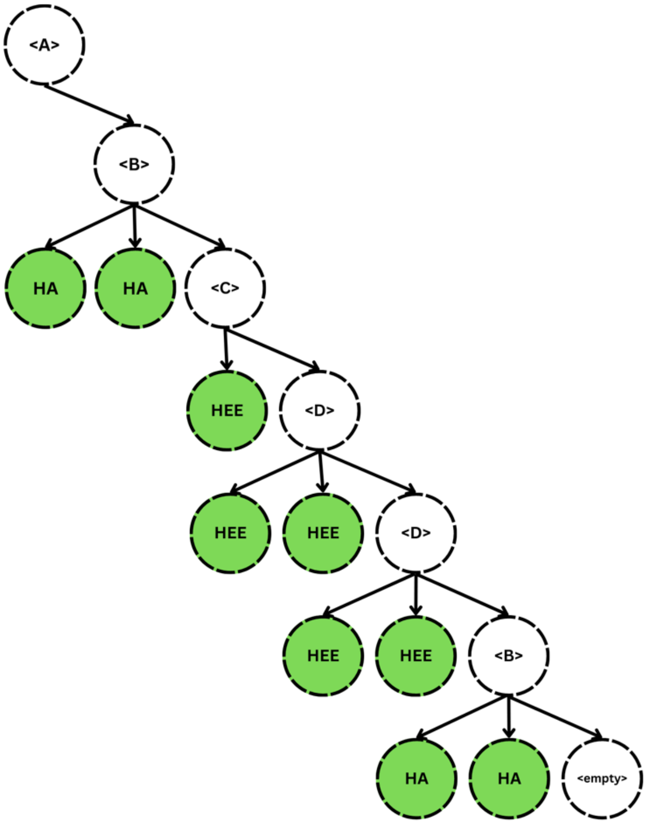
Problem Set Assignment 1 - BNF

The goal of this project is to illustrate how a BNF grammar may be organized and improved to produce a practical or relevant programming language. A "start" symbol, sets of tokens, non-terminal symbols, and productions make up BNF grammar. There will be demonstration and testing of the grammar rules for many tiny languages with different syntax and restrictions.

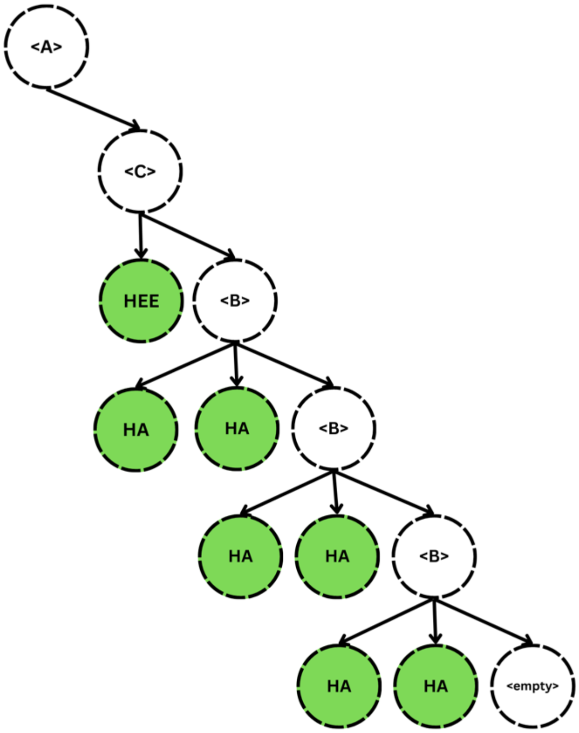
Problem 1: Laughter

```
<A> ::= <B> | <C>

<B> ::= HA HA <C> | HA HA <B> | HA HA <empty>

<C> ::= HEE <B> | HEE <D> | HEE <empty>

<D> ::= HEE HEE <B> | HEE HEE <D> | HEE HEE <empty>
```

- Parse Tree for `HA HA HEE HEE HEE HEE HEE HA HA`



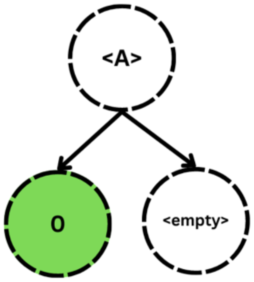
- Parse Tree for `HEE HA HA HA HA HA HA`



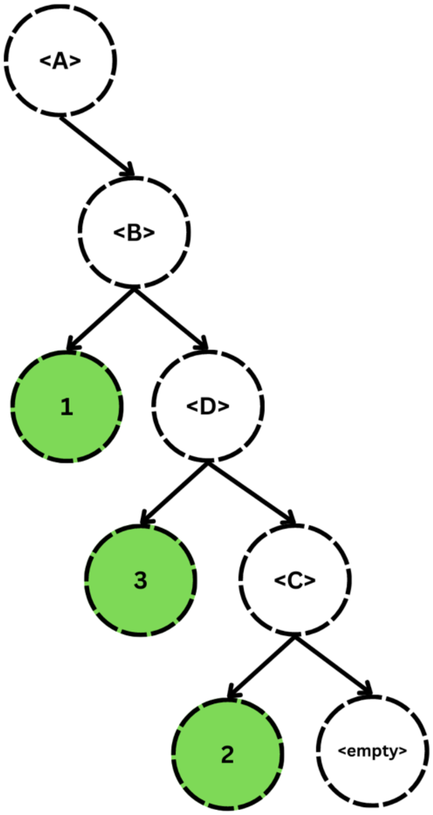
Problem 2: SQN (Special Quaternary Numbers)

```
<A> ::= 0 | <B> | <C> | <D> | <empty>
<B> ::= 1 <A> | 1 <C> | 1 <D> | <empty>
<C> ::= 2 <A> | 2 <B> | 2 <D> | <empty>
<D> ::= 3 <A> | 3 <B> | 3 <C> | <empty>
```

- Parse Tree for `0`



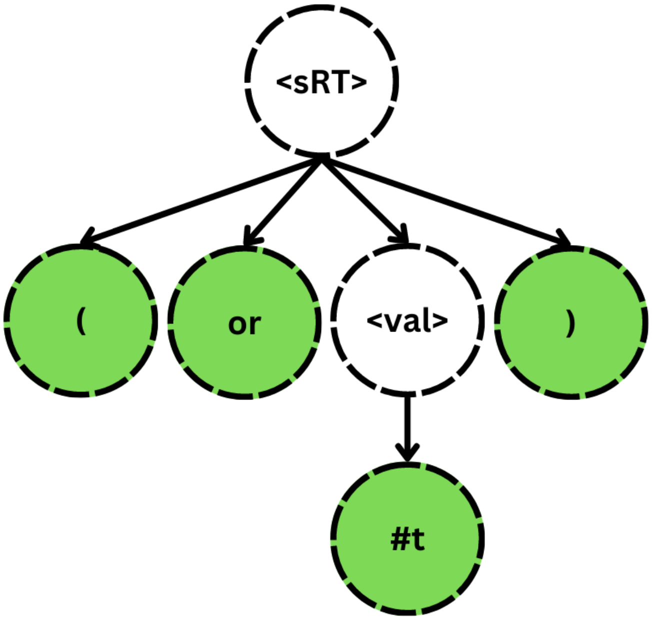
- Parse Tree for `132`



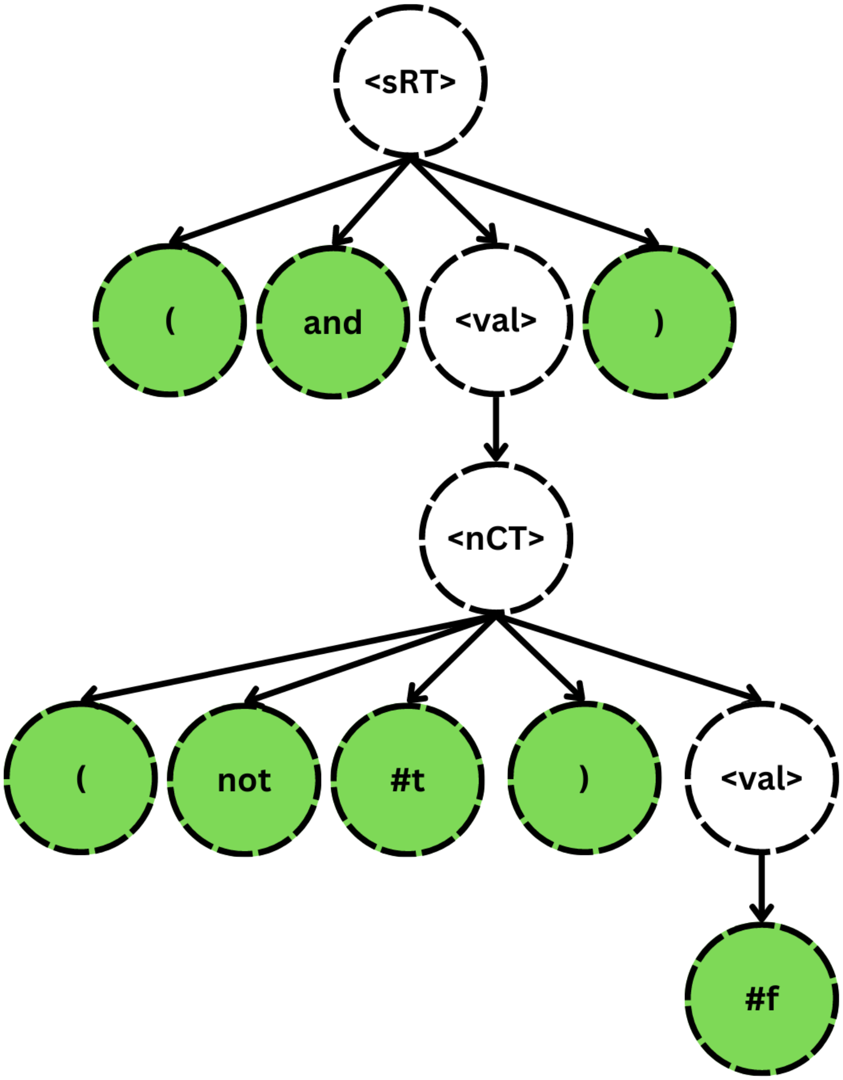
Problem 3: BXR

```
<sRT> ::= (  and <val>  ) | (  or <val>  ) | <nST> | <oST>
<val> ::=  #t  <val> |  #f  <val> | <nCT> | <oCT> |  #t  |  #f
<nCT> ::=  (  not  #t  ) <sRT> |  (  not  #f  ) <sRT> |  (  not  #t  )
<val> |  (  not  #f  ) <val>
<oCT> ::=  (   and  <val> ) |  (   or  <val> )
<nST> ::= (  not  #t  ) | (  not  #f  )
<oST> ::= (  and  #t  ) | (  or  #t  ) | (  and  #f  ) | (  or  #f  )
```

- Parse Tree for `( or #t )`



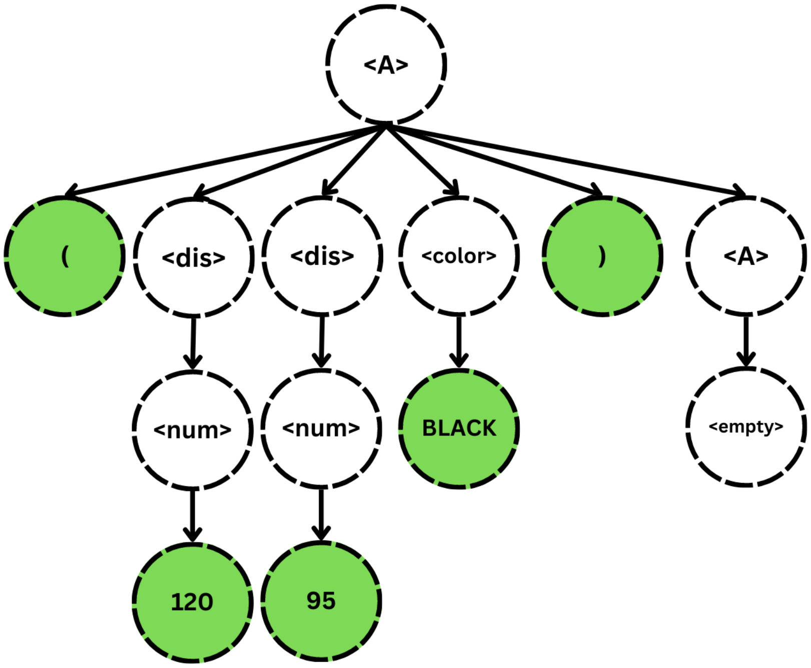
- Parse Tree for `( and ( not #t ) #f )`



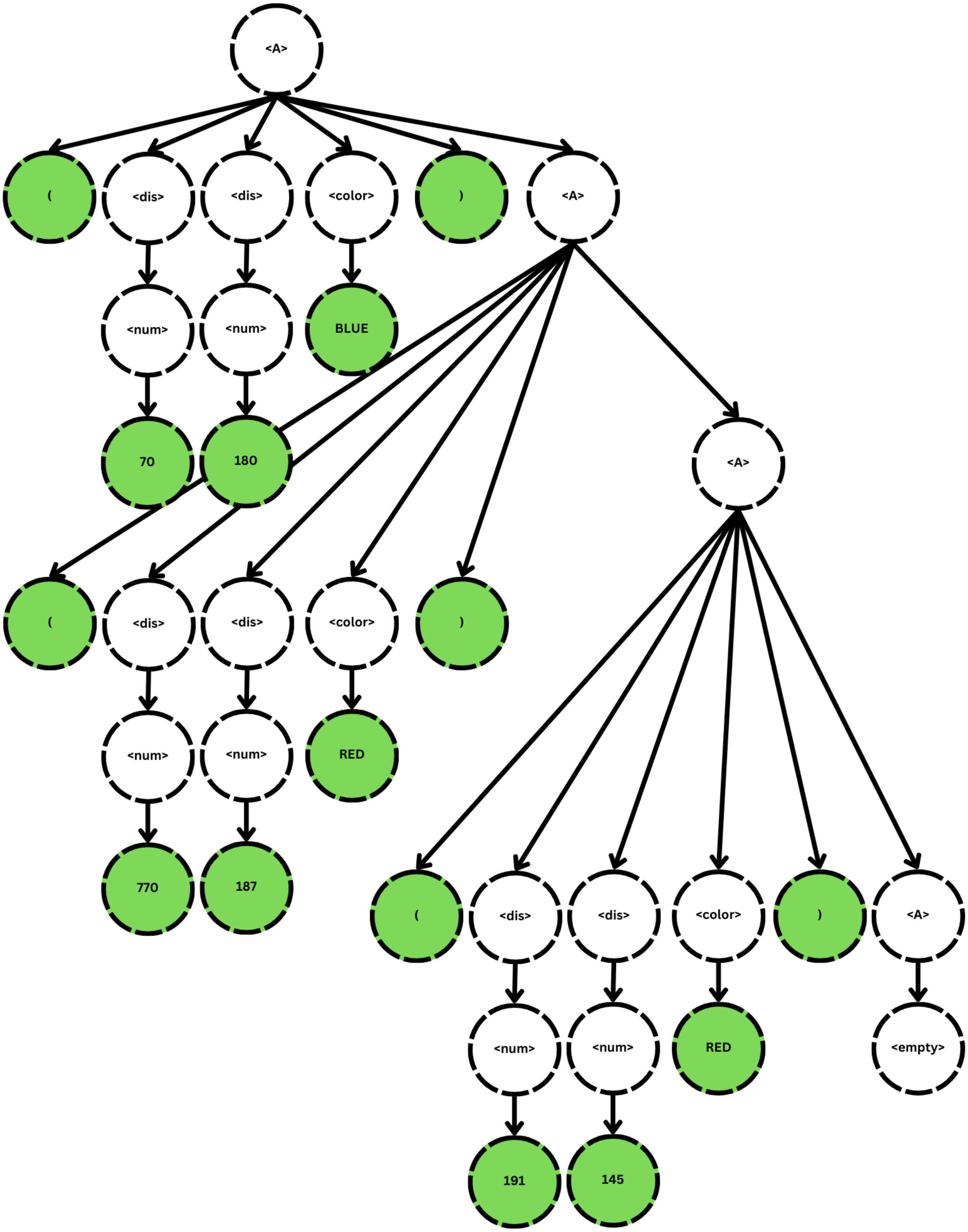
Problem 4: LSS (Line Segment Sequences)

```
<A> ::= ( <dis> <ang> <color> ) <A> | <empty>
<dis> ::= <num>
<ang> ::= <num>
<color> ::= RED | BLACK | BLUE
```

- Parse Tree for `( 120 95 BLACK )`



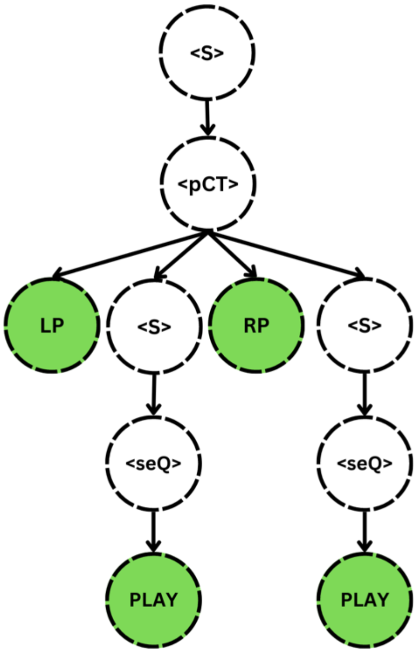
- Parse Tree for `( 70 180 BLUE ) ( 770 187 RED ) ( 191 145 RED )`



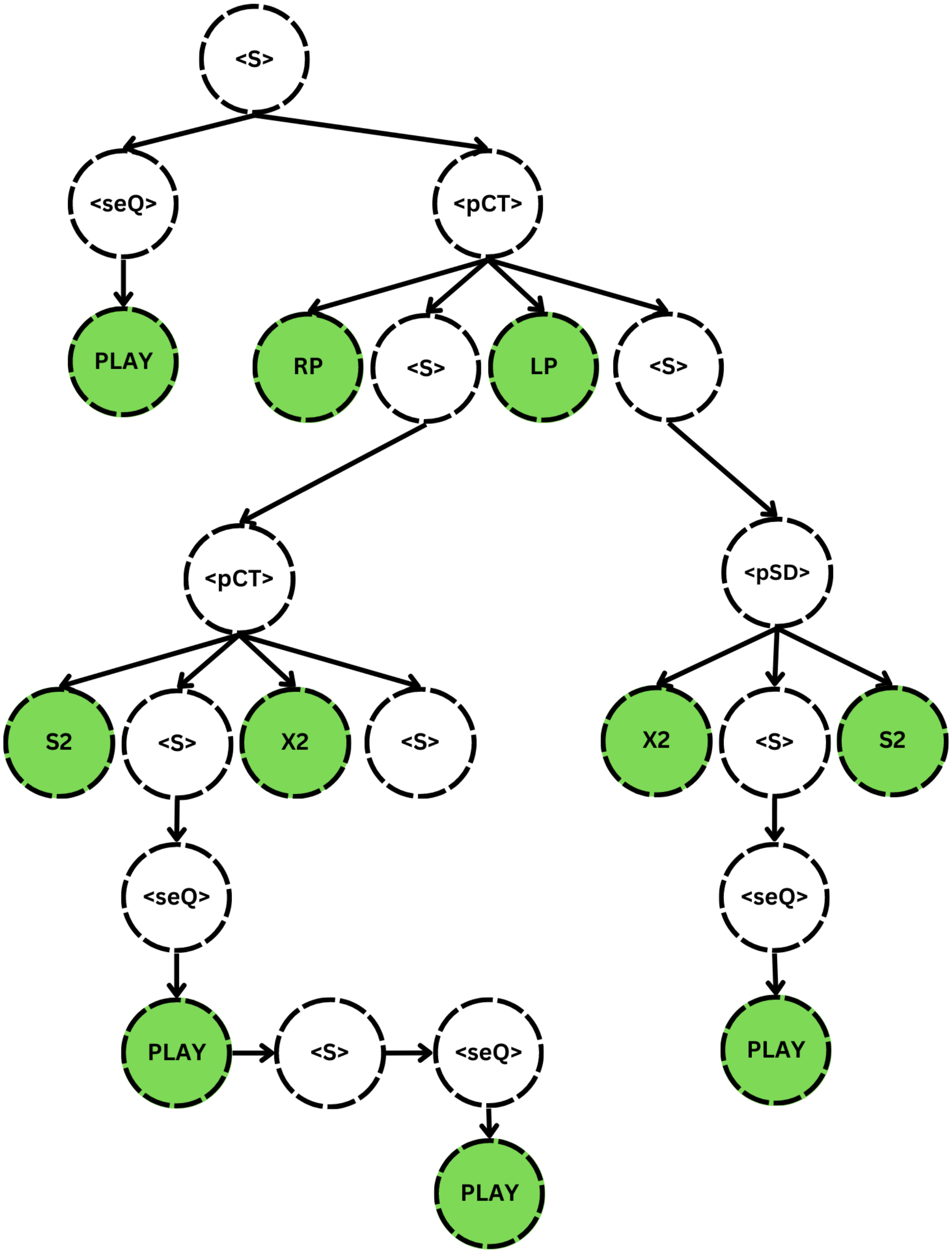
Problem 5: M-Lines

```
<S> ::= <seQ> | <pST> | <pSD> | <pCT>
<seQ> ::= PLAY | REST | PLAY <S> | REST <S>
<pCT> ::= RP <S> LP <S> | LP <S> RP <S> | S2 <S> X2 <S> | X2 <S> S2
<S> | S3 <S> X3 <S> | X3 <S> S3 <S>
<pSD> ::= RP <S> LP | LP <S> RP | S2 <S> X2 | X2 <S> S2 | S3 <S> X3 |
X3 <S> S3
<pST> ::= RP LP | LP RP | S2 X2 | X2 S2 | S3 X3 | X3 S3
```

- Parse Tree for `LP PLAY RP PLAY`



- Parse Tree for `PLAY RP S2 PLAY PLAY X2 LP X2 PLAY S2`



Problem 6: BNF?

Backus-Naur Form, or BNF as it is commonly known, is a system designed to write grammars for programming languages. To understand BNF, one must understand the 4 components that make it up. The first are the “tokens” that are a part of the language according to the definition. Secondly are the symbols that aren’t technically a part of the language but are still essential to its definition. Thirdly, the productions that convert the non-terminal symbols into a string of tokens and nonterminals. Lastly is the start symbol, a nonterminal symbol, which signifies the illustration of the language.