BNF Assignment

Eli Fereira

Abstract

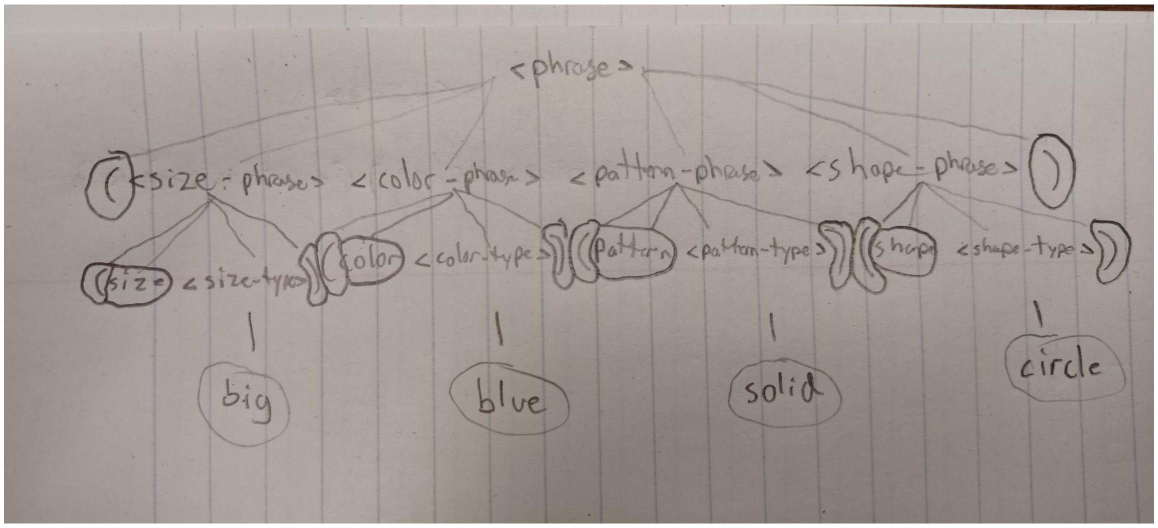
This Assignment is a good introduction to BNF, but also acts as a good introduction to learning languages in general. All five languages specified contain different syntaxes, different arguments, and different specifications. The more experience we get dealing with different syntaxes in smaller languages, the better we are equipped to learning new syntaxes in real programming languages.

Task 1: Shapes

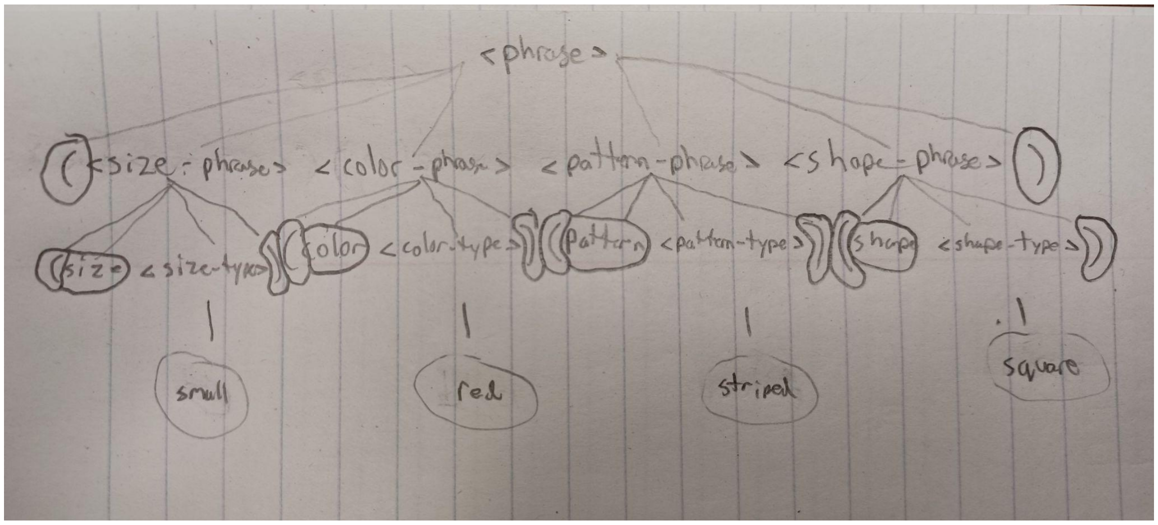
```
<statement> ::= ( <size-phrase> <color-phrase> <pattern-phrase> <shape-phrase> )

<size-phrase> ::= ( size <size-type> )
<color-phrase> ::= ( color <color-type> )
<pattern-phrase> ::= ( pattern <pattern-type> )
<shape-phrase> ::= ( shape <shape-type> )

<size-type> ::= big | medium | small
<color-type> ::= red | yellow | blue
<pattern-type> ::= striped | dotted | solid
<shape-type> ::= circle | square | triangle
```



<phrase>
(<size-phrase> <color-phrase> <pattern-phrase> <shape-phrase>)
(size <size-type>) (color <color-type>) (pattern <pattern-type>) (shape <shape-type>)
big
blue
solid
circle



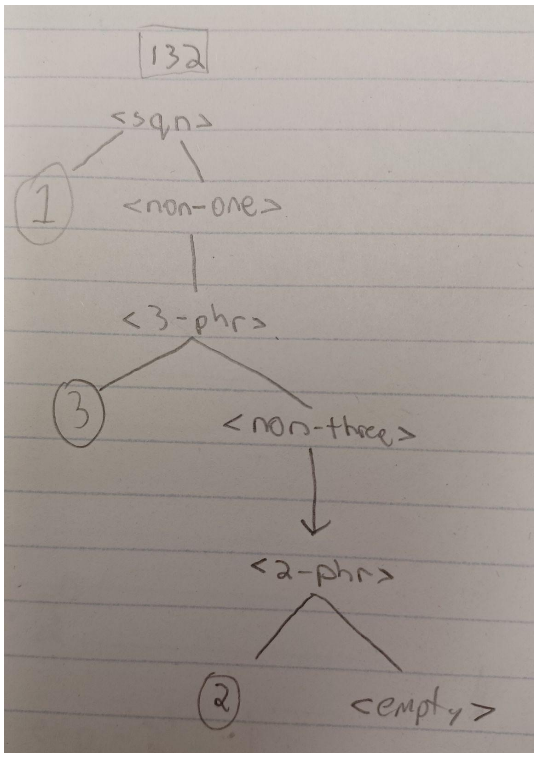
<phrase>
(<size-phrase> <color-phrase> <pattern-phrase> <shape-phrase>)
(size <size-type>) (color <color-type>) (pattern <pattern-type>) (shape <shape-type>)
small
red
striped
square

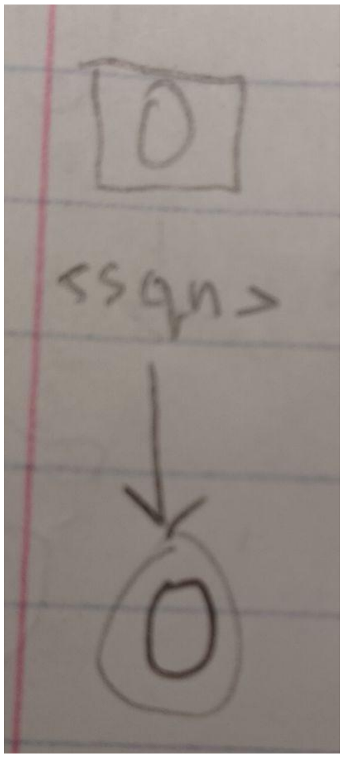
Task #2: SQN (Special Quaternary Numbers)

```
<sqn> ::= 0 | <1-phr> | <2-phr> | <3-phr>

<0-phr> ::= 0<non-zero>
<1-phr> ::= 1<non-one>
<2-phr> ::= 2<non-two>
<3-phr> ::= 3<non-three>

<non-zero>  ::= <1-phr> | <2-phr> | <3-phr> | <empty>
<non-one>   ::= <0-phr> | <2-phr> | <3-phr> | <empty>
<non-two>   ::= <1-phr> | <0-phr> | <3-phr> | <empty>
<non-three> ::= <1-phr> | <2-phr> | <0-phr> | <empty>
```





Why 1223 does not work:

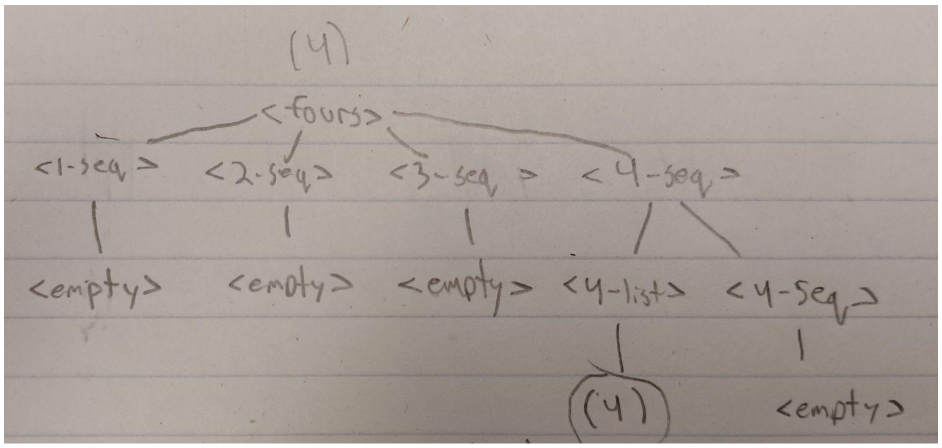
After getting `12` from the parse tree, you are left with one open branch, which is the `<non-two>` non-terminal. That terminal has no permutation that leads to a leading 2 phrase, so the input fails.

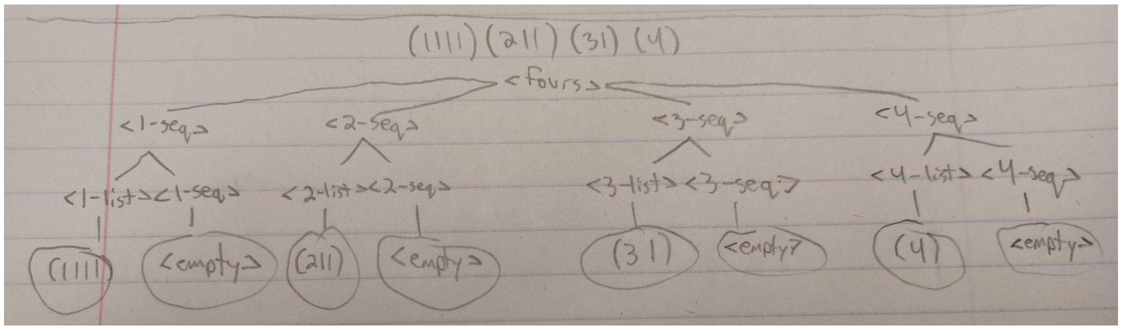
Task #3: Fours

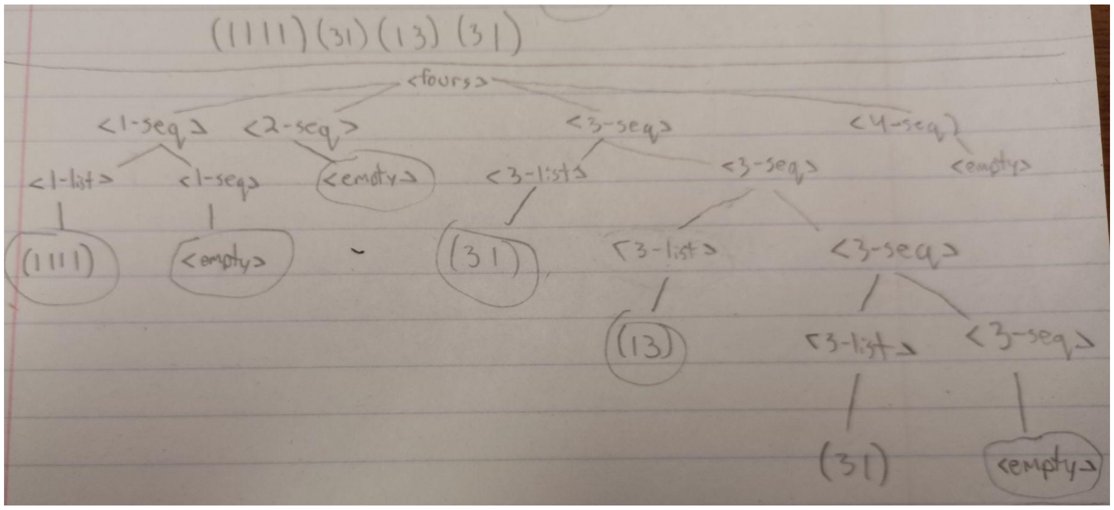
```
<fours> ::= <1-seq> <2-seq> <3-seq> <4-seq>

<1-seq> ::= <1-list> <1-seq> | <empty>
<2-seq> ::= <2-list> <2-seq> | <empty>
<3-seq> ::= <3-list> <3-seq> | <empty>
<4-seq> ::= <4-list> <4-seq> | <empty>

<1-list> ::= ( 1 1 1 1 ) | <empty>
<2-list> ::= ( 1 1 2 ) ( 1 2 1 ) ( 2 1 1 ) | <empty>
<3-list> ::= ( 3 1 ) ( 1 3 ) | <empty>
<4-list> ::= ( 4 ) | <empty>
```





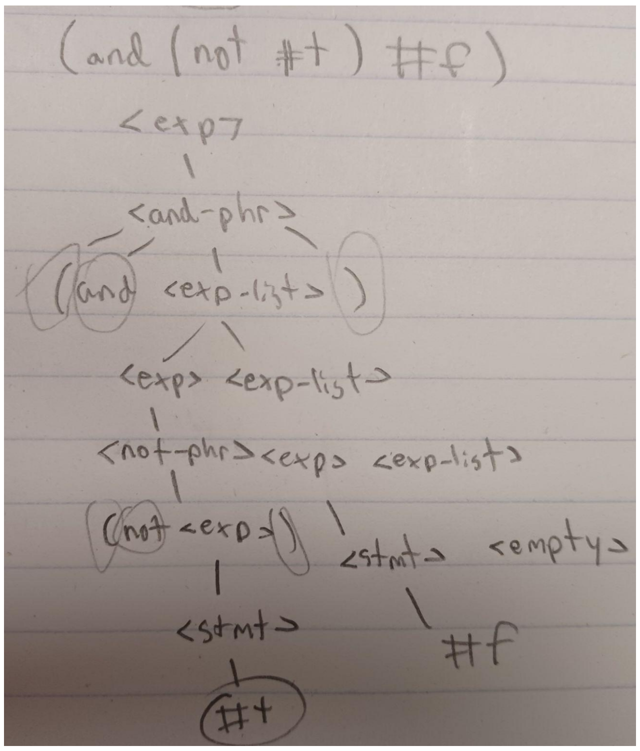


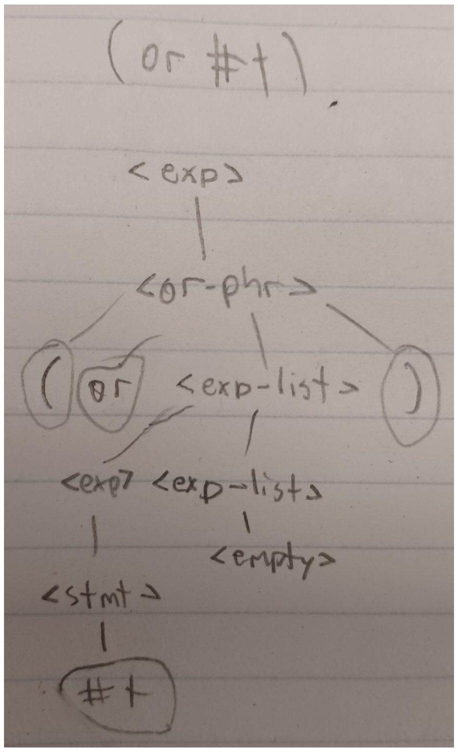
Task 4: BXR

```
<exp>      ::= <stmt> | <and-phr> | <or-phr> | <not-phr>
<exp-list> ::= <exp> <exp-list> | <empty>

<and-phr> ::= ( and <exp-list> )
<or-phr>  :: = ( or <exp-list> )
<not-phr> ::= ( not <exp> )

<stmt> ::= #t | #f
```



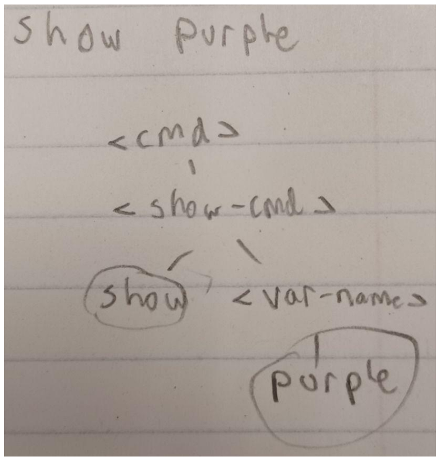


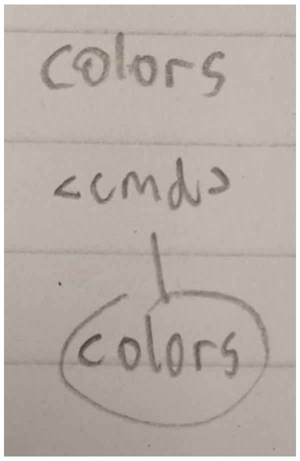
Task 5: Color Fun

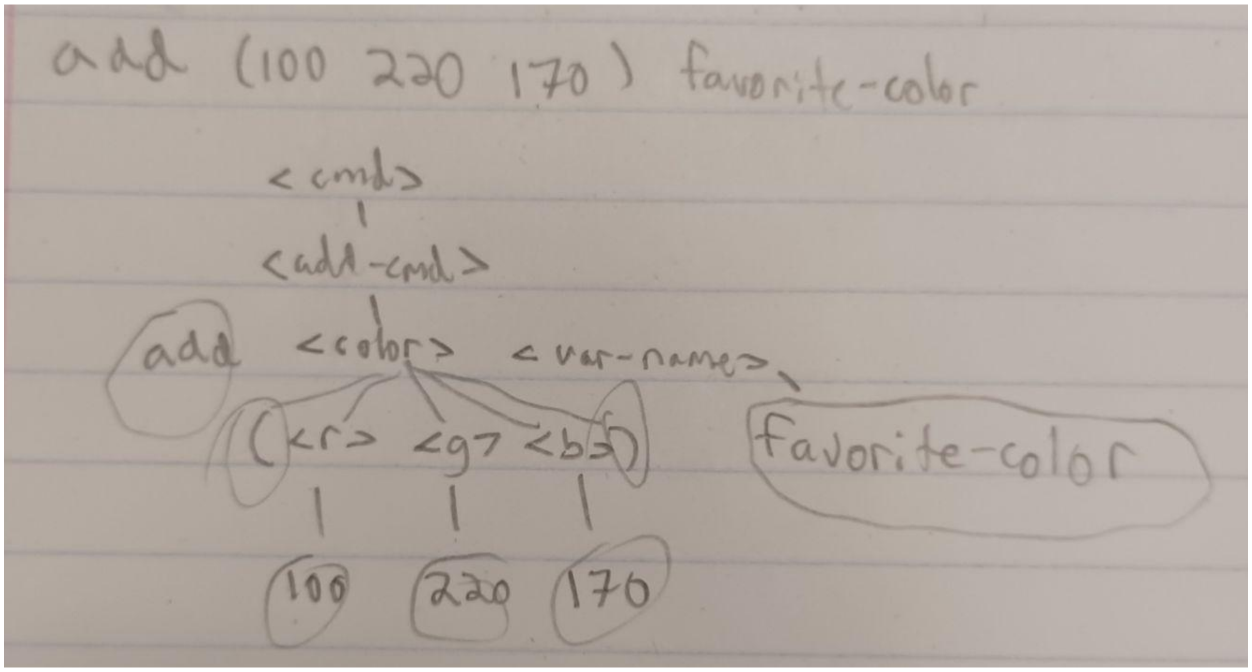
```
<cmd> ::= <add-cmd> | <show-cmd> | <desc-cmd> | colors | exit

<add-cmd>  ::= add <color> <var-name>
<show-cmd> ::= show <var-name>
<desc-cmd> ::= describe <var-name>

<color> ::= color | ( <r> <g> <b> ) | ( <r> <g> <b> <a> )
```







Task 6: BNF Description

BNF, also known as Backus-Naur Form, is a language designed for precisely describing other languages.

It consists of tokens, objects found in the languages, and variables, often called “non-terminals”. These tokens and variables are combined into a list of rules that map each token to a variable.

BNF exists to explain and demonstrate specific language features precisely, which is its advantage over explaining it in English. In fact, BNF can explain itself! Its simple design allows languages to be accurately described, regardless of complexity.