BNF Assignment

Learning Abstract

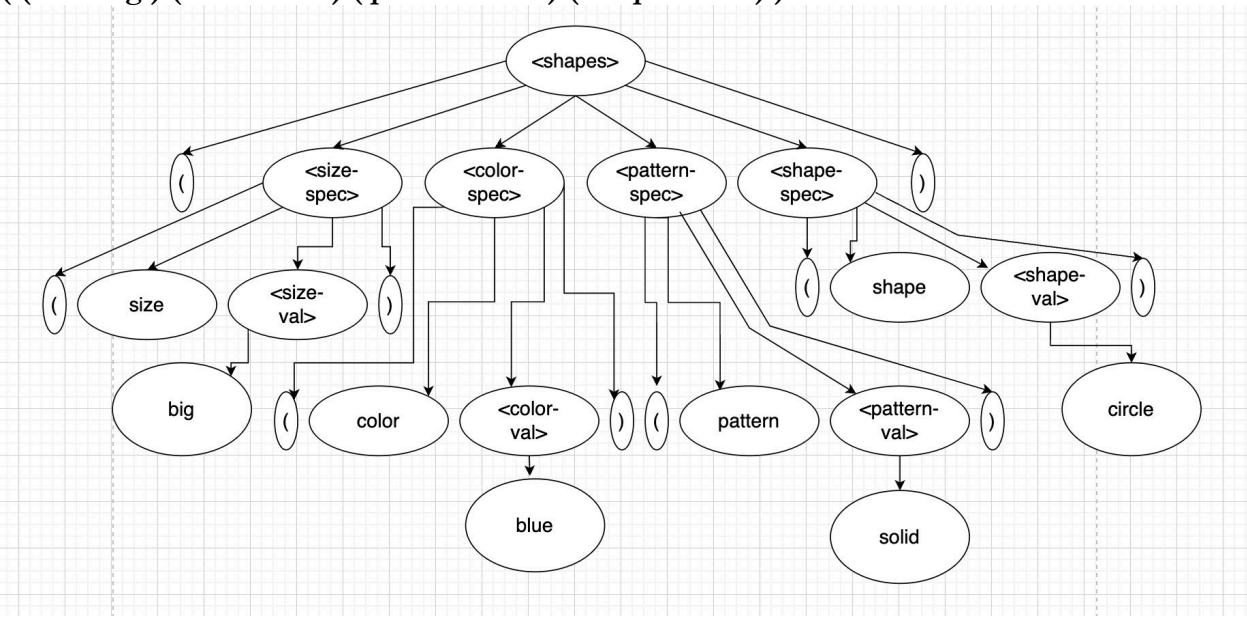
This assignment is all about BNF. I have composed some BNF grammars for given languages and drawn some BNF parse trees alongside describing BNF in English, in a straightforward, compelling manner.

Problem 1: Shapes

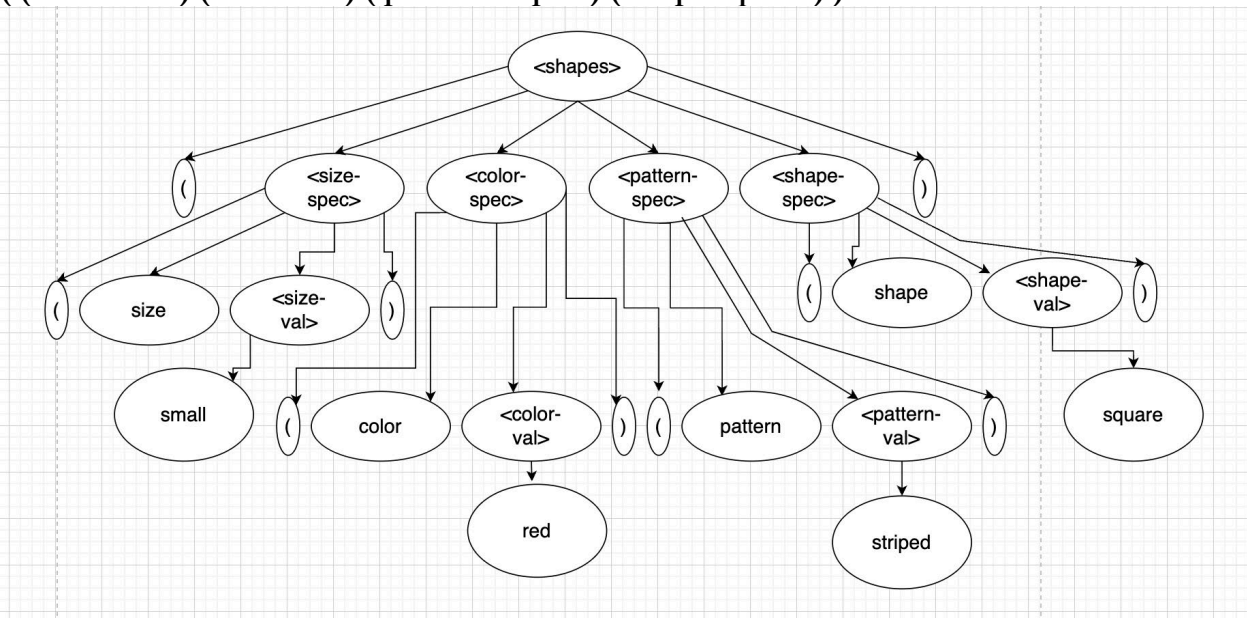
BNF Language:

<shapes> ::= (<size-spec> <color-spec> <pattern-spec> <shape-spec>)
<size-spec> ::= (size <size-val>)
<color-spec> ::= (color <color-val>)
<pattern-spec> ::= (pattern <pattern-val>)
<shape-spec> ::= (shape <shape-val>)
<size-val> ::= large | medium | small
<color-val> ::= red | blue | yellow
<pattern-val> ::= striped | dotted | solid
<shape-val> ::= circle | square | triangle

((size big) (color blue) (pattern solid) (shape circle))



((size small) (color red) (pattern striped) (shape square))



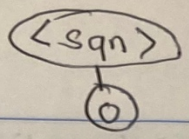
Task 2: Special Quaternary Numbers(SQN)

BNF Language:

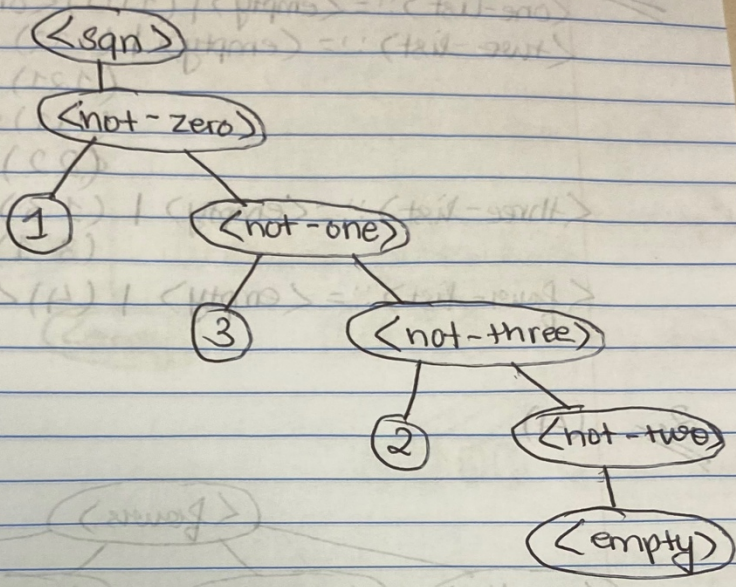
<sqn> ::= 0 | <not-zero>
<not-zero> ::= 1 <not-one> | 2 <not-two> | 3 <not-three> | <empty>
<not-one> ::= 0 <not-zero> | 2 <not-two> | 3 <not-three> | <empty>
<not-two> ::= 0 <not-zero> | 1 <not-one> | 3 <not-three> | <empty>
<not-three> ::= 0 <not-zero> | 1 <not-one> | 2 <not-two> | <empty>

Parse Trees:

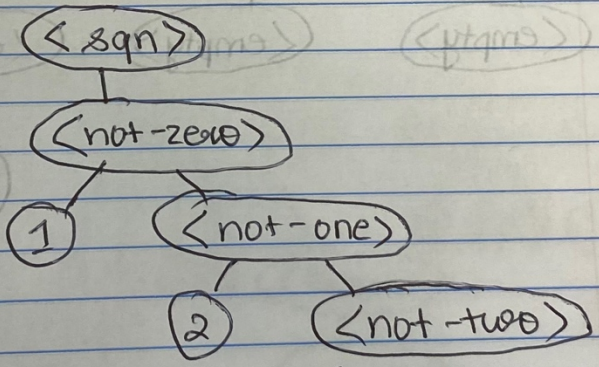
0.



132.



1223



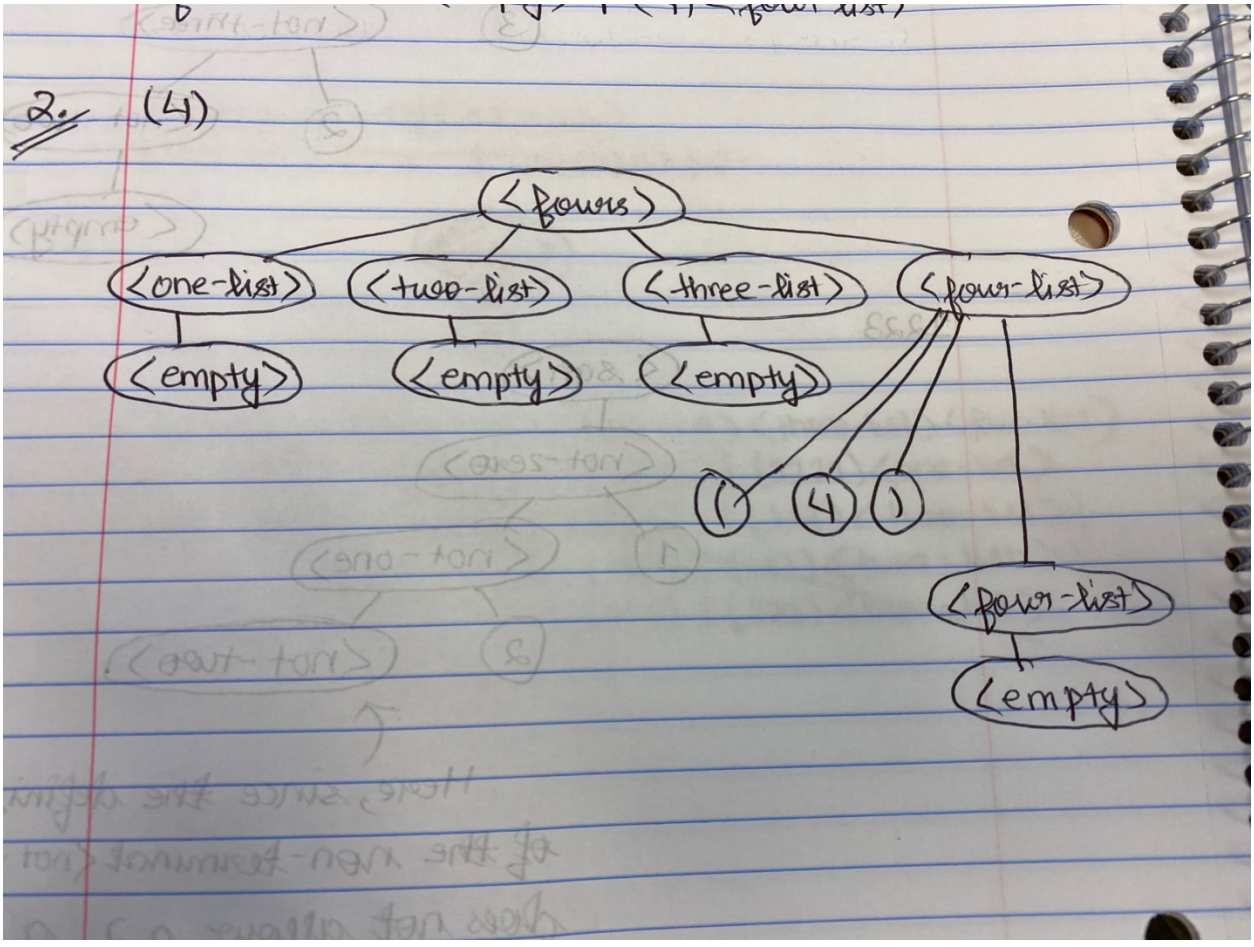
Here, since ~~the definition of the non-terminal~~ <not-two> does not allow a 2, a parse ~~tree cannot be drawn for~~ 1223.

Task 3: Fours

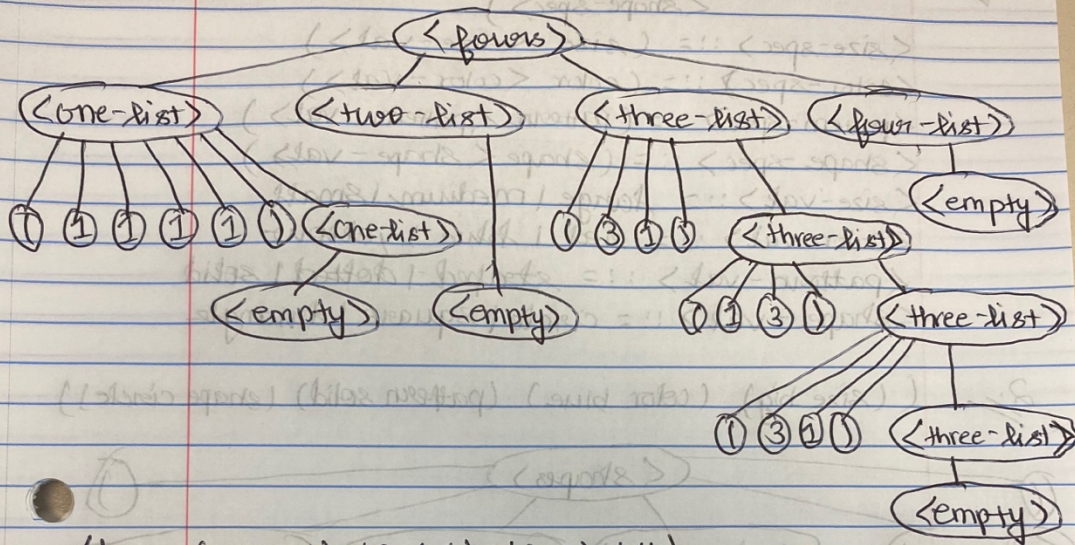
BNF Language:

<fours> ::= <one-list> <two-list> <three-list> <four-list>
<one-list> ::= <empty> | (1 1 1 1) <one-list>
<two-list> ::= <empty> | (1 1 2) <two-list> | (1 2 1) <two-list> | (2 1 1) <two-list> | (2 2) <two-list>
<three-list> ::= <empty> | (1 3) <three-list> |(3 1) <three-list>
<four-list> ::= <empty> | (4) <four-list>

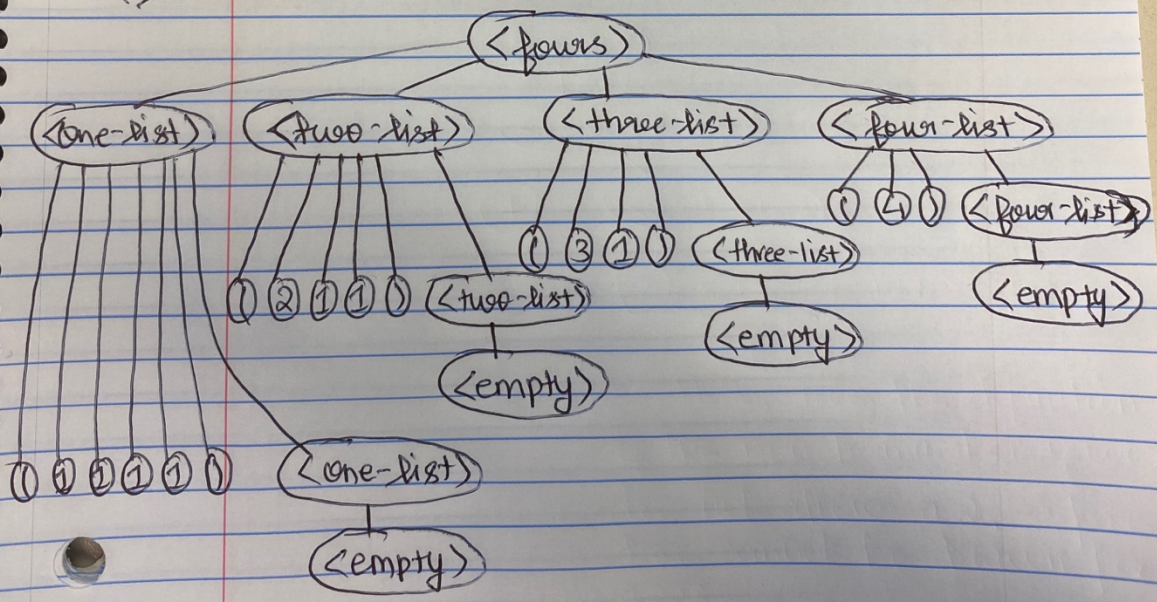
Parse Trees:



3. (1 1 1 1) (3 1) (1 3) (3 1)

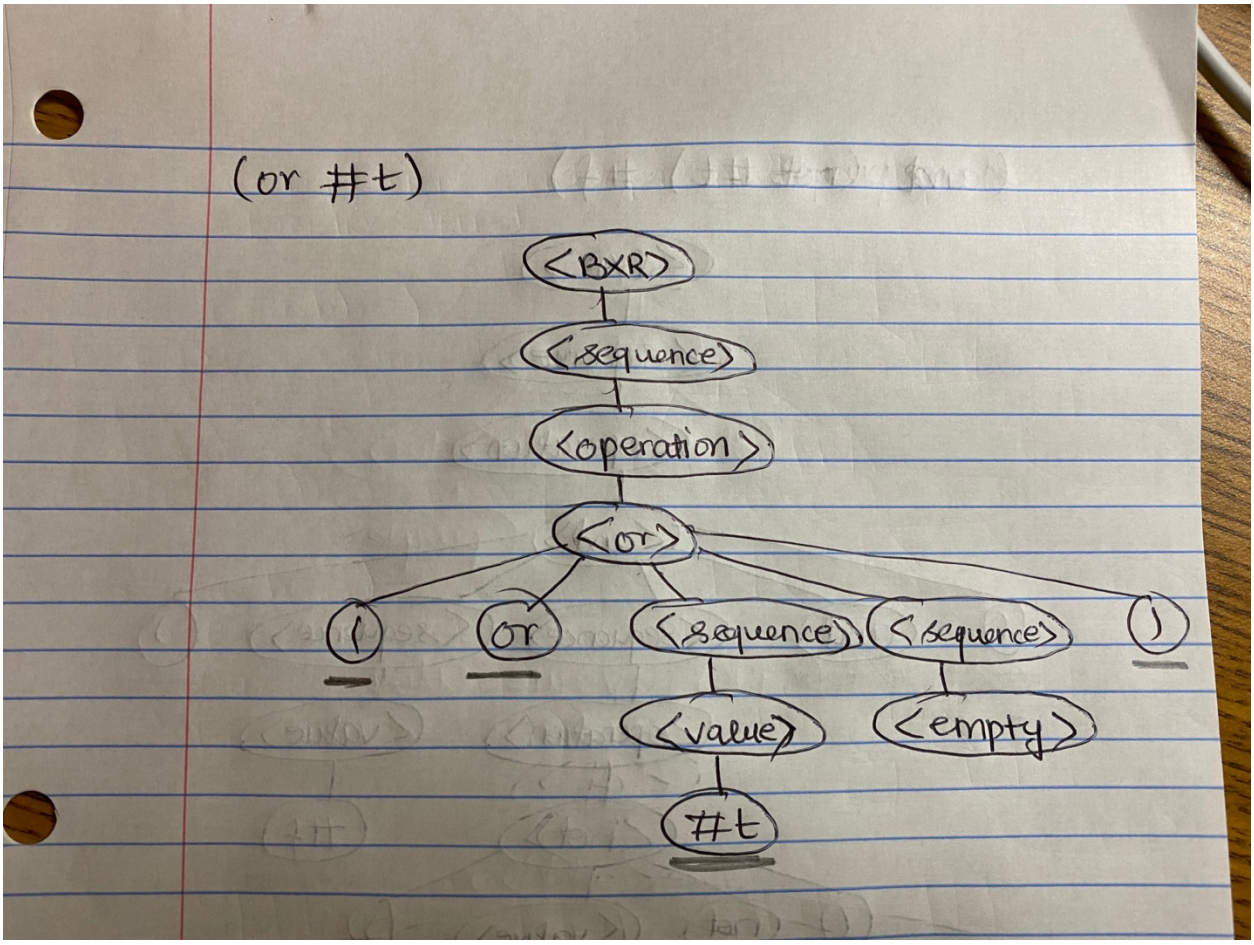


4. (1 1 1 1) (2 1 1) (3 1) (4)

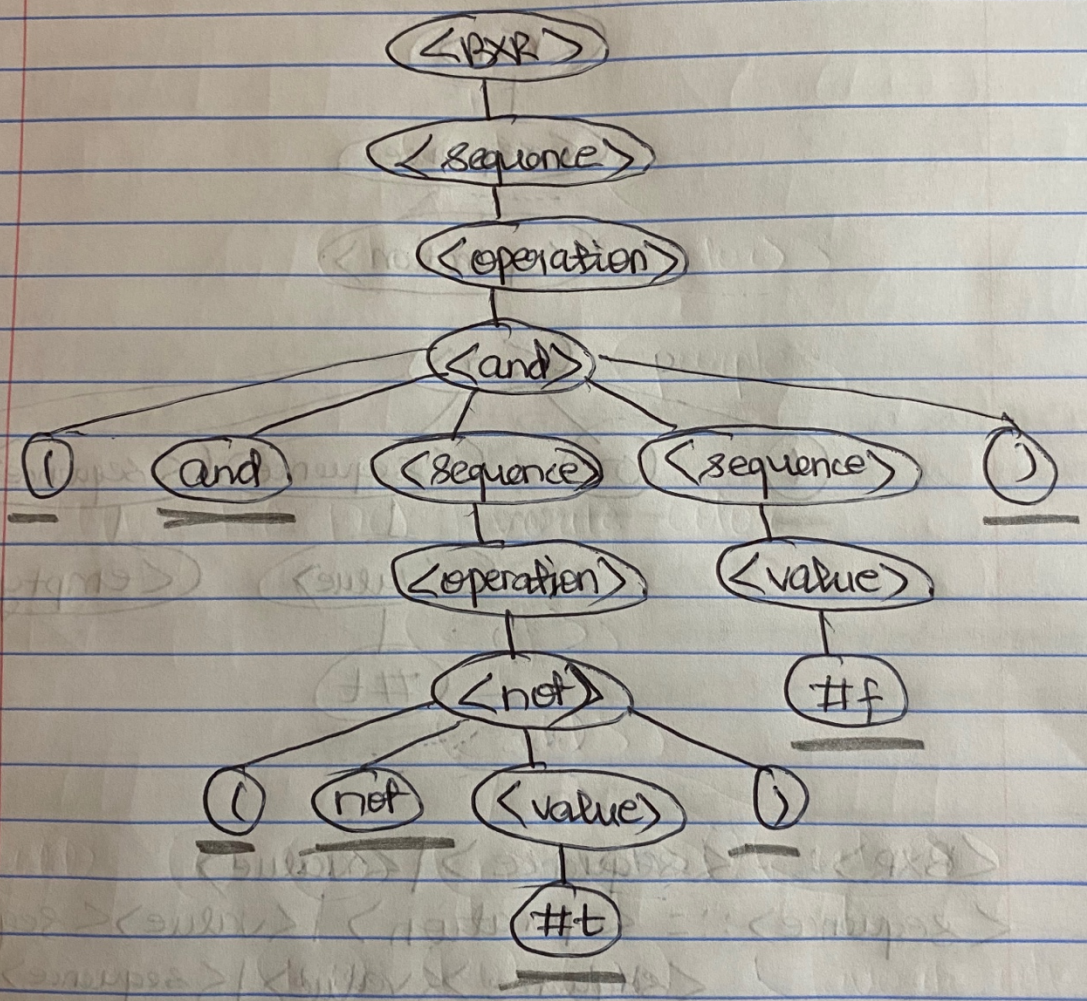


Task 4: BXR

<BXR> ::= <sequence> | <value>
<sequence> ::= <operation> | <value><sequence> | <sequence><value> |
<sequence><sequence> | <value> | <empty>
<operation> ::= <and> | <or> | <not>
<and> ::= (and <sequence><sequence>)
<or> ::= (or <sequence><sequence>)
<not> ::= (not <value>)
<value> ::= #t | #f

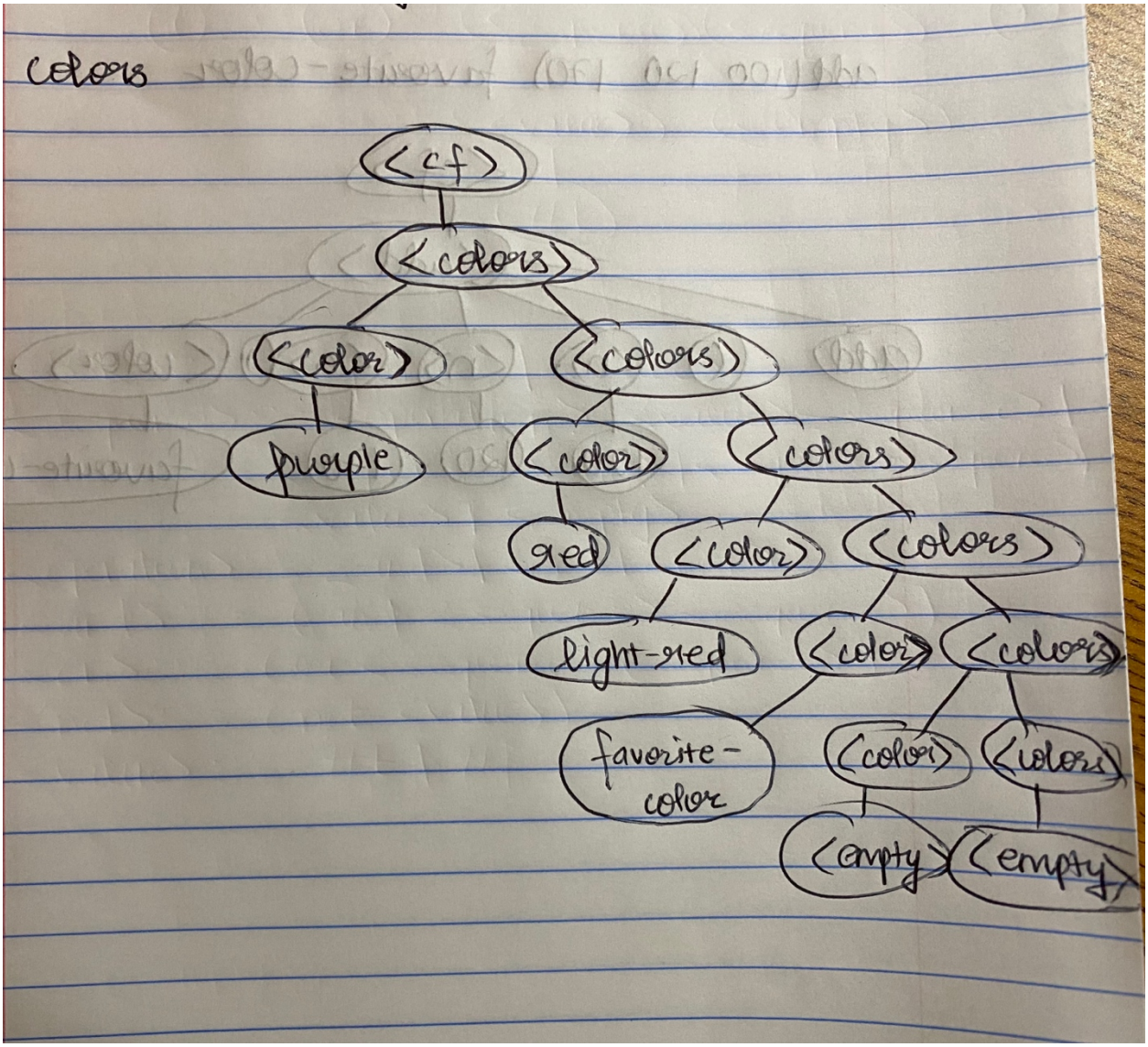


(and (not #t) #f)

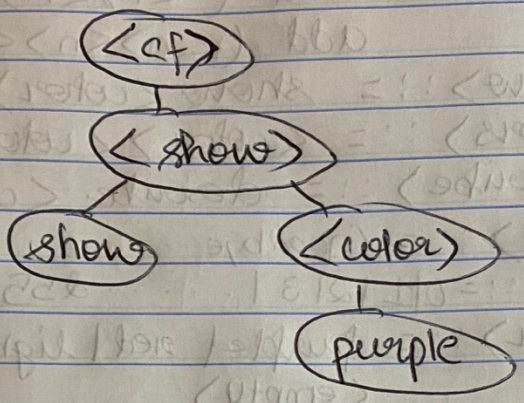

<BXR>
<sequence>
<operation>
<and>
(
and
<sequence>
<sequence>
)
<operation>
<value>
<not>
#f
(
not
<value>
)
#t

Task 5 : CF (Color Fun)

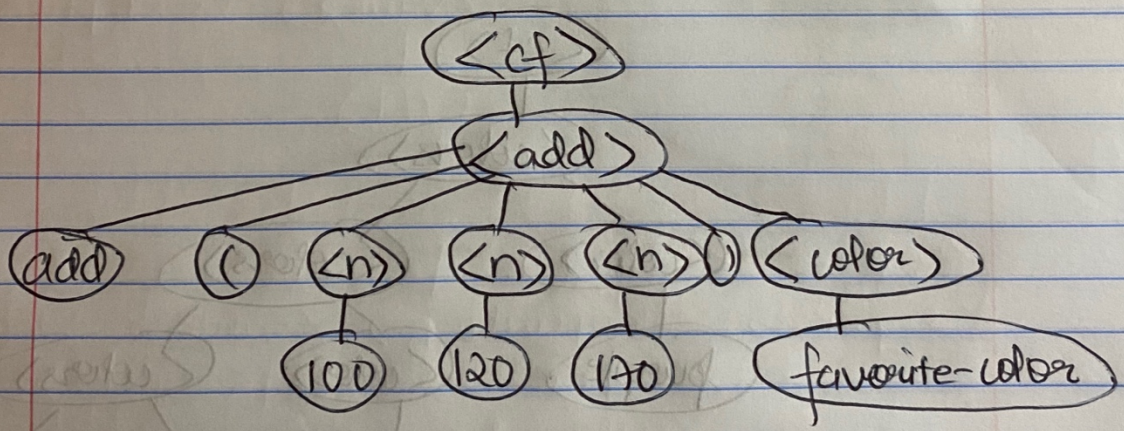
<cf> ::= <add> | <colors> | <show> | <describe> | <exit>
<add> ::= add (<n><n><n>) <color> | add (<n><n><n><n>) <color>
<show> ::= show<color>
<colors> ::= <color><colors> | <empty>
<describe> ::= describe <color>
<exit> ::= Goodbye...
<n> ::= 0|1|2|3|...255
<color> ::= purple | red | light-red | favorite-color | <empty>



show purple



add (100 120 170) favorite-color



Task 6: BNF?

BNF (the abbreviation for Backus Naur Form) is a form of context-free grammar which concisely defines the rules for the creation of a language. It sets the fundamentals for the language based on which expressions in the language can be created. The core elements of BNF include:

- Tokens, which are the part of the language being defined
- Non-terminals, which support the language being defined
- Productions, which map a non-terminal to a string of tokens and non-terminals, and
- A start symbol