BNF Assignment: Problem Set #1

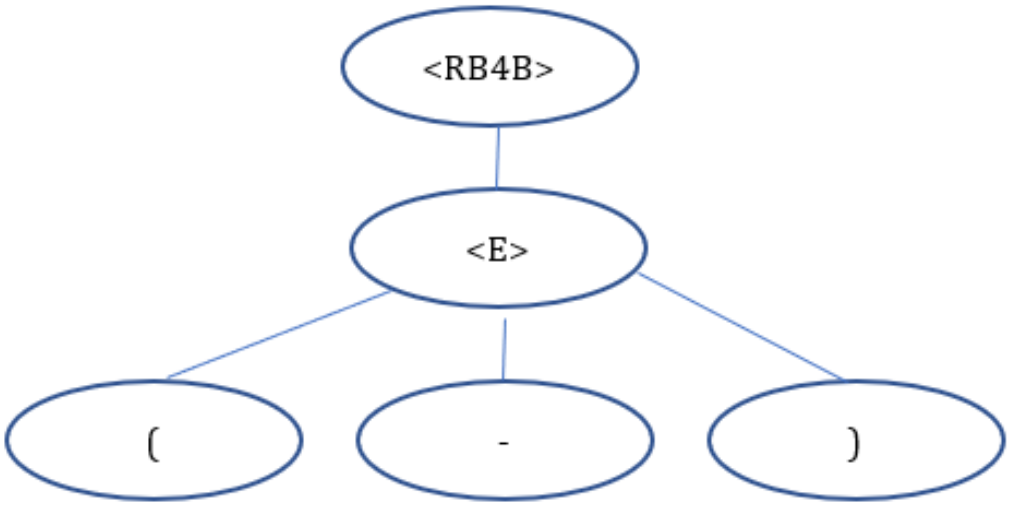
Learning Abstract

This Assignment will focus on BNF – Backus Normal Form. This is a notation I will be learning in order to write grammar for different languages. In addition to writing the grammar in BNF for different language questions, I will also be drawing the parse trees that correlate to the BNF grammar. Once I have completed these tasks, my final task will be to use what I've learned to articulate what exactly BNF is in a meaningful way. Overall, I will be familiarizing myself with BNF grammars and parse trees through the process of completing these questions.

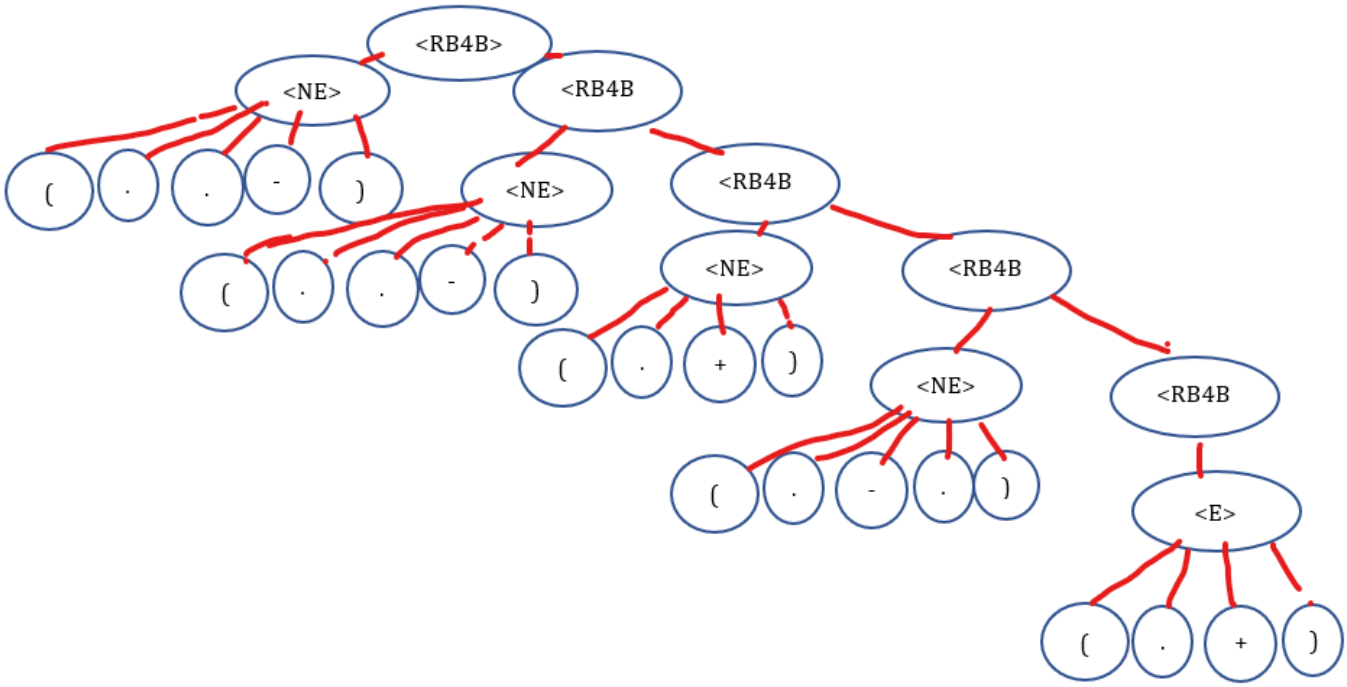
Problem 1: RB4B

1. **Write a BNF grammar description of the RB4B language.**
   Start Symbol = RB4B
   Tokens = { (,) , - , . , + , }
   Non-terminals = {RB4B, NE, E}
   <RB4B> ::= <NE> <RB4B> | <E> | <empty>
   <NE> ::= (|)| - | . | + | <empty>
   <E> ::= (-) | (. +)
   Dictionary = <NE> Non-ending, <ES> Ending

2. **Draw a parse tree, consistent with the BNF grammar that you crafted, for the following sentence: (-)**



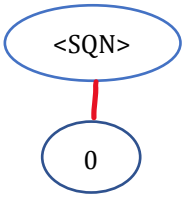
3. **Draw a parse tree, consistent with the BNF grammar that you crafted, for the following sentence: (. . -)(. . -)(. +)(. - .)(. +)**



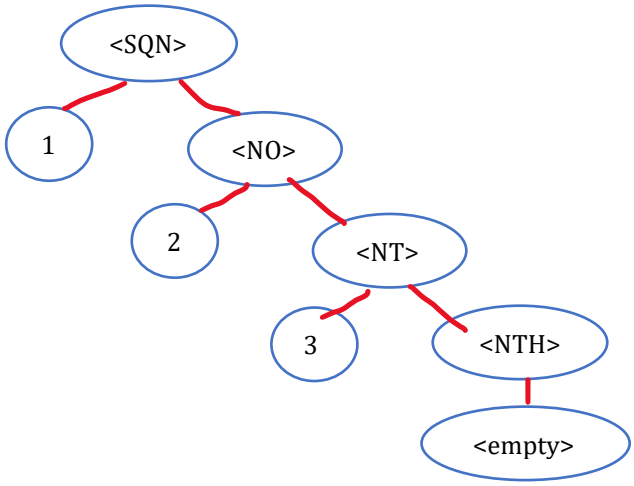
Problem 2: SQN (Special Quaternary Numbers)

1. **Write a BNF grammar description of the SQN language.**
   Start Symbol = SQN
   Tokens = {0,1,2,3}
   Non-terminals = {SQN, NO,NT,NTH,NZ}
   Dictionary = NO: not one, NT: not two, NTH: not three, NZ: not zero
   <SQN> ::= 0 | 1 <N0> | 2 <NT> |3 <NTH>
   <NO> ::= <empty> | 2 <NT> |3 <NTH>| 0 <NZ>
   <NZ> ::= <empty> | 1 <N0> | 2 <NT> |3 <NTH>
   <NT> ::= <empty> | 1 <N0> | 3 <NTH>| 0 <NZ>
   <NTH> ::= <empty> | 1 <N0> | 2 <NT> | 0 <NZ>

2. **Draw a parse tree, consistent with the BNF grammar that you crafted, for the following sentence: 0**



3. **Draw a parse tree, consistent with the BNF grammar that you crafted, for the following sentence: 132**



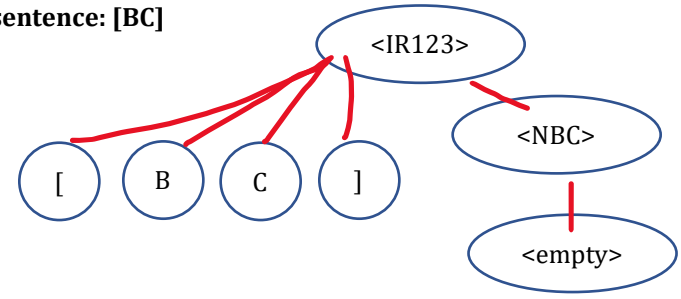
4. **Explain, in precise terms, why you cannot draw a parse tree, consistent with the BNF grammar that you crafted, for the string: 1223**

   If we follow the parse tree from top-down, We will go from <SQN> →1<NO> → 2<NT>. At this point, you have to select a rule from the nonterminal <NT>. The way the grammar is setup, there is not a rule that allows you to enter a 2 in <NT>. Therefore, after "12", you must either have <empty>, 1, 0, or 3. This applies for all the non-terminals, they have rules designed to prevent 2 adjacent occurrences of the same quaternary digit.

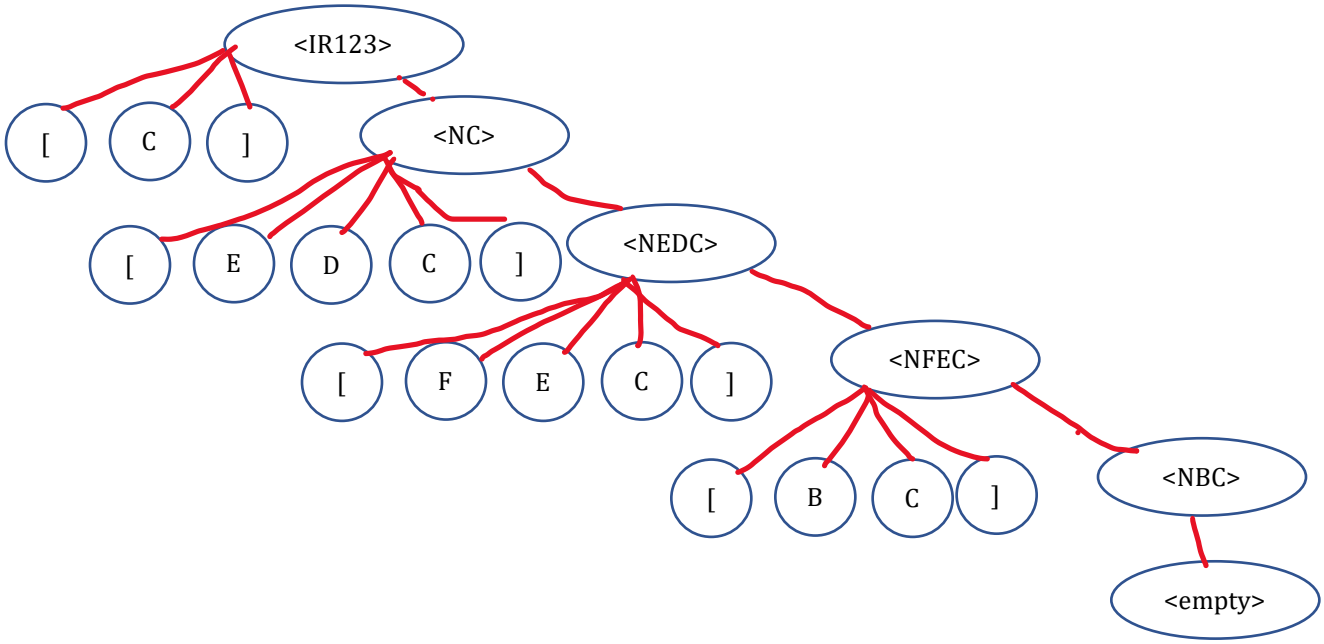
Problem 3: IR123

1. **Write a BNF grammar description of the IR123 language.**
   Start Symbol = IR123
   Tokens = {[C], [DC], [BC], [EDC], [FEC], [GFC]}
   Non-terminals = {IR123, NC, NDC, NBC, NEDC, NFEC, NGFC}
   Dictionary = NC: not C, NDC: not DC, NBC: not BC, NEDC: not EDC, NFEC: not FEC, NGFC: not GFC
   <IR123> ::= [C] <NC> | [DC] <NDC> | [BC] <NBC> | [EDC] <NEDC> | [FEC] <NFEC> | [GFC] <NGFC>
   <NC> ::= <empty> | [DC] <NDC> | [BC] <NBC> | [EDC] <NEDC> | [FEC] <NFEC> | [GFC] <NGFC>
   <NDC> ::= <empty> | [C] <NC> | [BC] <NBC> | [EDC] <NEDC> | [FEC] <NFEC> | [GFC] <NGFC>
   <NBC> ::= <empty> | [C] <NC> | [DC] <NDC> | [EDC] <NEDC> | [FEC] <NFEC> | [GFC] <NGFC>
   <NEDC> ::= <empty> | [C] <NC> | [DC] <NDC> | [BC] <NBC> | [FEC] <NFEC> | [GFC] <NGFC>
   <NFEC> ::= <empty> | [C] <NC> | [DC] <NDC> | [BC] <NBC> | [EDC] <NEDC> | [GFC] <NGFC>
   <NGFC> ::= <empty> | [C] <NC> | [DC] <NDC> | [BC] <NBC> | [EDC] <NEDC> | [FEC] <NFEC>

2. **Draw a parse tree, consistent with the BNF grammar that you crafted, for the following sentence: [BC]**



3. **Draw a parse tree, consistent with the BNF grammar that you crafted, for the following sentence: [C][E D C][F E C][B C]**



4. **Explain, in precise terms, why you cannot draw a parse tree, consistent with the BNF grammar that you crafted, for the string: [D C][B C][B C][C]**

   If we follow the parse tree from top-down, we will go from <IR123> → [DC] <NDC> → BC<NBC>. At this point, you must select a rule to go with from the nonterminal <NBC>. The way the grammar is setup, there is no choice to select [BC] in a parse tree from the nonterminal <NBC>. Therefore, after "[DC] [BC]", you must either have <empty>, [C], [DC], [EDC], [FEC], or [GFC]. This applies for all the non-terminals; they have rules designed to prevent 2 adjacent occurrences of the same bracketed sequence.

Problem 4: IR123

1. **Write a BNF grammar description of this language**

   Start symbol = BXR
   Tokens = {#t, #f, (,), and, or, not}
   Non-terminals = {BXR, operator, constant, BXR-extra}

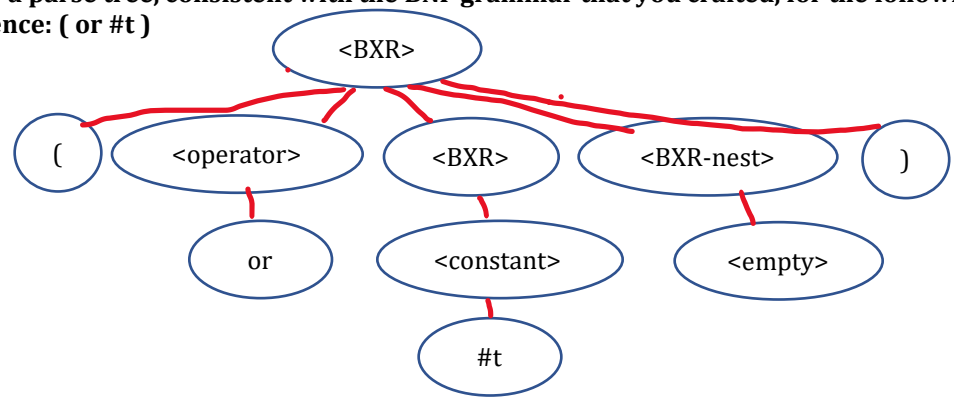
   <BXR> ::= (<operator> <BXR> <BXR-nest>) | <constant> | <empty>

   <operator> ::= and | or | not

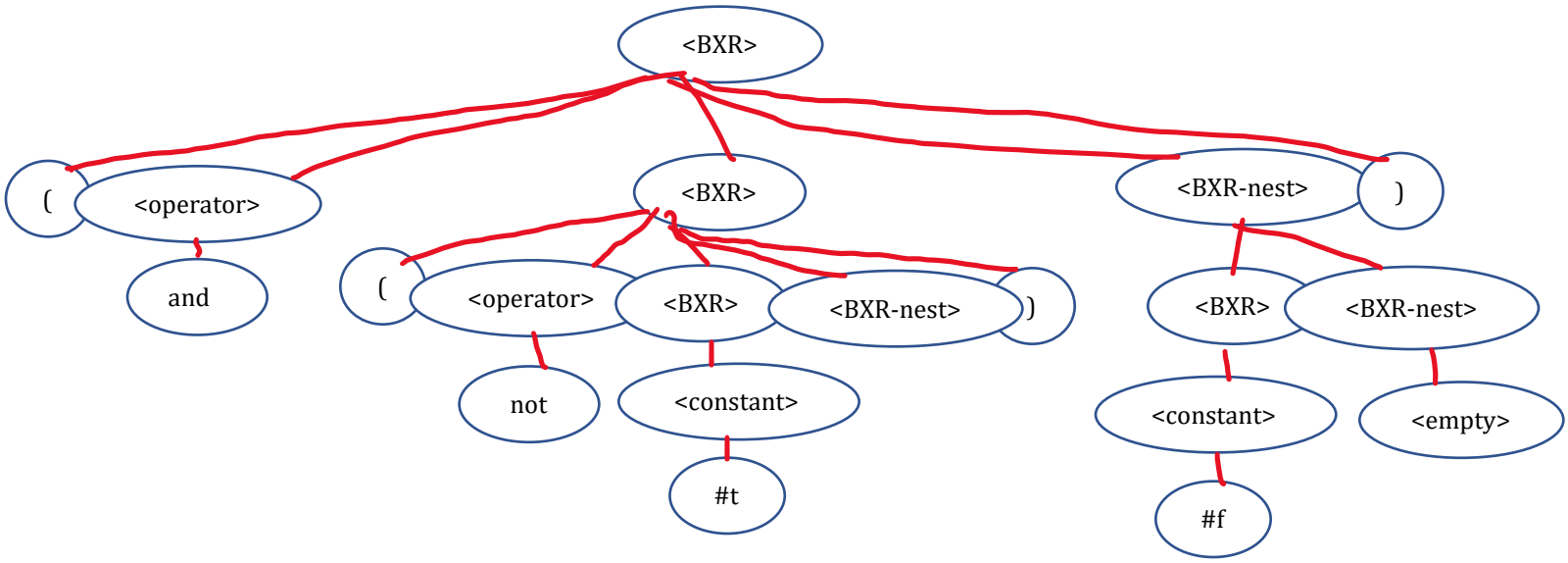
   <constant> ::= #t | #f

   <BXR-nest> ::= <BXR> <BXR-nest> | <empty>

2. **Draw a parse tree, consistent with the BNF grammar that you crafted, for the following sentence: (or #t)**

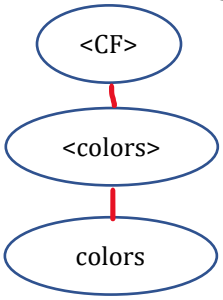


3. **Draw a parse tree, consistent with the BNF grammar that you crafted, for the following sentence: (and (not #t) #f)**

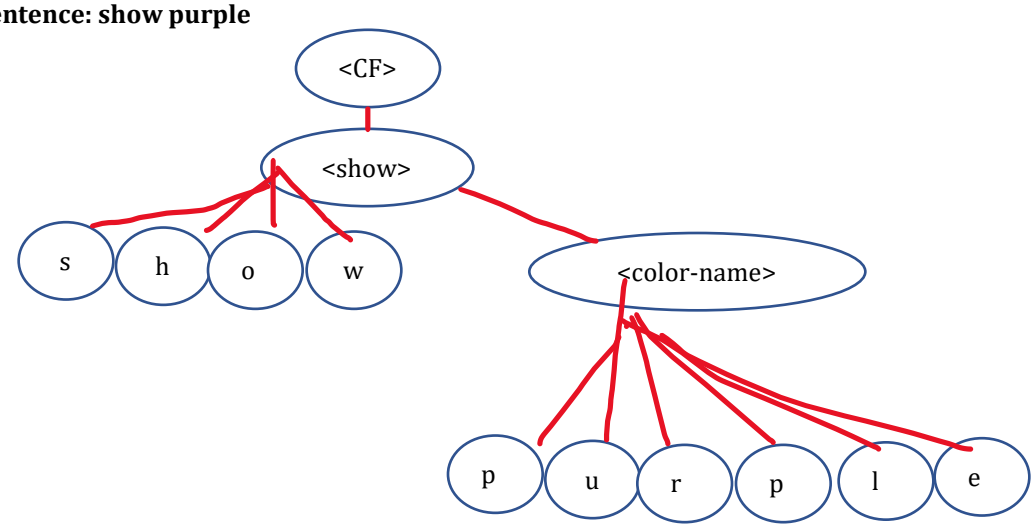


Problem 5: CF (Color Fun)

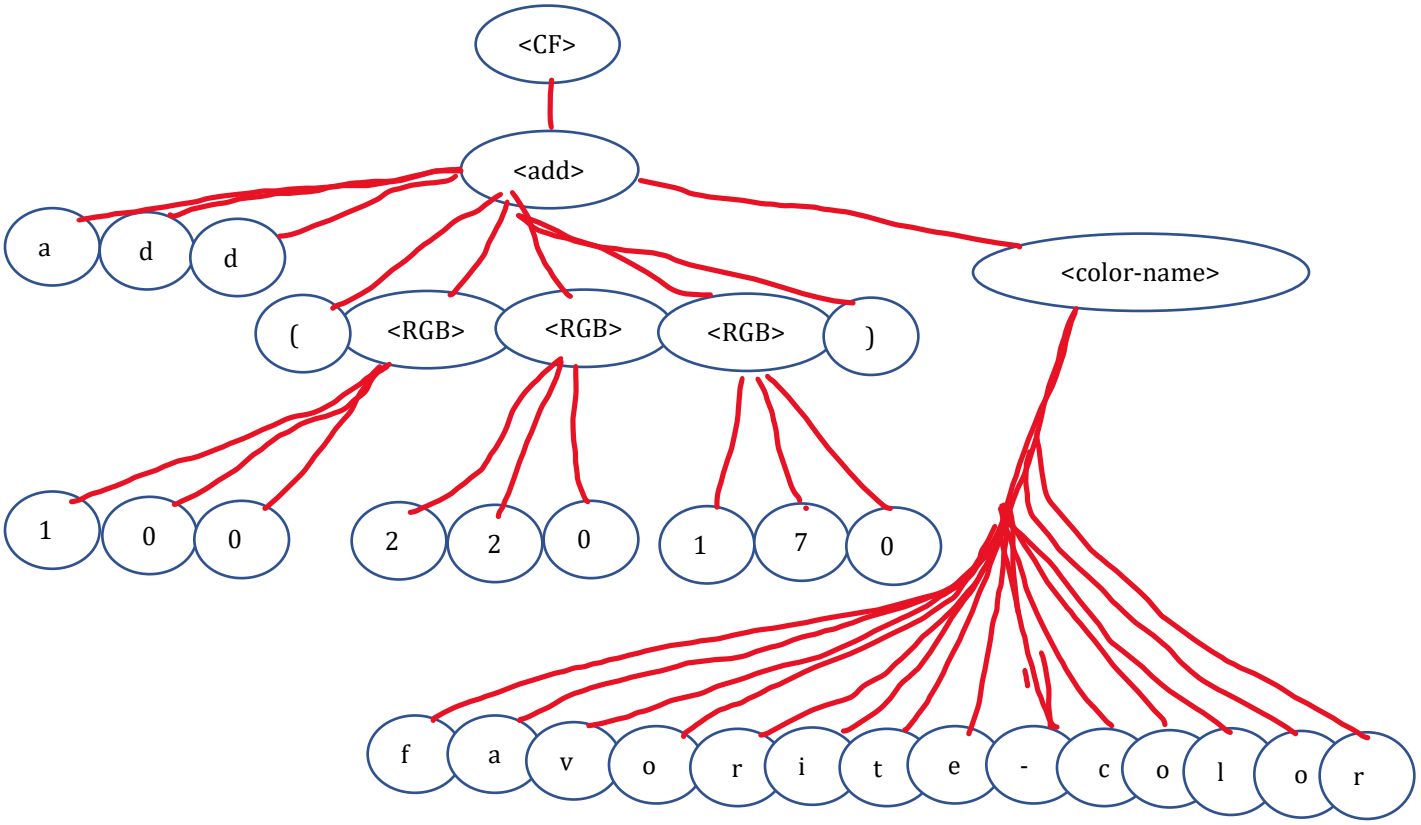
1. **Write a BNF grammar description of this language.**
   Start Symbol = CF
   Non-terminals = {CF, add, show, describe, exit, colors, color-name, RGB}
   <CF> ::= <add> | <show> | <describe> | <colors> | <exit>
   <add> ::= add (<RGB> <RGB> <RGB>) <color-name> | add (<RGB> <RGB> <RGB> <RGB>) <color-name> | add color <color-name>
   <show> ::= show <color-name>
   <describe> ::= describe <color-name>
   <colors> ::= colors
   <RGB> ::= 0|1|2|3|....|253|254|255
   <color-name> ::= color
   <exit> ::= exit
2. **Draw a parse tree, consistent with the BNF grammar that you crafted, for the following sentence: colors**



3. **Draw a parse tree, consistent with the BNF grammar that you crafted, for the following sentence: show purple**



4. **Draw a parse tree, consistent with the BNF grammar that you crafted, for the following sentence: add (100 220 170) favorite-color**



Problem 6: BNF?

1. **Please write an answer, in natural language (English, please), without examples, in a manner that you believe will serve to meaningfully inform the student about the nature and significance of BNF. Please do so in no more than 100 words.**

   BNF stands for Backus Normal Form. It is a notation used to describe the syntax of programming languages. Syntax means the rules that govern how you combine different symbols and strings in coding. BNF provides in a clear manner to the reader, what those syntax rules are. BNF uses a top down approach, allowing you conveniently move down the BNF grammar as you code to understand what the next string should be in the language. Learning new languages can be complicated, but using BNF notation will walk you through all the rules on what the system will accept in a less-complicated way.