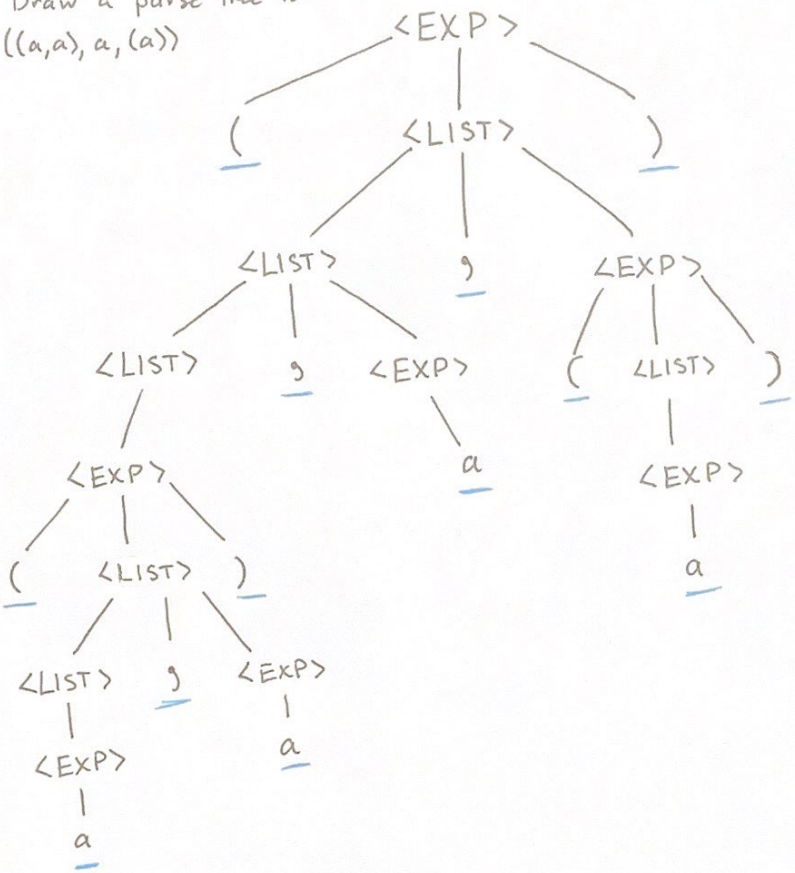
1) BNF Grammar

EXP ::= (LIST) | a

LIST ::= LIST , EXP | EXP

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



((a,a), a, (a))

b. Translate BNF to EBNF

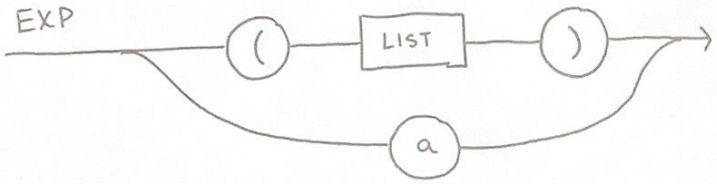
EXP ::= (LIST) | a
LIST ::= LIST , EXP | EXP

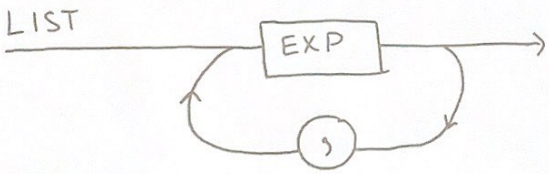
} Original

EXP ::= (LIST) | a
LIST ::= EXP {, EXP}

} Converted

c. Draw syntax diagrams





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

$$\text{First}(\text{EXP}) = \text{First}(\text{(LIST)}) \cup \text{First}(a)$$
$$= \{(\} \cup \{a\}$$
$$= \{(, a\}$$

$$\text{First}(\text{LIST}) = \text{First}(\text{EXP})$$
$$= \{(, a\}$$

$$\text{Follow}(\text{LIST}) = \{)\}$$

$$\text{Follow}(\text{EXP}) = \{,\} \cup \text{Follow}(\text{LIST})$$
$$= \{,\} \cup \{)\}$$
$$= \{,,)\}$$

2. Consider the following BNF grammar

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

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

FACTOR ::= (EXP) | DIGIT

DIGIT ::= 0 | 1 | 2 | 3

a. Translate into EBNF.

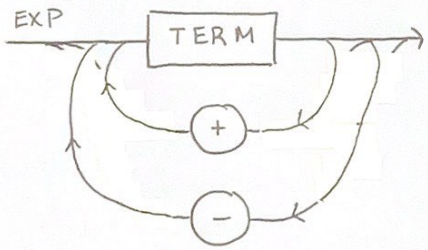
EXP ::= TERM {(+|-) TERM}

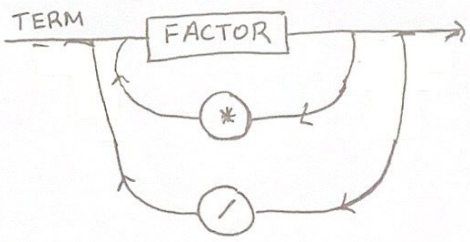
TERM ::= FACTOR {(*|/) FACTOR}

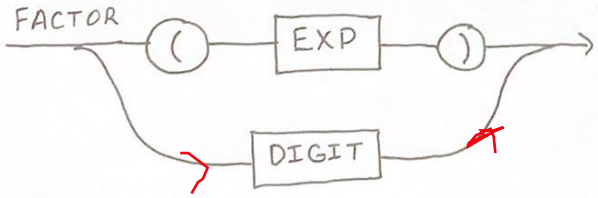
FACTOR ::= (EXP) | DIGIT

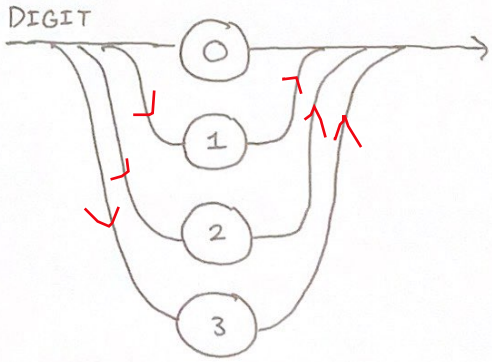
DIGIT ::= 0 | 1 | 2 | 3

b. Draw syntax diagrams.









-1%: Missing arrows.

C. What are the two requirements on a grammar for a predictive parser to be able to make right choice?

(1) The branches (first sets) lead to different items within the rule

e.g. Given $A \rightarrow B \mid C$

$First(B) \cap First(C) = \phi$

(2) One branch leads to an item within the rule and the other branch exits the rule.

eg. Given $A \rightarrow D[E]F$

$First(E) \cap Follow(E) = \phi$

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

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

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

FACTOR ::= (EXP) | DIGIT

DIGIT ::= 0|1|2|3

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

First(FACTOR) = First(FACTOR) ∪ 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) = {)}

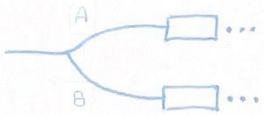
Follow(TERM) = {+, -} ∪ {Follow(EXP)} = {+, -,)}

Follow(FACTOR) = {*, /} ∪ {Follow(TERM)} = {*, /, +, -,)}

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

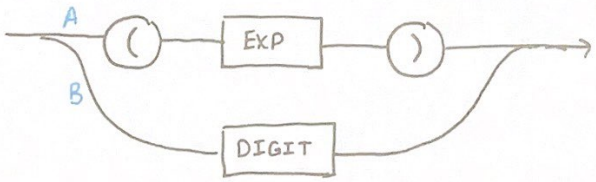
e. Prove that the grammar satisfy the two requirements defined in (c).

Condition 1: $First(A) \cap First(B) = \phi$



From our grammar: FACTOR ::= (EXP) | DIGIT

Syntax Diagram:



$First(A) \cap First(B) = \phi$

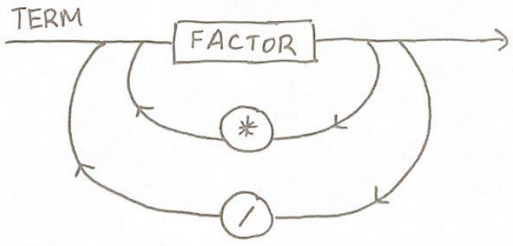
$\Rightarrow First(\,(\ EXP\ )\,) \cap First(DIGIT) = \phi$

$\Rightarrow \{(\} \cap \{0, 1, 2, 3\} = \phi$

Condition 2: $First(c) \cap Follow(x) = \phi$

From our grammar: TERM ::= FACTOR {(*|/) FACTOR}

Syntax Diagram:



$First(c) \cap Follow(x) = \phi$

$\Rightarrow First(FACTOR) \cap Follow(FACTOR) = \phi$

~~$\Rightarrow \{(, 0, 1, 2, 3\} \cap \{*, +, -,)\} = \phi$~~

$\Rightarrow \{(, 0, 1, 2, 3\} \cap \{*, /, +, -,)\} = \phi$

Refering to problem (2.d.):
$First(FACTOR) = \{(, 0, 1, 2, 3\}$
$Follow(FACTOR) = \{*, /, +, -,)\}$
Derived in previous exercise

3. EBNF Given:

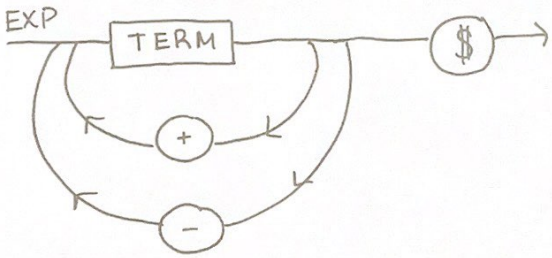
EXP ::= TERM { (+|−) TERM }

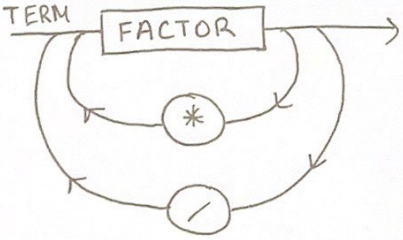
TERM ::= FACTOR { (*|/) FACTOR }

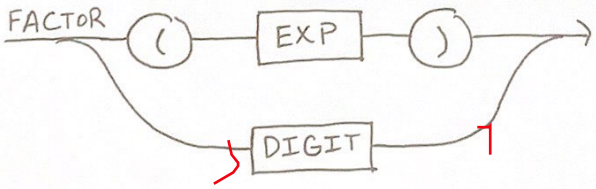
FACTOR ::= (EXP) | DIGIT

DIGIT ::= 0 | 1 | 2 | 3

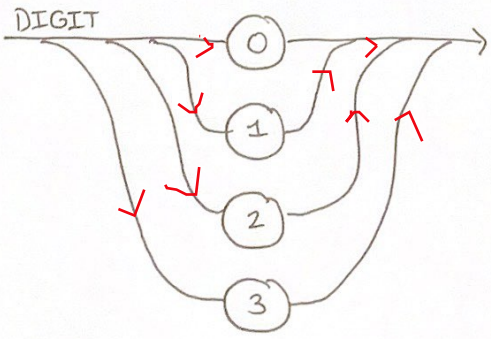
Syntax Diagrams:







-1%: Missing arrows.



Recursive - Descent Pseudocode:

```
EXP( )
   TERM()
   if (token == '+')
      match('+')
      TERM();
   else if (token == '-')
      match('-')
      TERM()
   else if (token == '$')
      match('$')
      break
   else
      break
```

```
TERM()
   FACTOR();
   if(token == '*')
     match('*')
     FACTOR()
   else if(token == '/')
     match('/')
     FACTOR()
   else
     break
```

```
FACTOR()
   if(token == '(')
     match('(')
     EXP()
     if(token == ')')
       match(')')
     else
       break
   else
     DIGIT()
```

```
DIGIT()
   if (token in [0,1,2,3])
     match(token)
   else
     error
```

```
match(t)
   if ( token == t)
      advanceTokenPtr
   else
      error
```

Legal Test Cases:

1) String = 1 + 3 $

EXP() → TERM → FACTOR → DIGIT → 1

1 → EXP → +

+ → TERM → FACTOR → DIGIT → 3

3 → EXP → $

∴ String 1+3 $ is valid

String Ptr Status

1 + 3 $
↑

1 + 3 $
  ↑

1 + 3 $
  ↑

1 + 3 $
    ↑

---

2) String = (1+3)*(2+1)$

EXP → TERM → FACTOR → (

(→ EXP → TERM → FACTOR → DIGIT → 1

1 → EXP → +

+ → TERM → FACTOR → DIGIT → 3

3 → FACTOR →)

) → TERM → *

* → FACTOR → (

(→ EXP → TERM → FACTOR → DIGIT → 2

2 → EXP → +

+ → TERM → FACTOR → DIGIT → 1

1 → FACTOR →)

) → TERM → EXP → $

∴ String (1+3)*(2+1)$ is valid

String Ptr Status

(1+3)*(2+1)$
↑

(1+3)*(2+1)$
 ↑

(1+3)*(2+1)$
  ↑

(1+3)*(2+1)$
   ↑

(1+3)*(2+1)$
    ↑

(1+3)*(2+1)$
     ↑

(1+3)*(2+1)$
      ↑

(1+3)*(2+1)$
       ↑

(1+3)*(2+1)$
        ↑

(1+3)*(2+1)$
         ↑

(1+3)*(2+1)$
          ↑

Illegal Test Cases

1) String = 1++3$

EXP → TERM → FACTOR → DIGIT → 1

1 → EXP → +

+ → TERM → FACTOR → DIGIT → error

∴ String = 1++3$ is invalid

String Ptr Status

1++3$
↑

1++3$
 ↑

1++3$
  ↑

2) String = (1+3$

EXP → TERM → FACTOR → (

(→ EXP → TERM → FACTOR → DIGIT → 1

1 → EXP → +

+ → TERM → FACTOR → DIGIT → 3

3 → FACTOR → break → EXP → break

∴ String (1+3$ is invalid

String Ptr Status

(1+3$
↑

(1+3$
 ↑

(1+3$
  ↑

(1+3$
   ↑

(1+3$
    ↑

3) String = 10+3$

EXP → TERM → FACTOR → DIGIT → 1

1 → TERM → break → EXP → break

∴ String 10+3$ is invalid

String Ptr Status

10+3$
↑

10+3$
 ↑