Harley Wakeman 2/3/2023

Abstract: In this assignment I will create DNF grammars, DNF parse trees, and explain DNF thoughtfully. It will refine my knowledge of the DNF that I've learned in class.

Problem #1:

<start> ::= <p> | <p2>

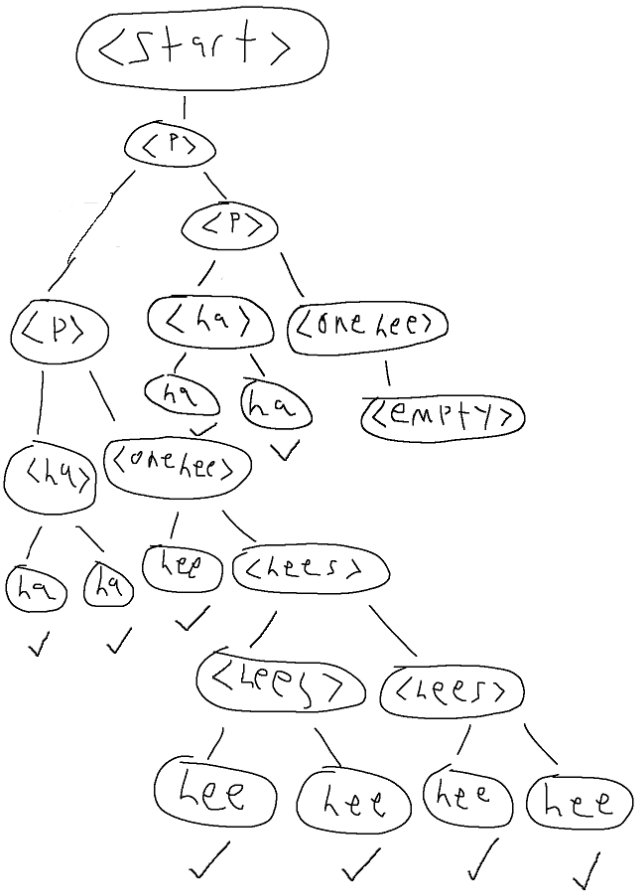
<p> ::= <p> <p> | <ha> <onehee>

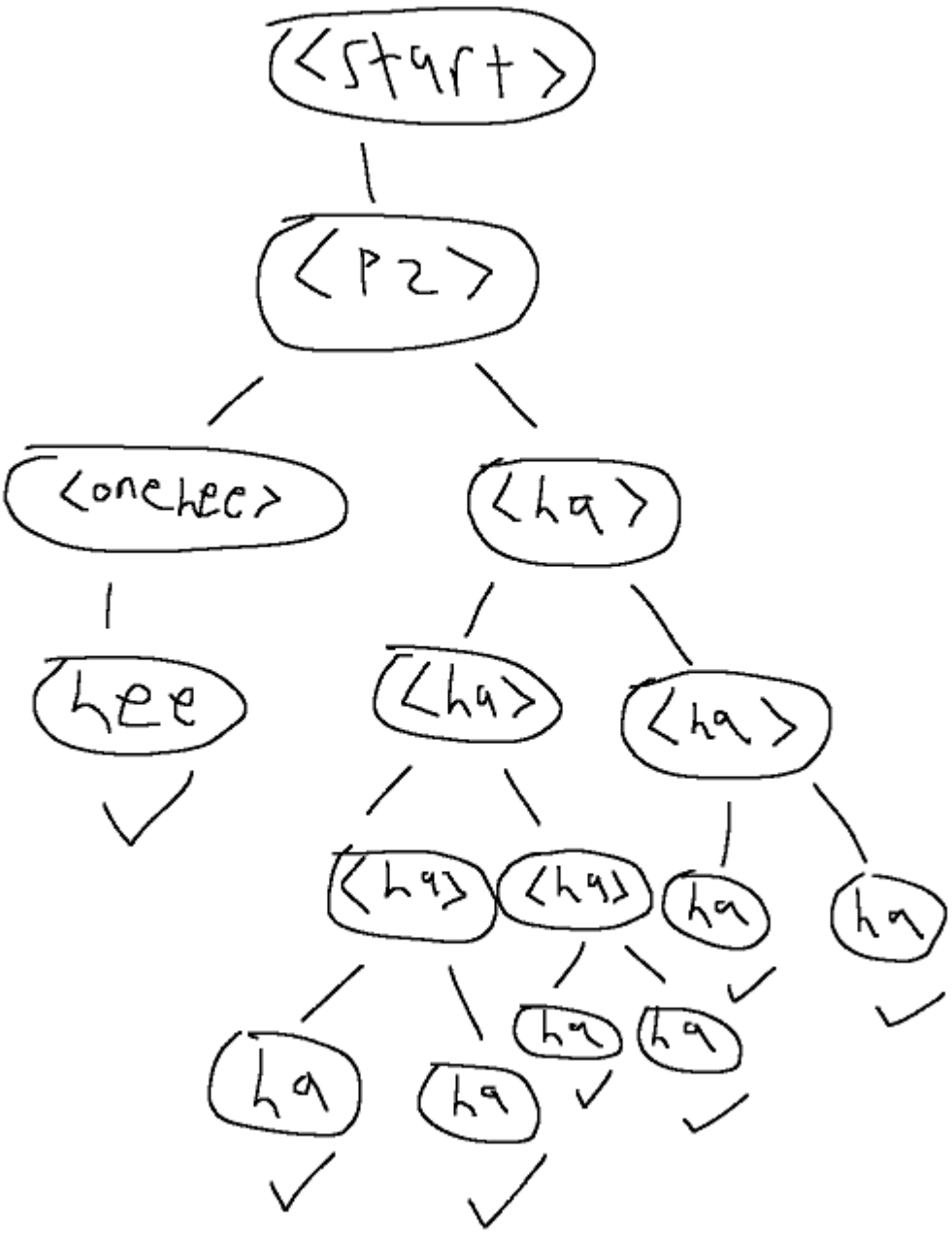
<p2> ::= <p2> <p2> | <onehee> <ha>

<ha> ::= ha ha |<ha> <ha>

<onehee> ::= hee | hee <hees> | <empty>

<hees> ::= hee hee | <hees> <hees>




<start>
<p2>
<onehee>
<h9>
hee
<h9>
<h9>
<h9>
<h9>
h9
h9
h9
h9
h9
h9

Harley Wakeman 2/3/2023

Problem #2 SQN:

<start> ::= 0 | <quaternary-num>

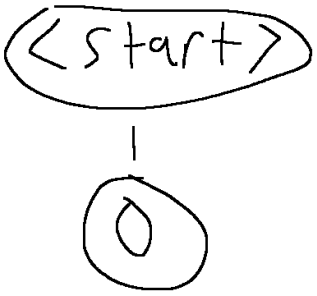
<quaternary-num> ::= 0 <quaternary-num-0>| 1 <quaternary-num-1> | 2 <quaternary-num-2> | 3 <quaternary-num-3>

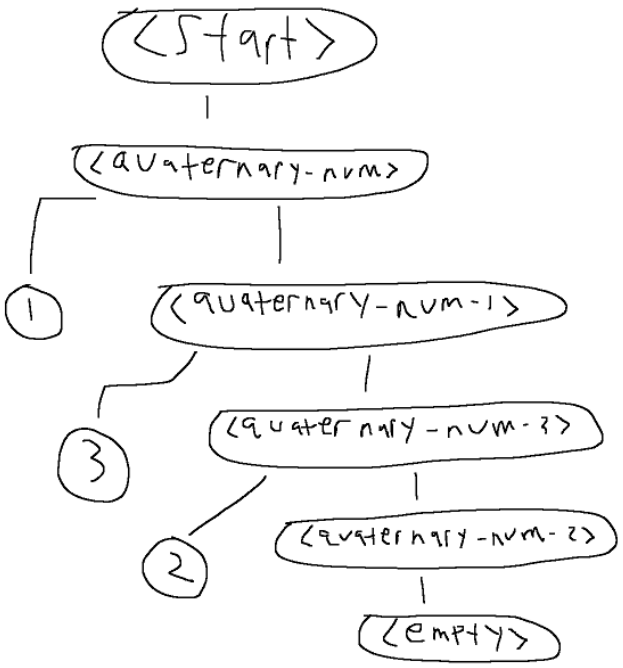
<quaternary-num-0> ::= 1 <quaternary-num-1> | 2 <quaternary-num-2> | 3 <quaternary-num-3> | <empty>

<quaternary-num-1> ::= 0 <quaternary-num-0> | 2 <quaternary-num-2> | 3 <quaternary-num-3> | <empty>

<quaternary-num-2> ::= 0 <quaternary-num-0> | 1 <quaternary-num-1> | 3 <quaternary-num-3> | <empty>

<quaternary-num-3> ::= 0 <quaternary-num-0> | 1 <quaternary-num-1> | 2 <quaternary-num-2> | <empty>





I wouldn't be able to craft a parse tree of 1223 because after using the first 2, I would be left with a nonterminal of <quaternary-num-2>, which does not contain a rule that uses another 2. I could go to one of the other nonterminals, but this would force me to use another number that isn't 2 in the meantime.

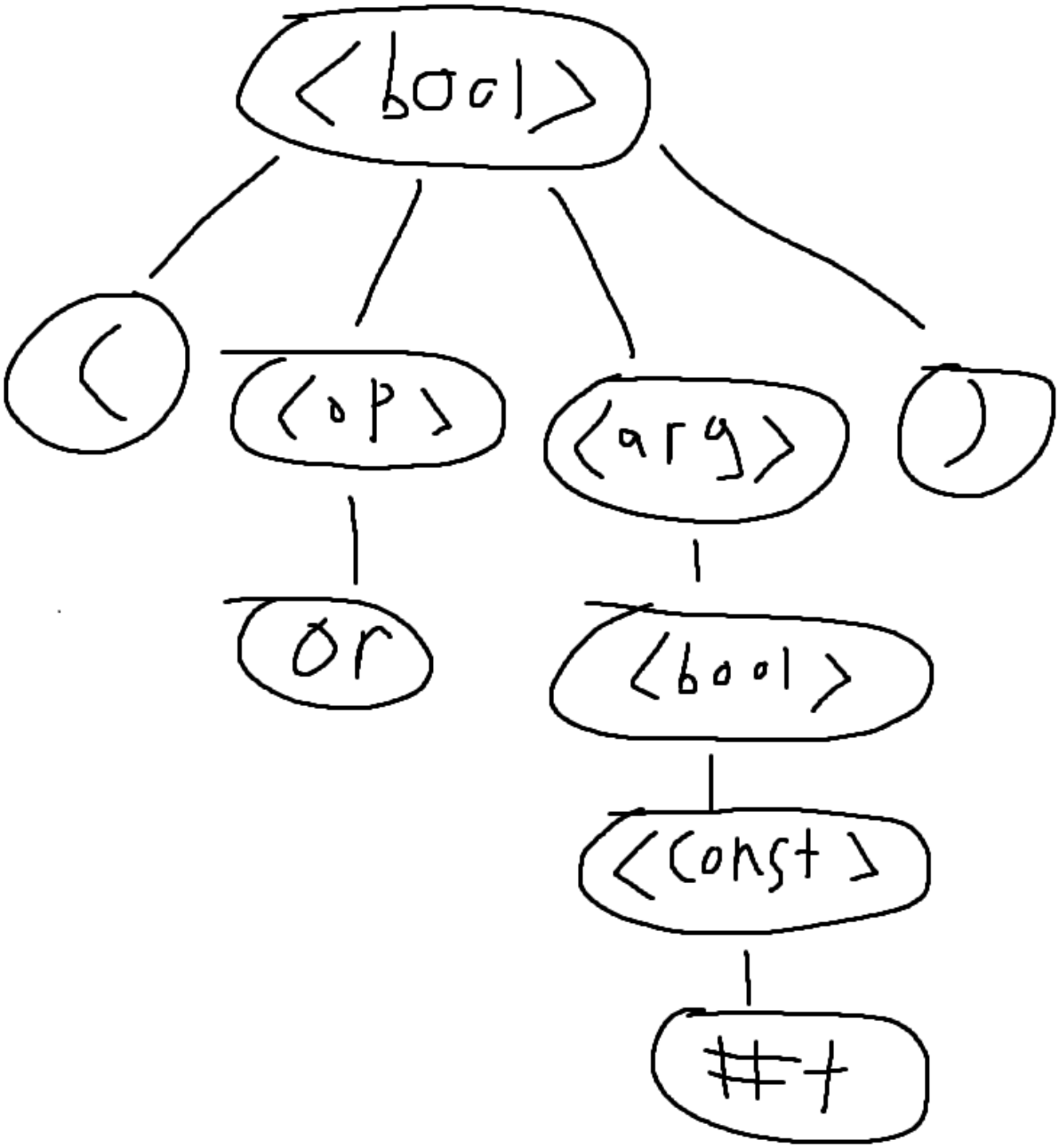
Problem #3:

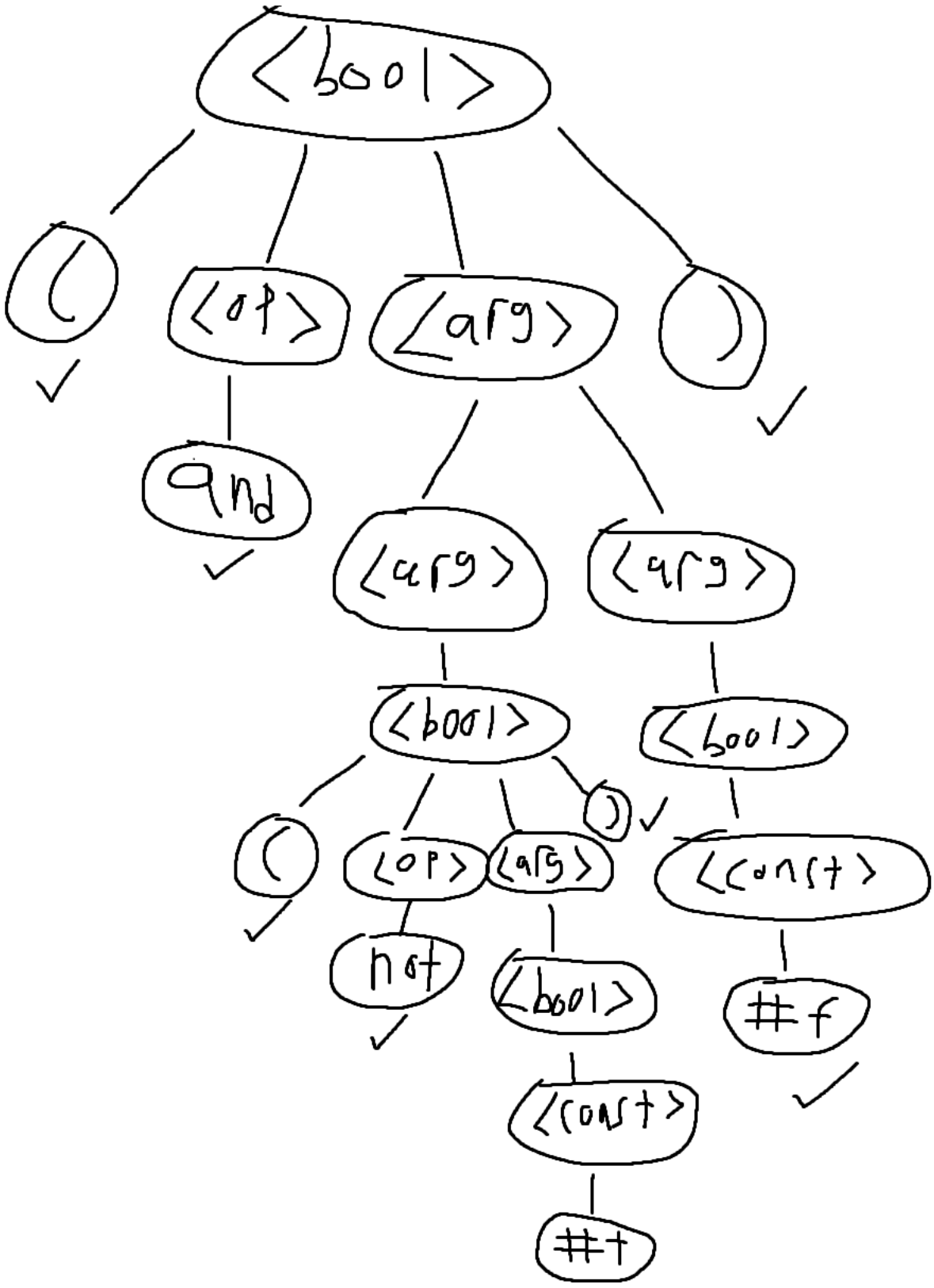
<bool> ::= (<op> <arg>) | <const>

<const> ::= #t | #f

<op> ::= and | or | not

<arg > ::= <arg> <arg> | <bool> | <empty>


<bool>
(
<op>
<arg>
)
or
<bool>
<const>
#t

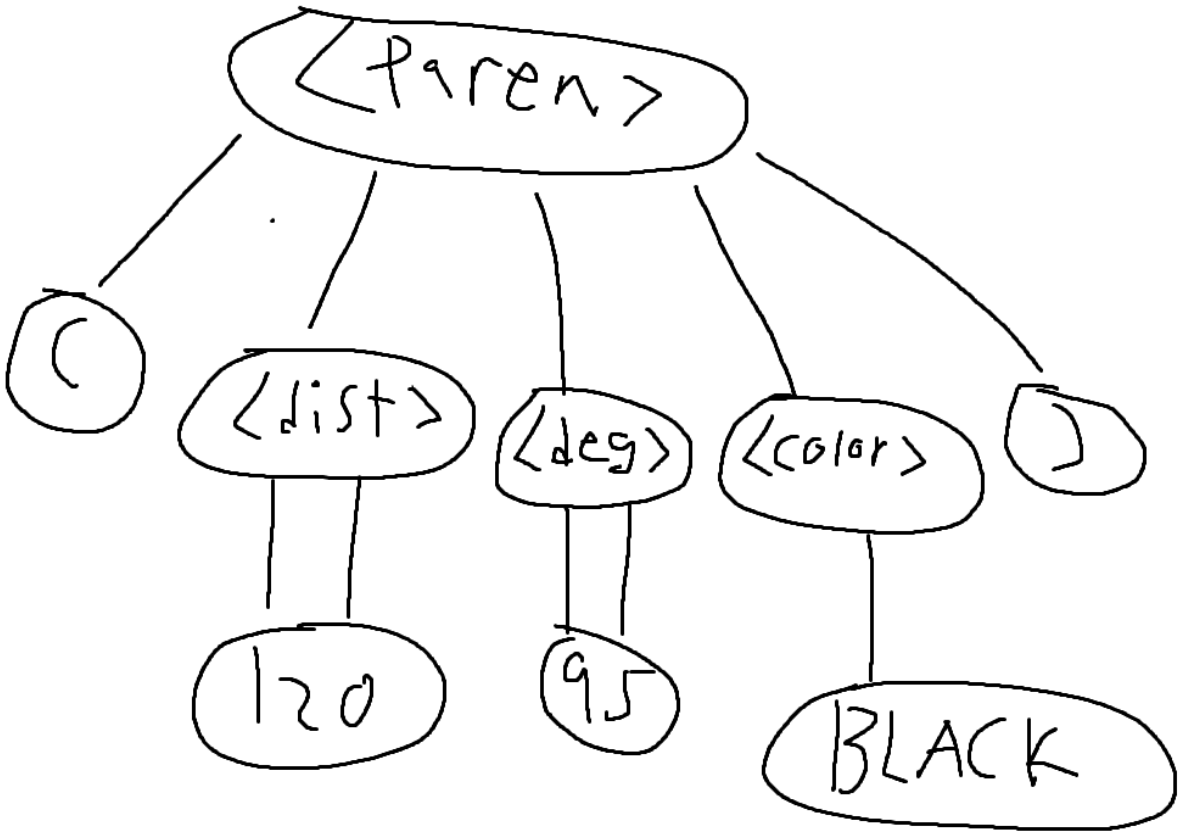

<bool>
(
<op>
<arg>
)
and
<arg>
<arg>
<bool>
<bool>
(
<op>
<arg>
)
<const>
not
<bool>
#f
<const>
#t

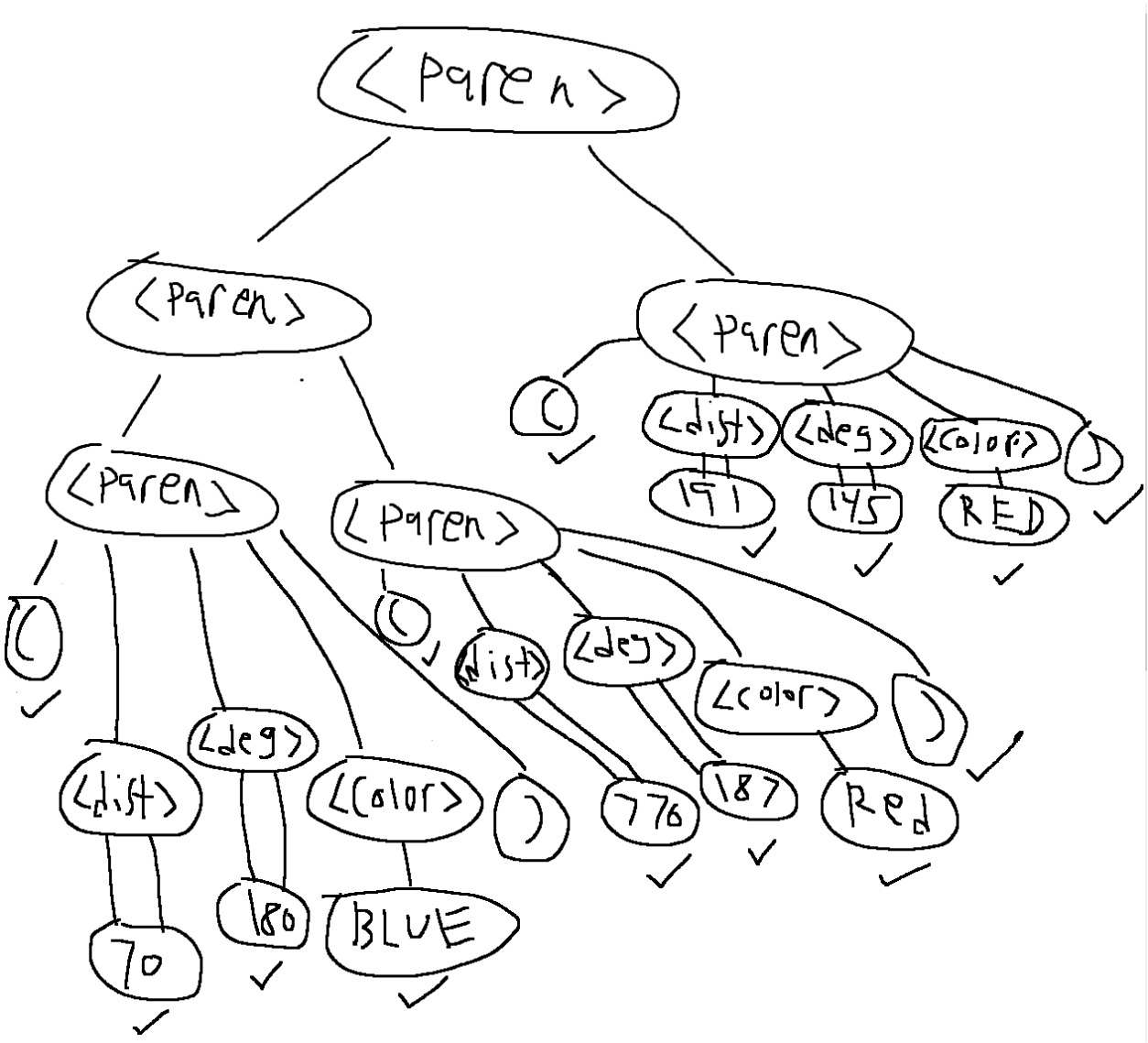
Harley Wakeman 2/3/2023

Problem #4:

<paren> ::= <paren> <paren> | (<dist> <deg> <color>) | <empty>

<color> ::= RED | BLACK | BLUE





Problem #5:

<event> ::= <event> <event> | <empty> | <rp-lp> | <lp-rp> | <s2-x2> | <x2-x2> | <s3-x3> | PLAY | REST

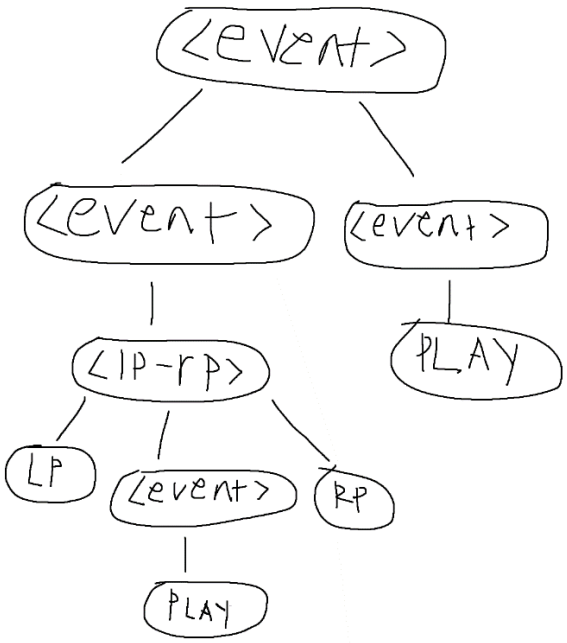
<rp-lp> ::= RP <event> LP

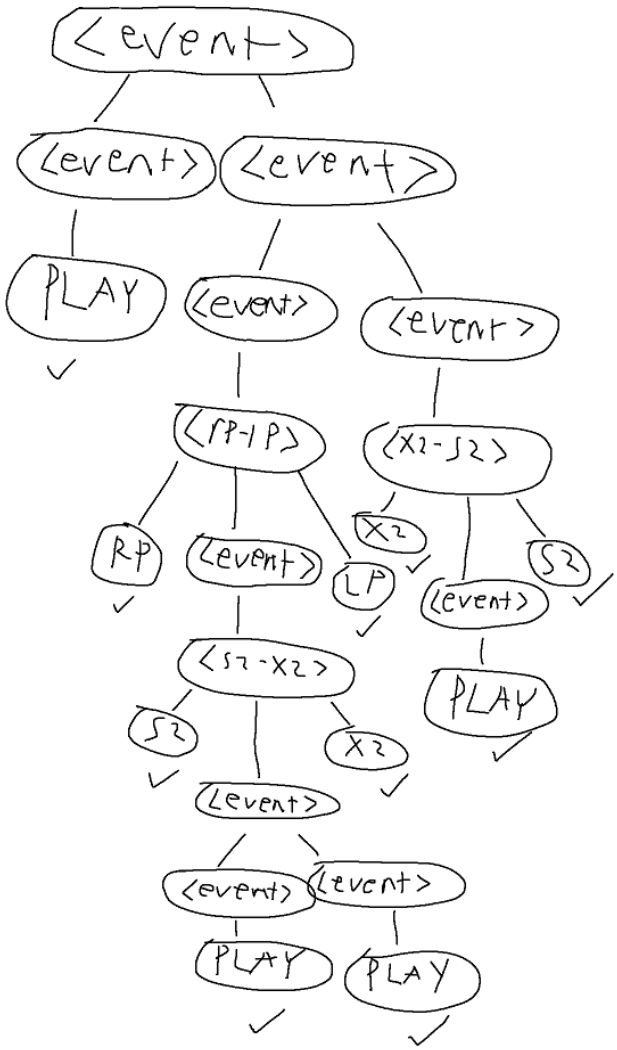
<lp-rp> ::= LP <event> RP

<s2-x2> ::= S2 <event> X2

<x2-s2> ::= X2 <event> S2

<s3-x3> ::= S3 <event> X3


<event>
<event>
<event>
PLAY
<LP-RP>
LP
<event>
RP
PLAY



Problem #6:

BNF allows us to understand the grammar of a language in an exact and scientific matter, rather than informally. In computer science, BNF describes the syntax of programming languages. BNF describes grammars by creating a set of rules that map nonterminal symbols to other nonterminal symbols or terminal tokens. Sentences or formulas can be created in the language if and only if one can start at the starting symbol and use the given rules to create the given sentence with terminal tokens.