Backus-Naur Form (BNF) Assignment

Brandon LaPointe

Learning Abstract

This assignment features several compositions of Backus-Naur Form (BNF) grammars for given languages. Each specified language includes two or more BNF parse trees drawn to fit each language with given examples of sentences run through the trees. The final problem consists of describing BNF in English, in a straightforward, compelling manner.

Problem 1 - Shapes

<shapes> ::= (<size-string> <color-string> <pattern-string> <shape-string>)

<size-string> ::= (size <size>)

<color-string> ::= (color <color>)

<pattern-string> ::= (pattern <pattern>)

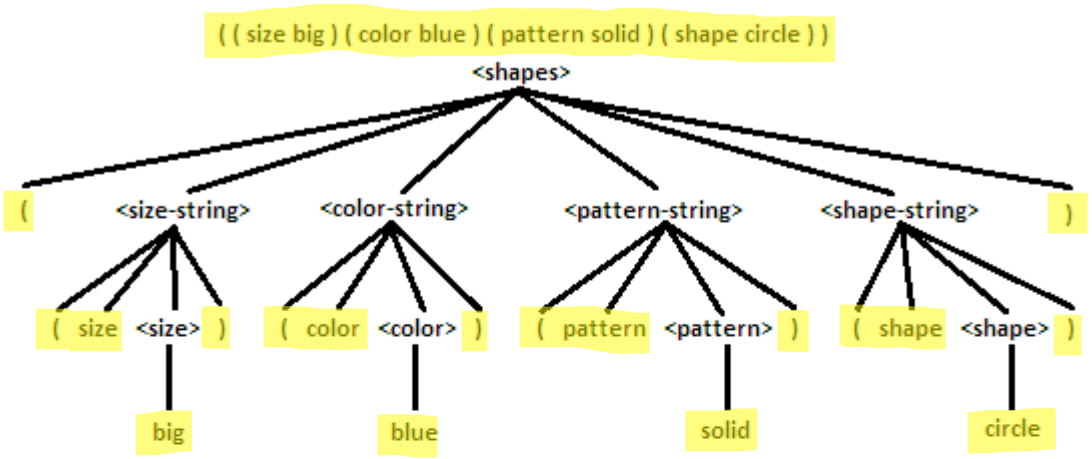
<shape-string> ::= (shape <shape>)

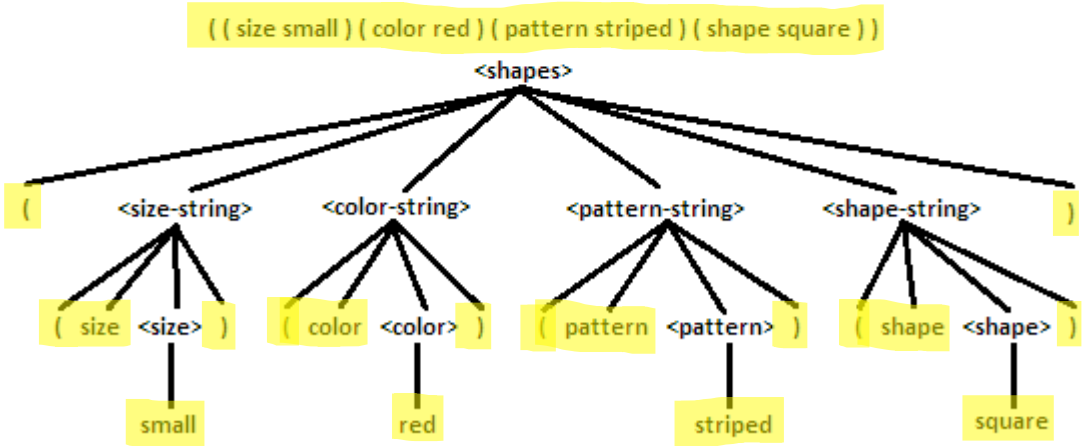
<size> ::= small | medium | big

<color> ::= red | blue | yellow

<pattern> ::= striped | dotted | solid

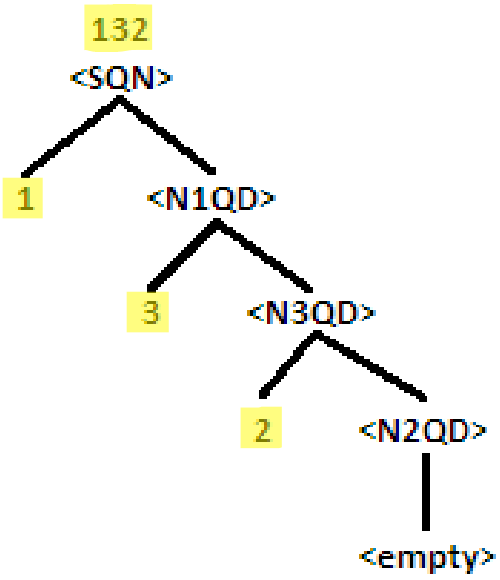
<shape> ::= circle | square | triangle





Problem 2 – Special Quaternary Numbers (SQN)

<SQN> ::= 0 | 1 <N1QD> | 2 <N2QD> | 3 <N3QD>	SQN = Special Quaternary Number
<NZQD> ::= <empty> | 1 <N1QD> | 2 <N2QD> | 3 <N3QD>	NZQD = Non-Zero Quaternary Digit
<N1QD> ::= <empty> | 0 <NZQD> | 2 <N2QD> | 3 <N3QD>	N1QD = Non-1 Quaternary Digit
<N2QD> ::= <empty> | 0 <NZQD> | 1 <N1QS> | 3 <N3QD>	N2QD = Non-2 Quaternary Digit
<N3QD> ::= <empty> | 0 <NZQD> | 1 <N1QD> | 2 <N2QD>	N3QD = Non-3 Quaternary Digit



The reason you cannot draw a parse tree for the string 1223, consistent with the BNF grammar above, is because the grammar requires a Non-2 Quaternary Digit (<empty>, 0, 1, or 3) to come after the input of a 2. Given that we have a second consecutive input of 2, the sentence is not consistent with the BNF grammar for the <SQN> language.

Problem 3 – Fours

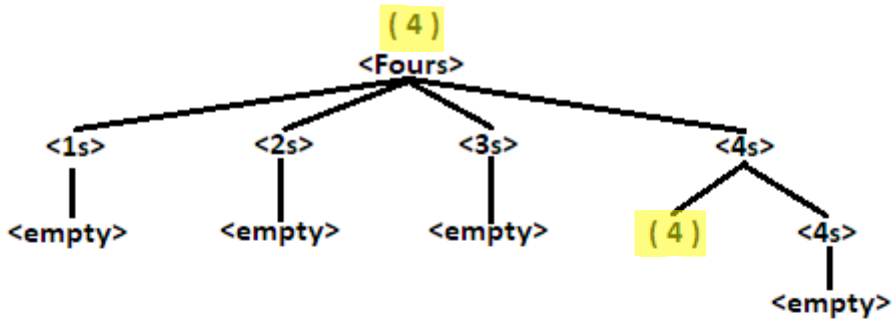
<Fours> ::= <1s> <2s3s> <4s>

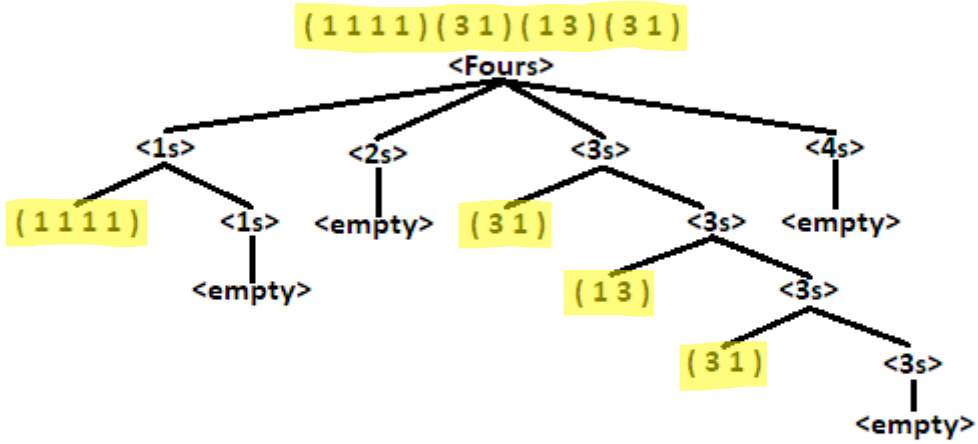
<1s> ::= <empty> | (1 1 1 1) <1s>

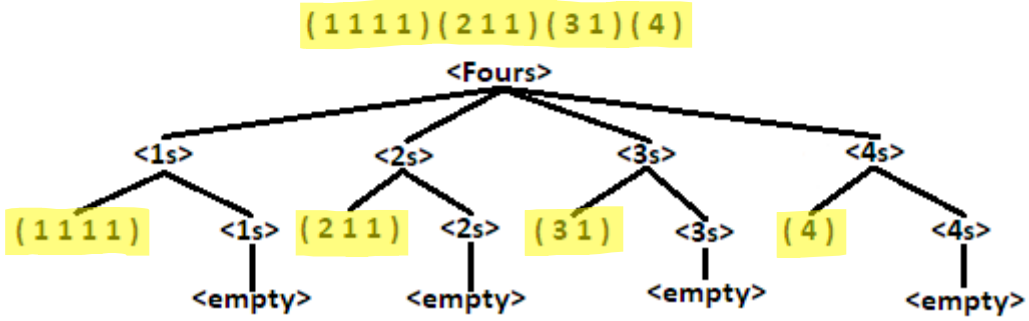
<2s> ::= <empty> | (1 1 2) <2s> | (1 2 1) <2s> | (2 1 1) <2s>

<3s> ::= <empty> | (3 1) <3s> | (1 3) <3s>

<4s> ::= <empty> | (4) <4s>







Problem 4 – BXR

<BXR> ::= <single> | <operations>

<single> ::= #t | #f | (and) | (or)

<operations> ::= <empty> | <operation> <operations>

<operation> ::= (<operator> <constant> <constants>) |

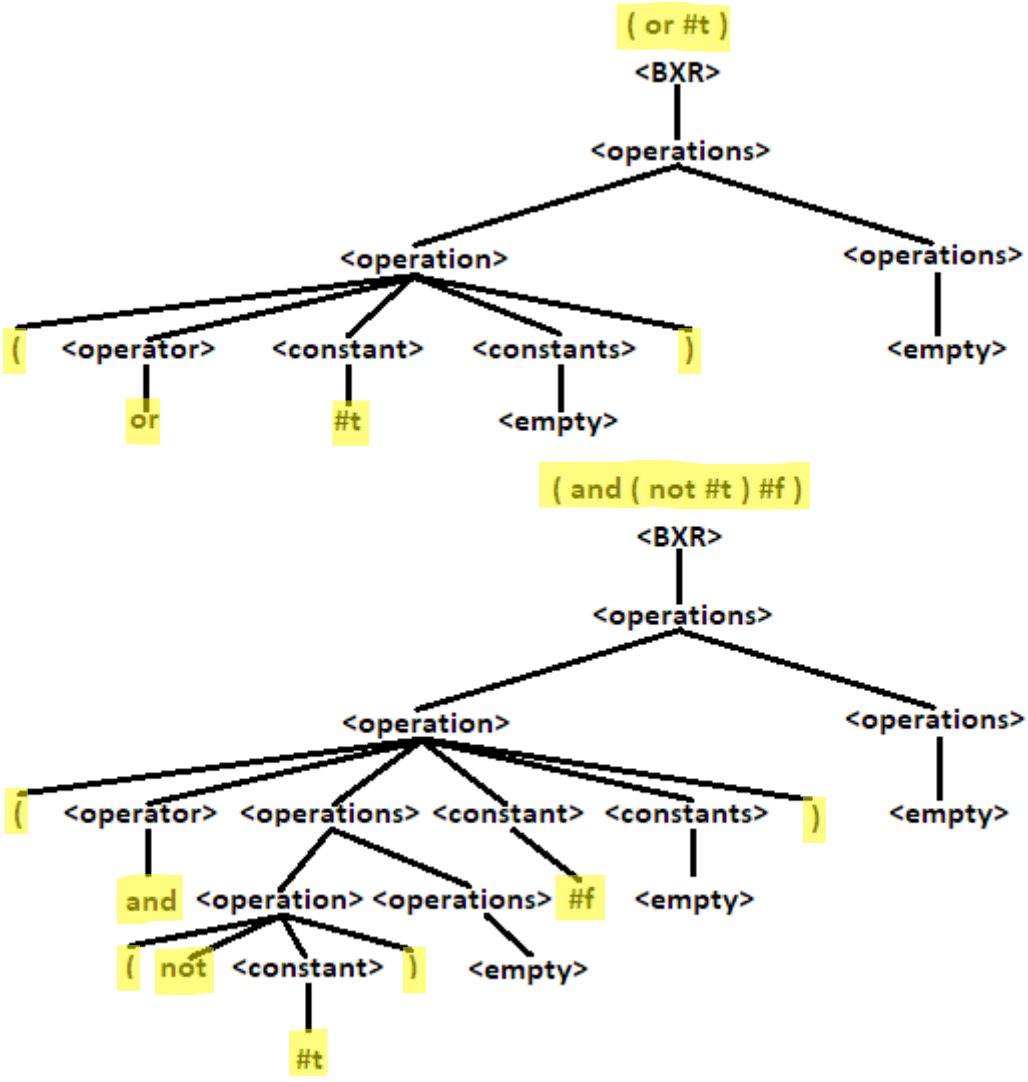
(<operator> <operations> <constant> <constants>) |

(not <constant>)

<operator> ::= or | and

<constant> ::= #f | #t

<constants> ::= <empty> | #f <constants> | #t <constants>



Problem 5 – Color Fun (CF)

<CF> ::= <add> | <show> | <describe> | colors | exit

<add> ::= add (<num> <num> <num>) <color> | add (<num> <num> <num> <num>) <color>

<show> ::= show <color>

<describe> ::= describe <color>

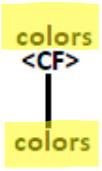
<color> ::= c1 | c2 | c3 | c4 | dark-red | red | light-red | dark-orange | orange | light-orange | dark-yellow | yellow | light-yellow |
dark-green | green | light-green | dark-blue | blue | light-blue | dark-purple | purple | light-purple | dark-pink | pink |
light-pink | dark-brown | brown | light-brown | white | dark-gray | gray | light-gray | black | favorite-color

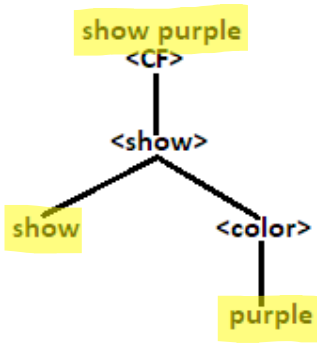
<num> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
1 <0-9> | 2 <0-9> | 3 <0-9> | 4 <0-9> | 5 <0-9> | 6 <0-9> |7 <0-9> | 8 <0-9> | 9 <0-9> |
1 <0-9> <0-9> | 2 <0-4> <0-9> | 25 <0-5>

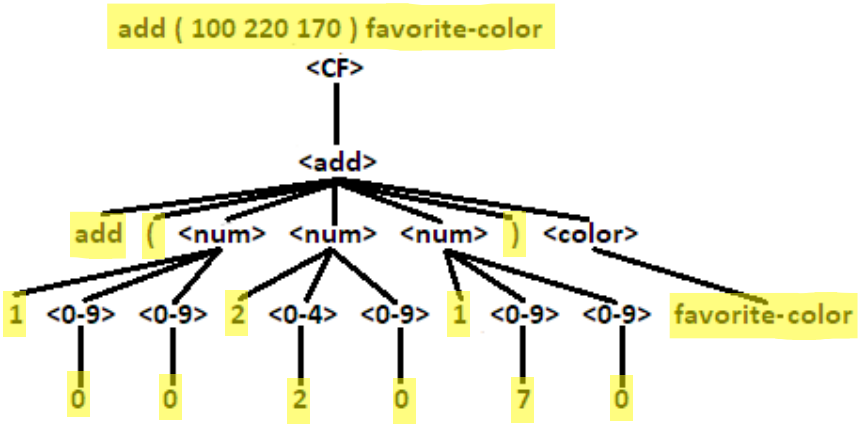
<0-9> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<0-5> ::= 0 | 1 | 2 | 3 | 4 | 5

<0-4> ::= 0 | 1 | 2 | 3 | 4







Problem 6 – BNF?

BNF, or Backus-Naur Form, is a formal way of describing the syntax of a language invented by John Backus and Peter Naur. It is a common method for developers of programming languages to specify the rules of a language in order to make sure that everyone is using a consistent arrangement of vocabulary. BNF uses a wide variety of symbols and expressions to create well-defined production rules of which the language must follow.