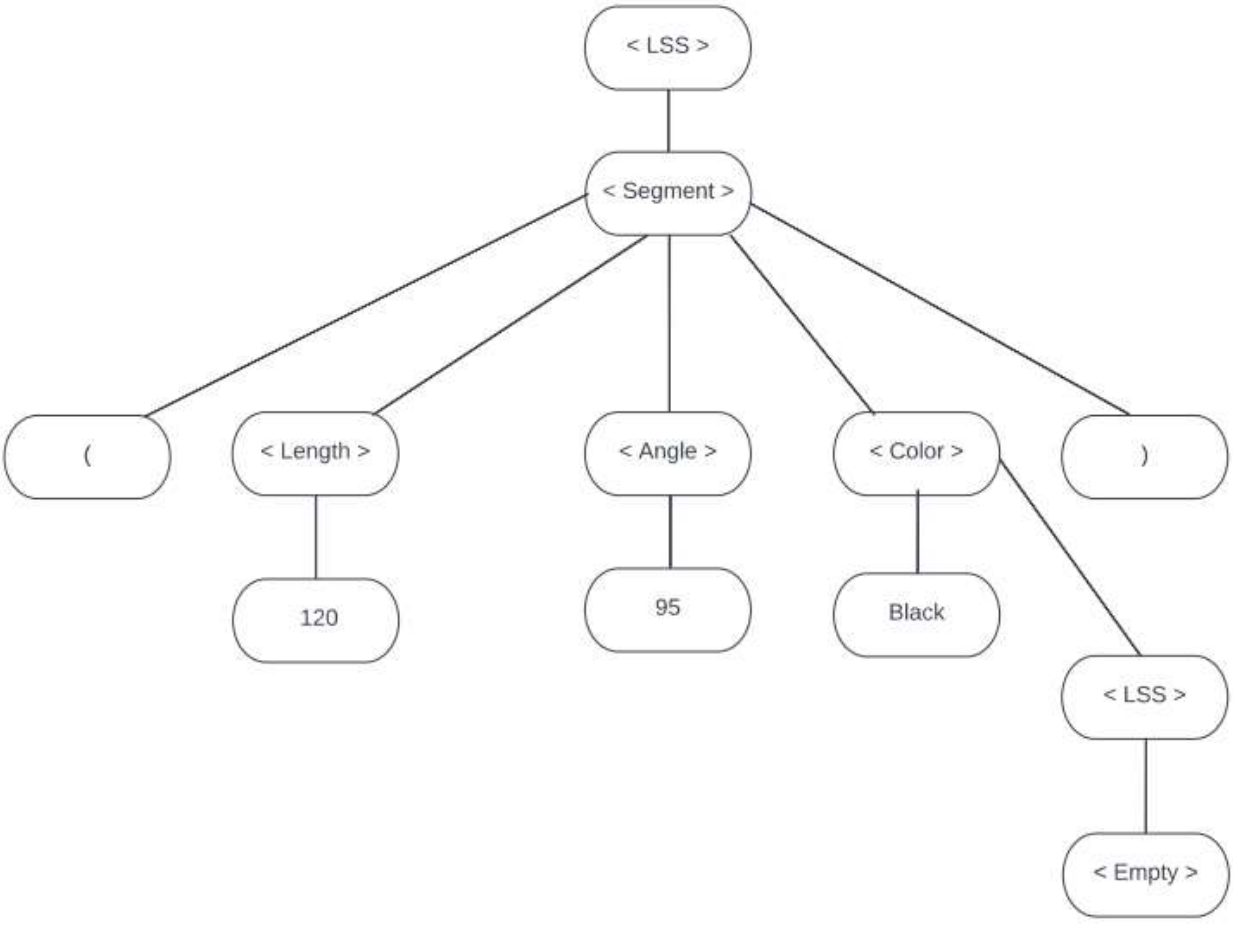
Problem 4 - LSS (Line Segment Sequences)

Non-Terminals: { LSS, Segment, Length, Angle, Color, Empty**}**
Productions = The Following Set of Rules

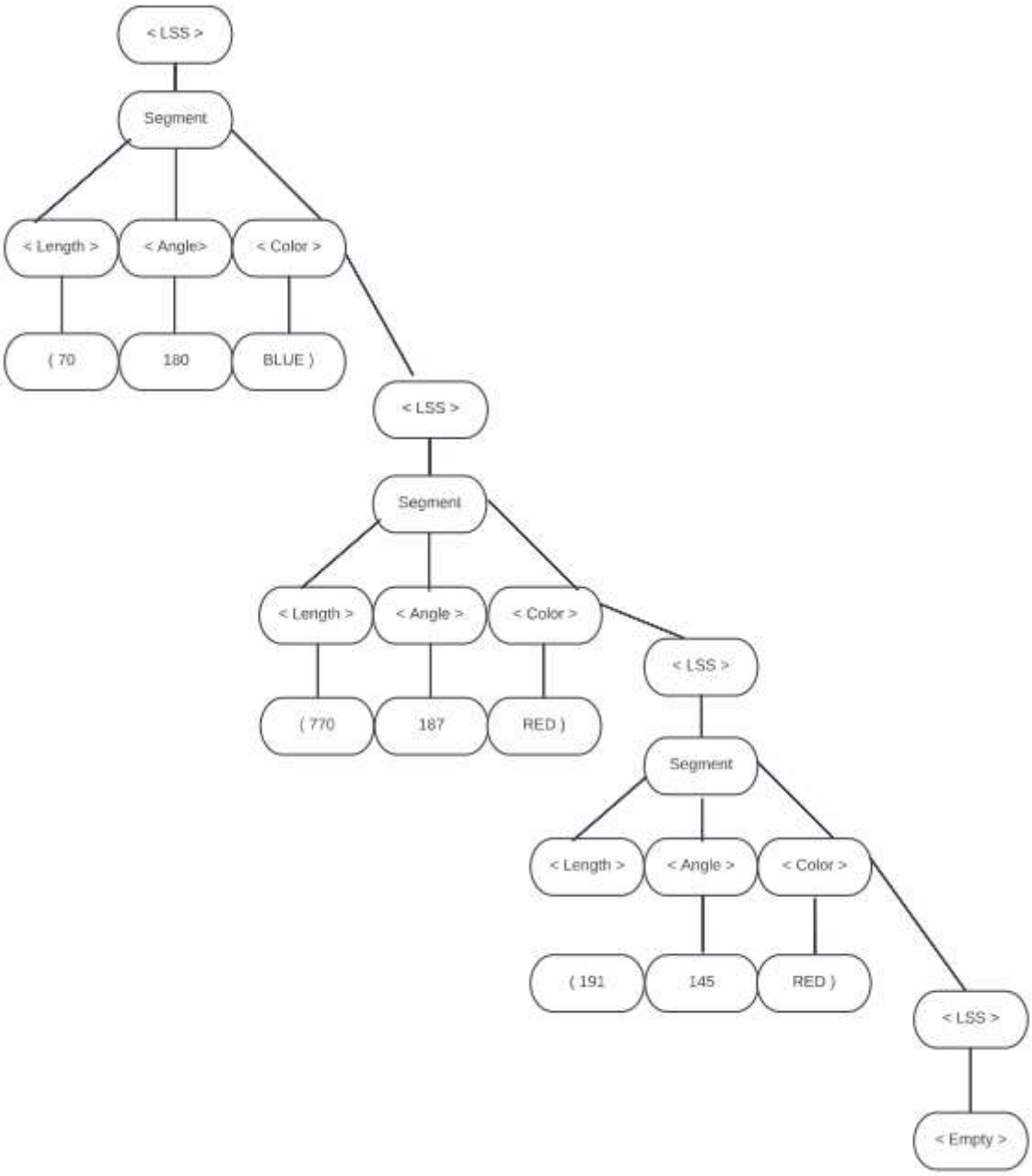
< **LSS** > **::=** < LSS >**|**< Segment > | < Empty >
< **Segment** > **::= (** < Length > < Angle > < Color >)
< **Length** > **::=** (int
< **Angle** > **::=** int
< **Color** > **::=** BLACK) | RED) | BLUE) | < LSS >

Starting Symbol: < LSS >

Parse Tree - (120 95 BLACK)



Parse Tree - (70 180 BLUE) (770 187 RED) (191 145 RED)



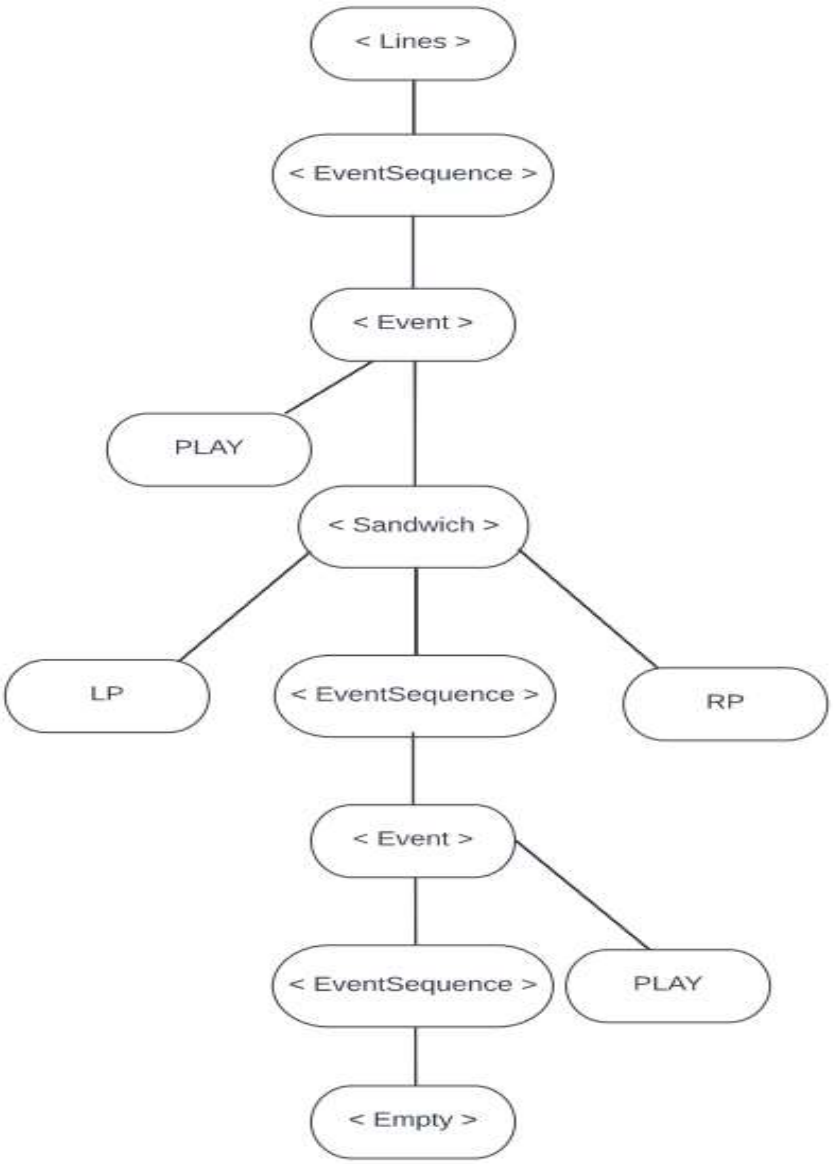
Problem 5 - M-Lines

Non-Terminals: { Lines, EvenSequence, Event, Sandwich **}**
Productions = The Following Set of Rules

< **Lines** > **::=** < EventSequence > | < Empty >
< **EventSequence**> **::=** < Event > | < Sequence > | < Event > < Sequence > |
<Sequence > < Event > | < Empty >
< **Event** > **::=** PLAY < EventSequence > | REST < EventSequence >
< **Sandwich** > **::=** RP < EventSequence > LP | LP < EventSequence > RP |
S2 < EventSequence > X2 | X2 < EventSequence > S2 |
S3 < EventSequence > X3 | X3 < EventSequence > S3

Starting Symbol: < Lines >

Parse Tree - LP PLAY RP PLAY



Problem 6 - BNF?

BNF Otherwise known as Backus-Naur form is a type of notation that is issued to describe the syntax of languages such as programming languages and instruction sets. It consists of a set of terminal symbols, set of non terminal symbols and a set of production rules of the form. These symbols are then used to describe the instructions or execution of a program often done using parse trees.