Thinh Nguyen
CSC135

P L H.W Assignment 1

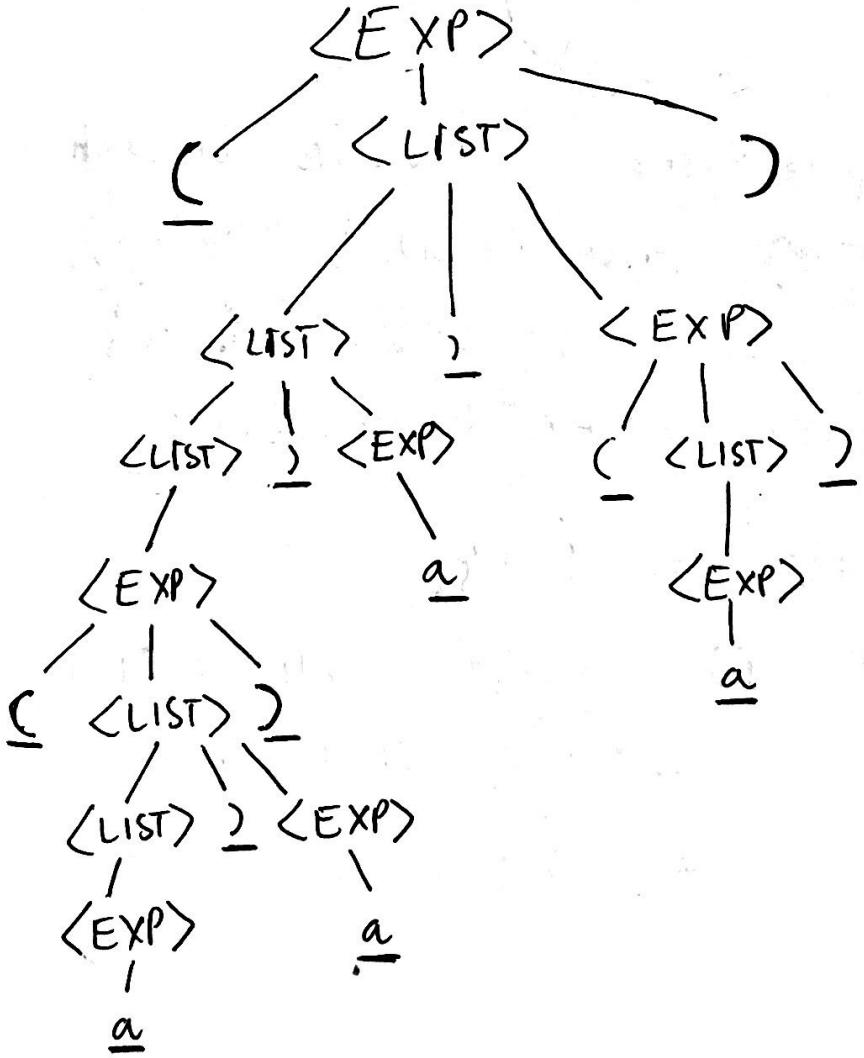
Part2

#1 BNF Grammar:

EXP::= (LIST) | a

LIST ::= LIST, EXP | EXP

a) Draw a parse tree for ((a,a), a, (a))



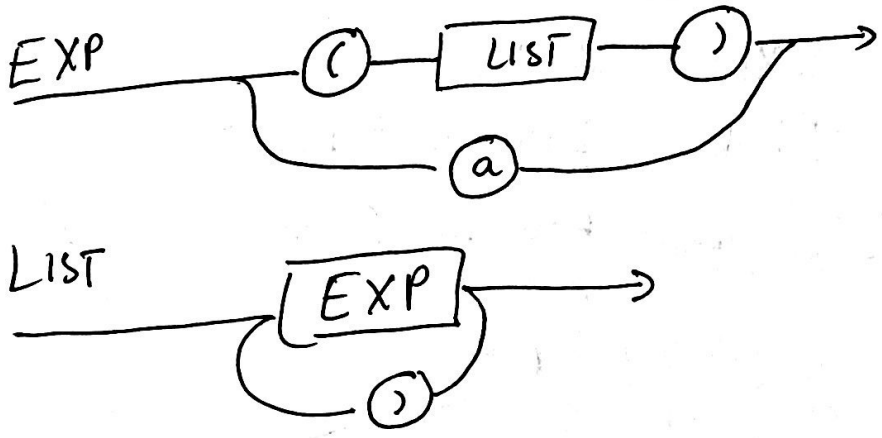
(b) Translate to EBNF:

EBNF:

EXP ::= (LIST) | a

LIST ::= EXP {, EXP}

(c) Draw syntax diagrams



(d) Compute First and Follow sets for each of the non-terminals

First (EXP) = First ((LIST)) ∪ First (a)

= { (} ∪ { a } = { (, a }

First (LIST) = First (EXP) = { (, a }

Follow (LIST) = {) }

Follow (EXP) = { , } ∪ Follow (LIST)

= { , } ∪ {) }

= { , ,) }

#2 BNF:

EXP ::= EXP + TERM | EXP - TERM | TERM

TERM ::= TERM * FACTOR | TERM / FACTOR | FACTOR

FACTOR ::= (EXP) | DIGIT

DIGIT ::= 0 | 1 | 2 | 3

a) EBNF:

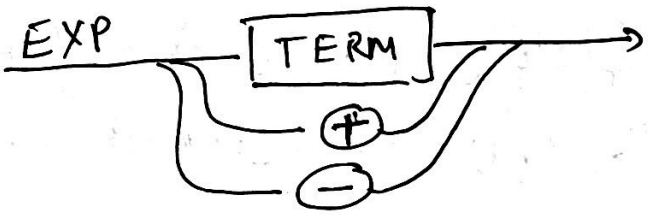
EXP ::= TERM { (+ | -) TERM }

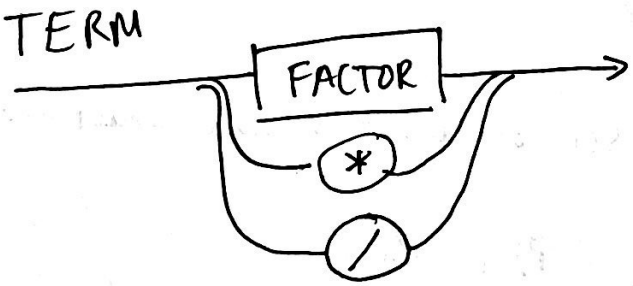
TERM ::= FACTOR { (* | /) FACTOR }

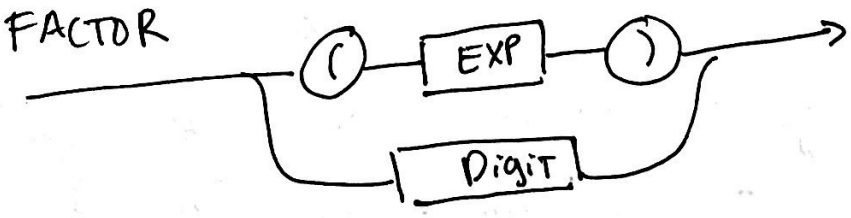
FACTOR ::= (EXP) | DIGIT

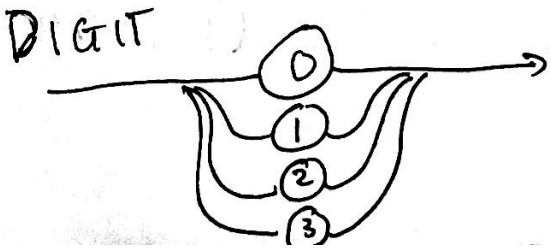
DIGIT ::= 0 | 1 | 2 | 3

b) Syntax diagrams:









c) Two requirements on a grammar for a predictive parser:

1. The branches (First sets) lead to different items within the rule,
2. One branch leads to an item within the rule, and the other branch exits the rule. ($First(A) \cap Follow(A) = \emptyset$)

d)

First (DIGIT) $= \{0, 1, 2, 3\}$

First (FACTOR) $= \{(, 0, 1, 2, 3\}$

First (TERM) = First (FACTOR) $= \{(, 0, 1, 2, 3\}$

First (EXP) = First (TERM) $= \{(, 0, 1, 2, 3\}$

Follow (EXP) $= \{)\}$

Follow (TERM) $= \{+, -\} \cup \{$ Follow(EXP)$\} = \{+, -,)\}$

Follow (FACTOR) $= \{*, /\} \cup \{$ Follow (TERM)$\} = \{*, /, +, -,)\}$

Follow (DIGIT) = Follow (FACTOR) $= \{*, /, +, -,)\}$

e) Prove:

1. Condition (First): First sets of any two choices must not have any tokens in comment.

Example: FACTOR ::= (EXP) | DIGIT

First (EXP) $\cap$ First (DIGIT) $= \{(\} \cap \{0, 1, 2, 3\}$

$= \emptyset$

2. Condition (Second) : $First(A) \cap Follow(A) = \emptyset$

Example: TERM ::= FACTOR {(*| /) FACTOR}

First (FACTOR) $\cap$ Follow (FACTOR) $= \{(\} \cap \{)\}$

$= \emptyset$

Recursive Descent Recognizer Pseudocode

```
procedure exp()
{      term();
       if (token == '+')
       { match('+');
         term();
       }
      else if  (token =='-')
        { match('-');
          term();
        }
       else break;
}

procedure term()
{       factor();
      if (token == '*')
        { match('*');
          factor();
        }
else if (token == '/')
         { match ('/');
          factor();
          }
else break;
}

procedure factor()
```

```
{
  if (token == '(')
      { match ('(');
      exp();
        match (')');
       }
       else digit();
}

procedure digit()
{
      if (token in [0,1,2,3])
      {
       match(token);
       match ($);
       }
       else error;
}

match(t)
{
if (token==t)
 {
advanceTokenPtr;
}
  else error;
}
```