Sam Ghent
CSC344
BNF Assignment

Abstract: In this assignment I will be creating BNF grammers for some given languages, drawing parse trees, and explaining BNF in my own words.

---

(1) – Laughter

BNF Grammar (Task 1):

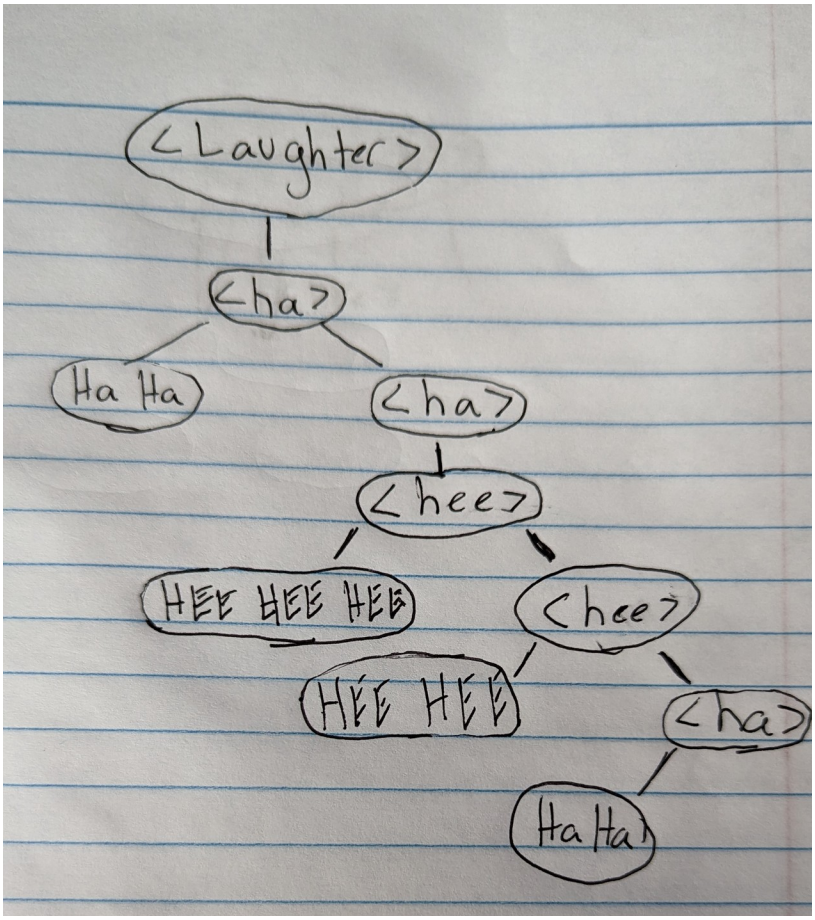
<Laughter> ::= <Ha> | <Hee> | <empty>

<Ha> ::= HA HA <Ha> | <Hee> | <empty>

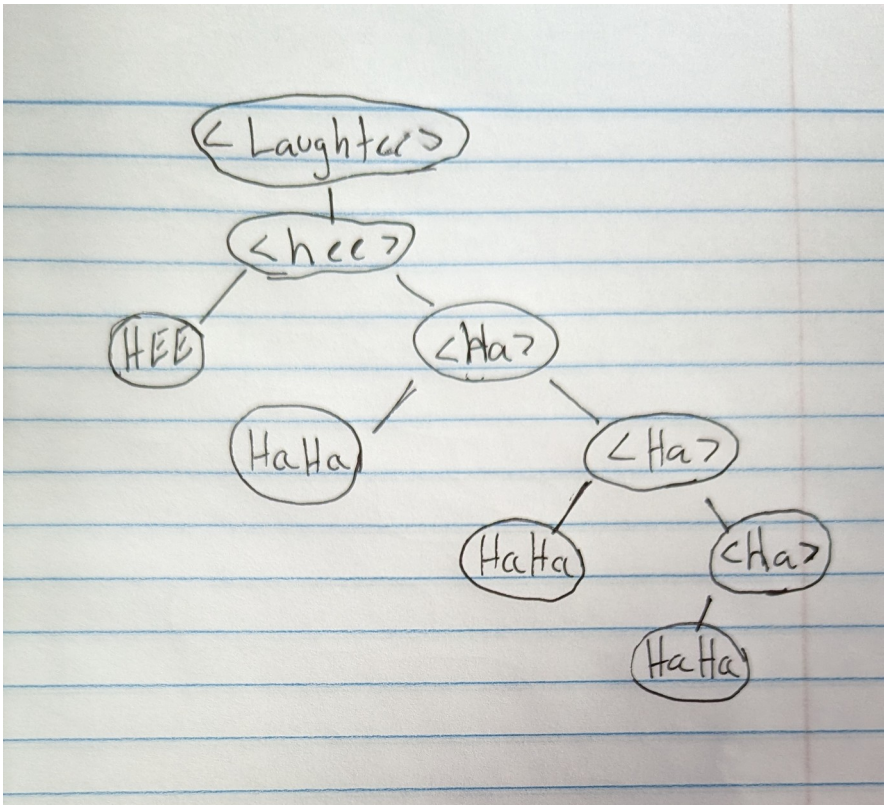
<Hee> ::= Hee | Hee <Ha> | Hee Hee <Hee>

<Reject> ::= <Ha> | <Hee>

Parse Tree (Task 2): HA HA HEE HEE HEE HEE HEE HA HA



Parse Tree (Task 3): HEE HA HA HA HA HA HA



---

(2) – Special Quaternary Numbers (SQN)

BNF Grammar (Task 1):

<SQN**> ::=** 0 | <Zero>

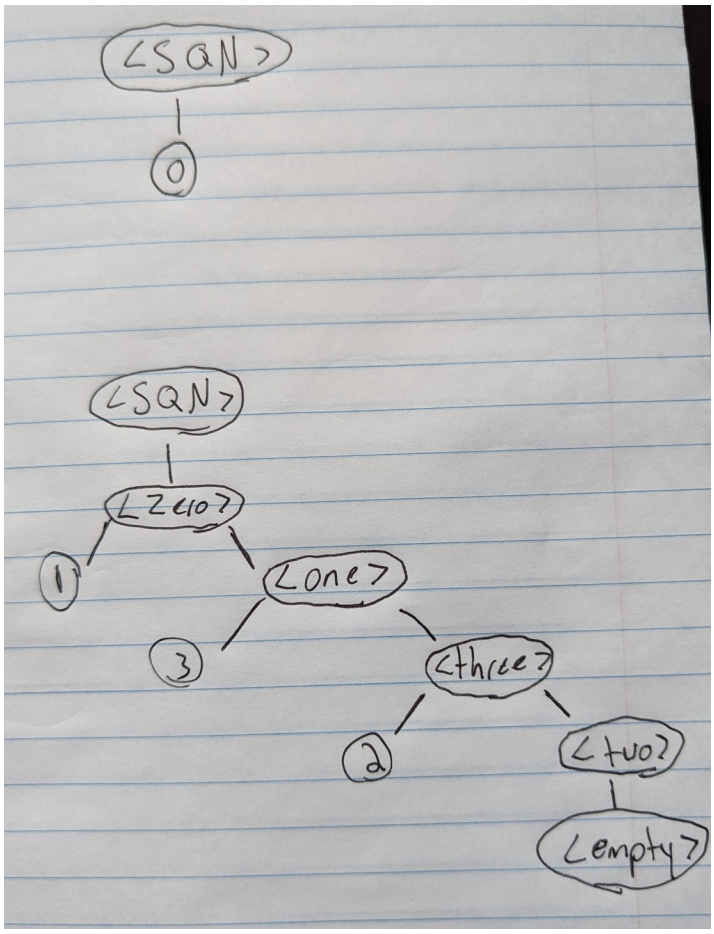
<zero> ::= 1 <one> | 2 <two> | 3 <three> | <empty>

<one> ::= 0 <zero> | 2 <two> | 3 <three> | <empty>

<two> ::= 0 <zero> | 1 <one> | 3 <three> | <empty>

<three> ::= 0 <zero> | 1 <one> | 2 <two> | <empty>

Parse Tree (Task 2 & Task 3):



Task 4: Explain, in precise terms, why you cannot draw a parse tree, consistent with the BNF grammar that you crafted, for the string: 1223

By examining our grammar we can see that after we hit that intial two we have three options. Either 0, 1, or 3. So the String 1223 simply can not happen because we cannot have a 2 after a 2 in this grammar.

---

(3) – BXR

BNF Grammar(Task 1):

<BXR> ::= <operation> | #t | #f

<operation> ::= <and> | <or> | <not>

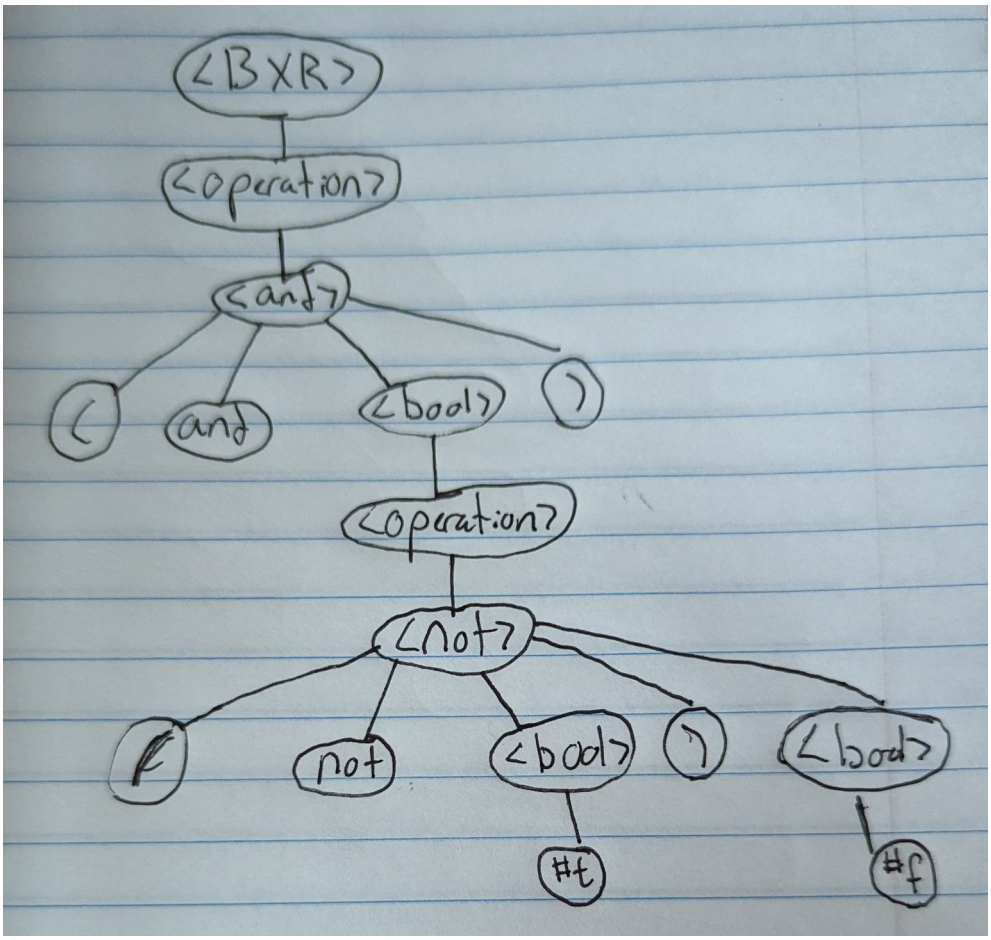
<and> ::= (and <bool>) | (and)

<or> ::= (or <bool>) | (or)

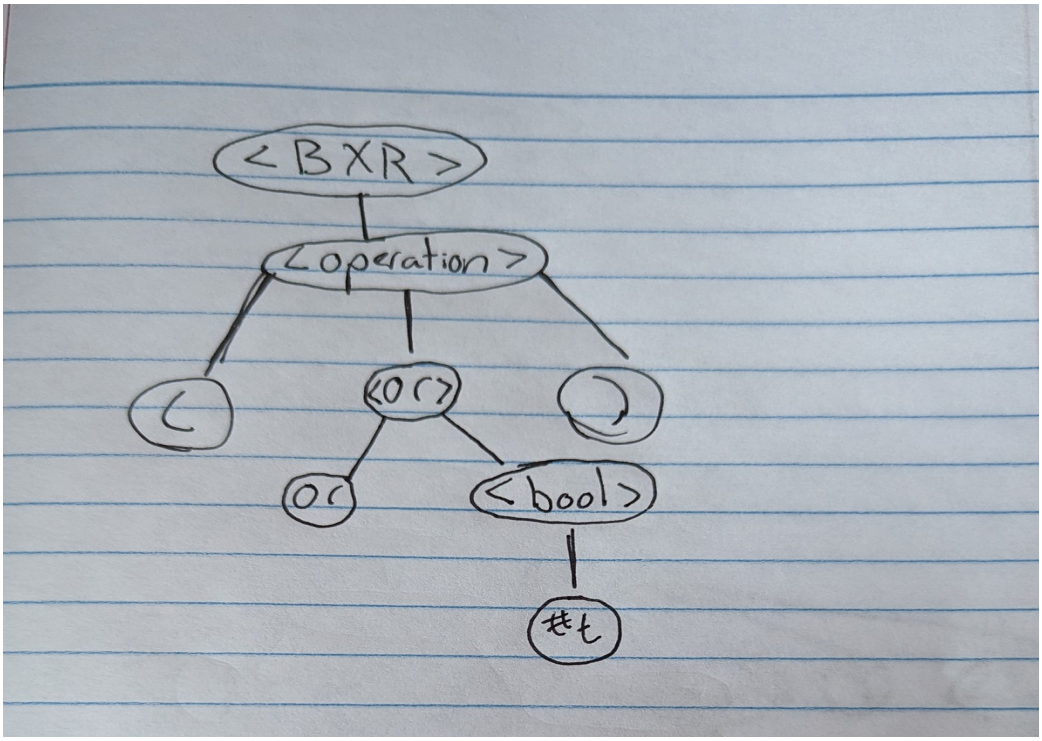
<not> ::= (not <bool>) | (not <bool>) <bool>

<bool> ::= #t | #f | <operation>

Parse Tree (Task 2): (or #t)



Parse Tree (Task 3): (and (not #t) #f)



(4) – Line Segment Sequences (LSS)

BNF Grammar (Task 1):

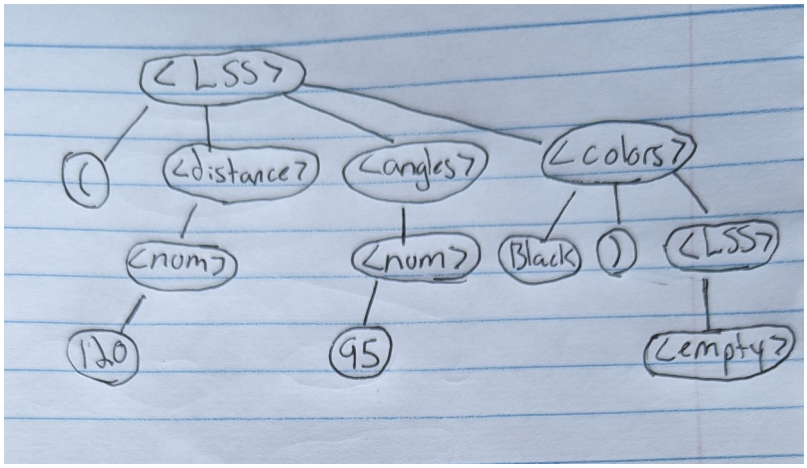
<LSS> ::= <distance> <angle> <color> | <empty>

<distance> ::= <num>

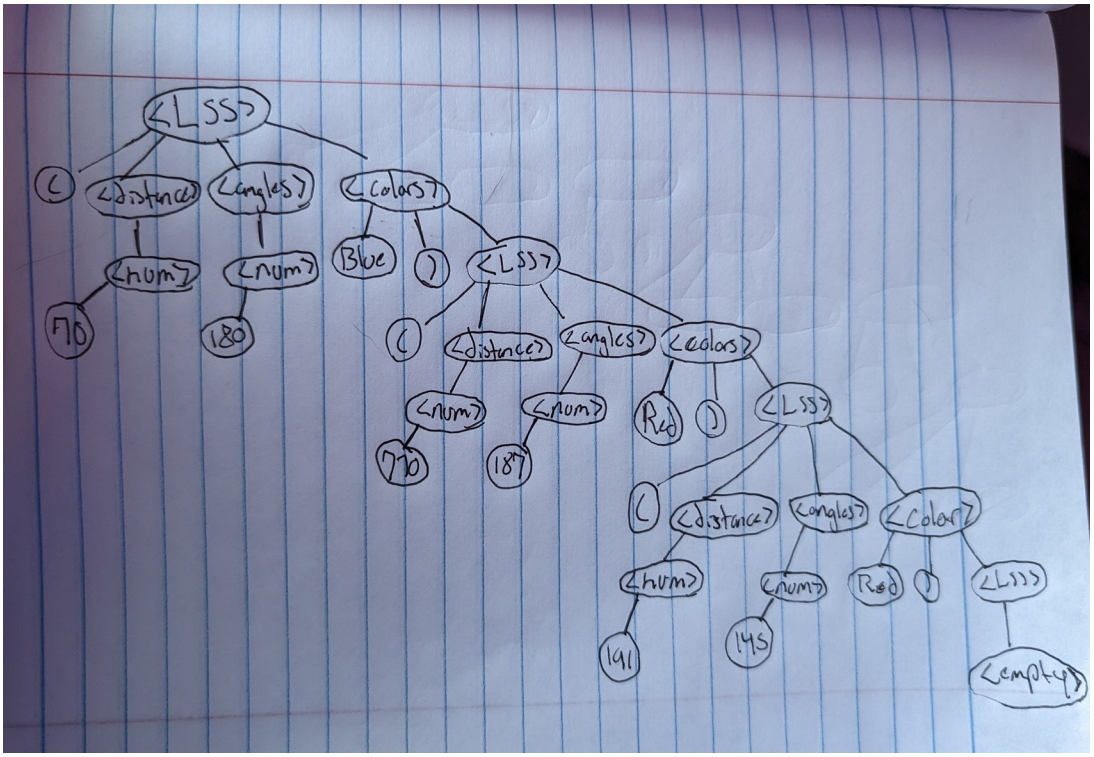
<angle> ::= <num>

<color> ::= Red) <LSS> | Blue) <LSS> | Black) <LSS>

Parse Tree (Task 2): (120 95 Black)



Parse Tree (Task 3): (170 180 Blue) (770 187 Red) (191 145 Red)



---

(5) – M-Lines

BNF Grammar (Task 1):

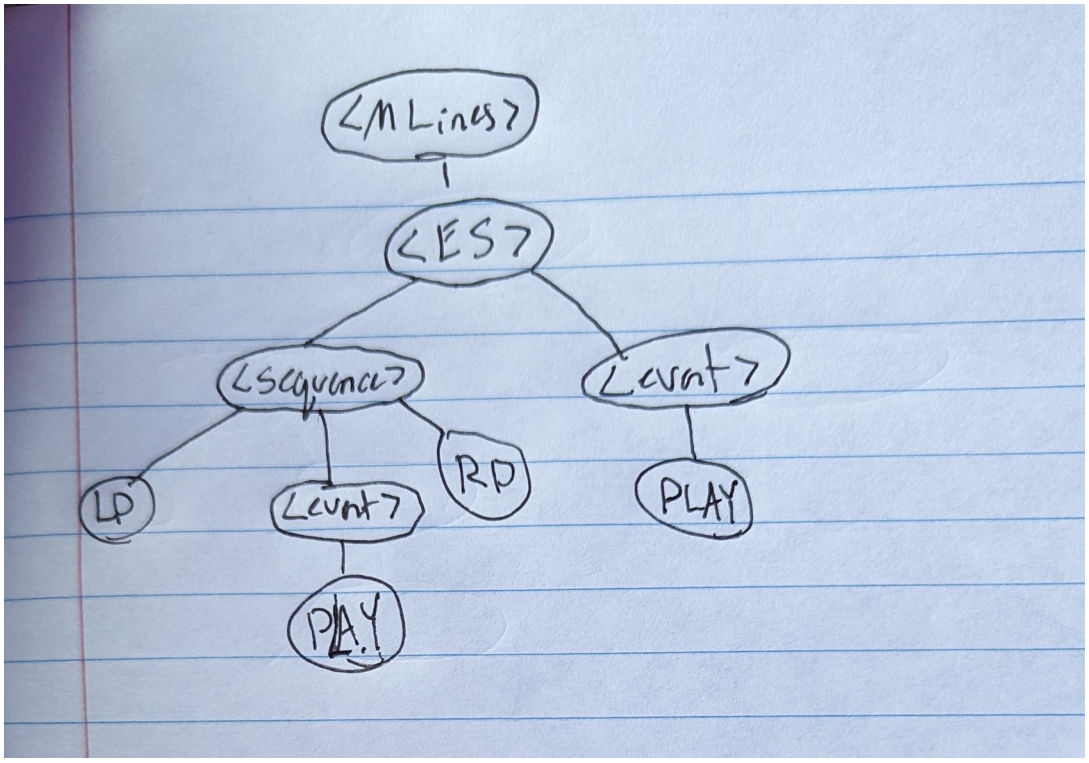
<Mlines> ::= <ES> | <empty>

<ES> ::= <event> | <sequence> | <event> <sequence> | <sequence> <event> | <empty>

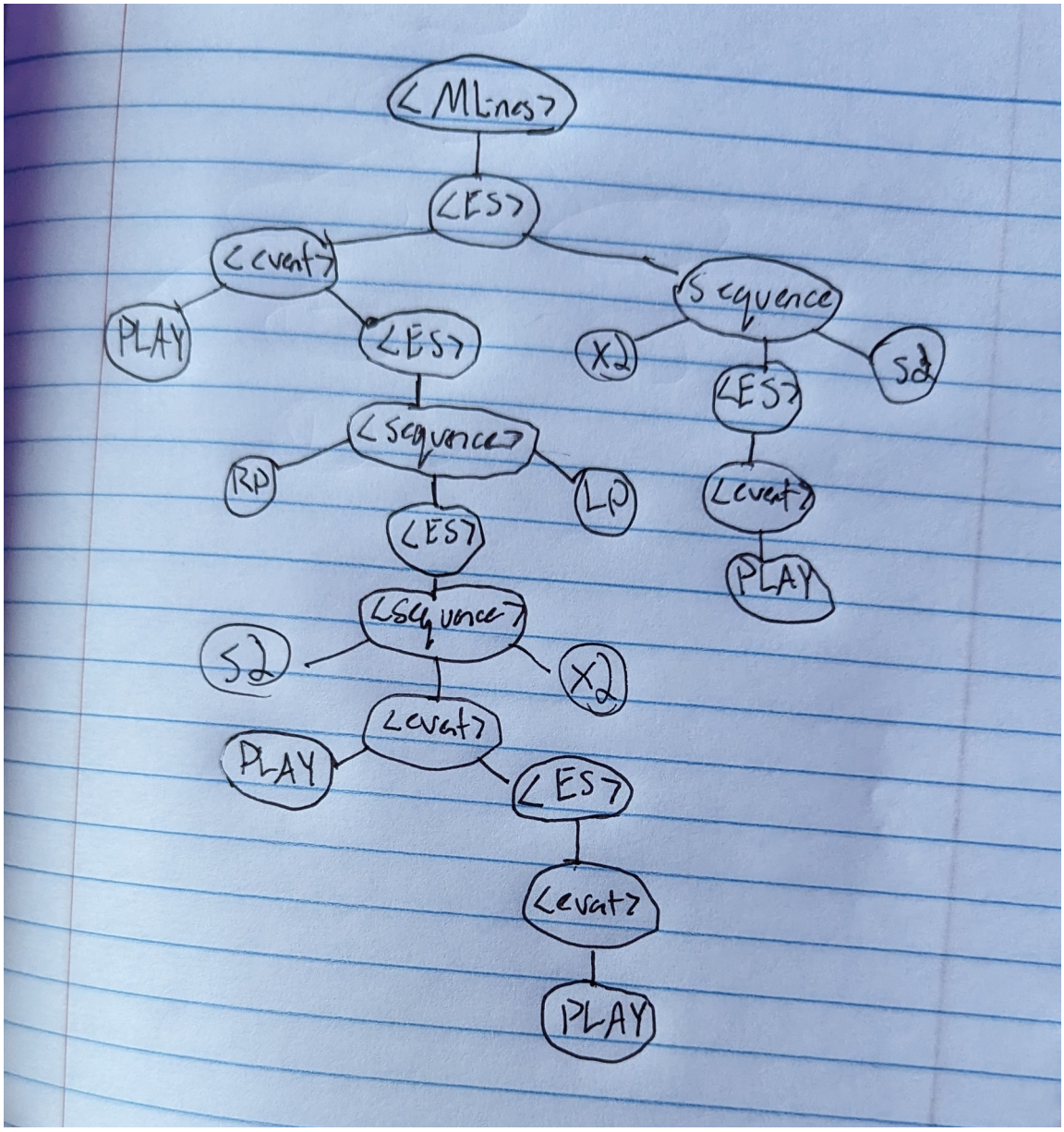
<event> ::= PLAY <ES> | REST <ES> | <ES> PLAY | <ES> REST | <empty>

<sequence> ::= RP <ES> LP | LP <ES> RP | S2 <ES> X2 | X2 <ES> S2 | S3 <ES> X3 | X3 <ES> S4 | <empty>

Parse Tree (Task 2): LP PLAY RP PLAY



Parse Tree (Task 2): PLAY RP S2 PLAY PLAY X2 LP X2 PLAY S2



(6) – BNF?

BNF stand for Backus-Naur Form and we use it to create grammar for programming languages. Through a start symbol, tokens, nonterminal symbols, and productions we can define the syntax of a language and make it act in a consistant manner.