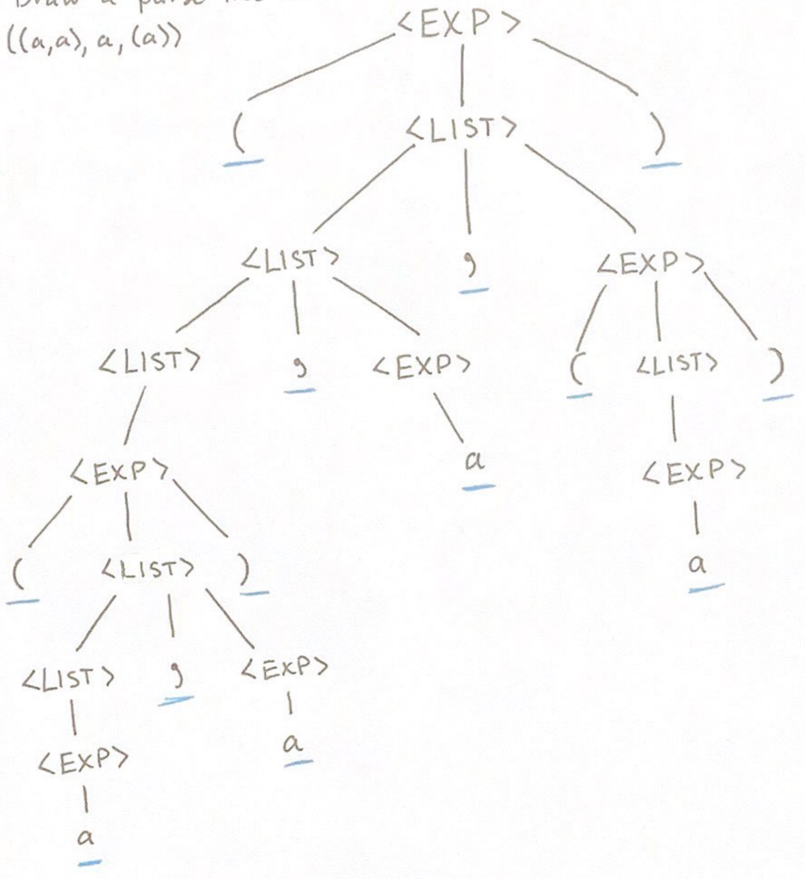
1) BNF Grammar

EXP ::= (LIST) | a

LIST ::= LIST , EXP | EXP

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



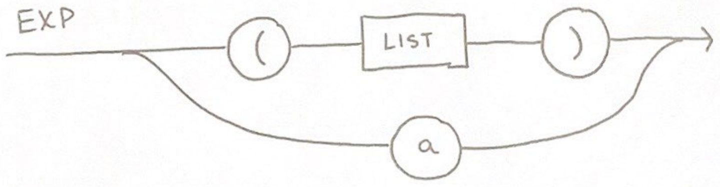
((a, a), a, (a))

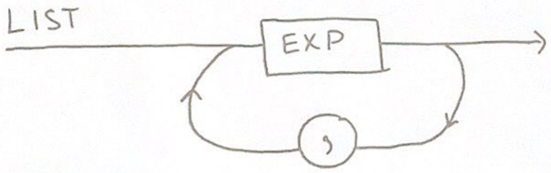
b. Translate BNF to EBNF

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

$$\begin{aligned}
\text{First}(\text{EXP}) &= \text{First}((\,\text{LIST}\,)) \cup \text{First}(a) \\
&= \{ (\} \cup \{a\} \\
&= \{ (\,,\ a \}
\end{aligned}$$

$$\begin{aligned}
\text{First}(\text{LIST}) &= \text{First}(\text{EXP}) \\
&= \{ (\,,\ a \}
\end{aligned}$$

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

$$\begin{aligned}
\text{Follow}(\text{EXP}) &= \{ ; \} \cup \text{Follow}(\text{LIST}) \\
&= \{ ; \} \cup \{) \} \\
&= \{ ;\,,\ ) \}
\end{aligned}$$

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.

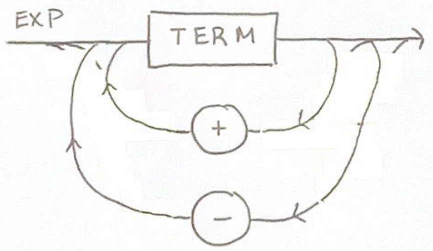
EXP ::= TERM {(+|−) TERM}

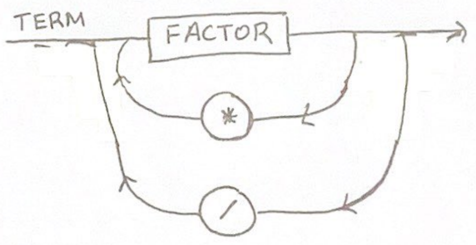
TERM ::= FACTOR {(*|/) FACTOR}

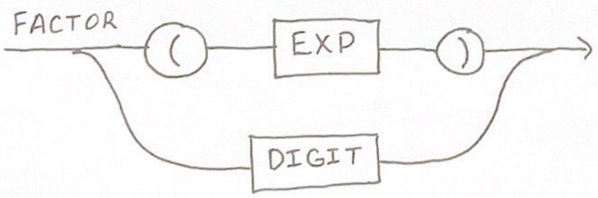
FACTOR ::= (EXP) | DIGIT

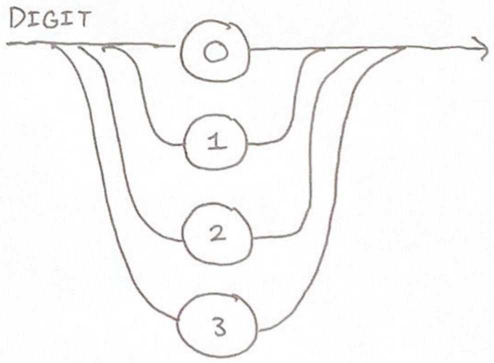
DIGIT ::= 0|1|2|3

b. Draw syntax diagrams.









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 $A \rightarrow B \mid C$

$\text{First}(B) \cap \text{First}(C) = \emptyset$

(2) One branch leads to an item within the rule and the other branch exit's the rule.

e.g. Given $A \rightarrow D[E]F$

$\text{First}(E) \cap \text{Follow}(E) = \emptyset$

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) U First(DIGIT)

= { (} U {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) = {) }

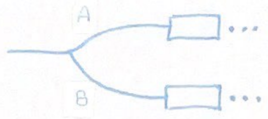
Follow(TERM) = {+, -} U {Follow(EXP)} = {+, -,)}

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

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

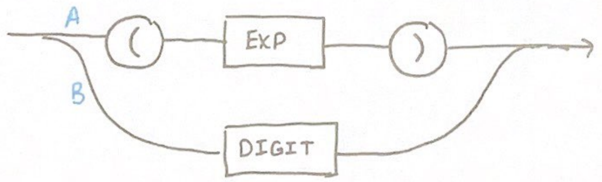
e. Prove that the grammar satisfy the two requirements defined in (c).

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



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

Syntax Diagram:

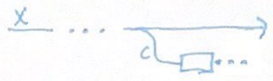


First(A) $\cap$ First(B) = $\emptyset$

$\Rightarrow$ First((EXP)) $\cap$ First(DIGIT) = $\emptyset$

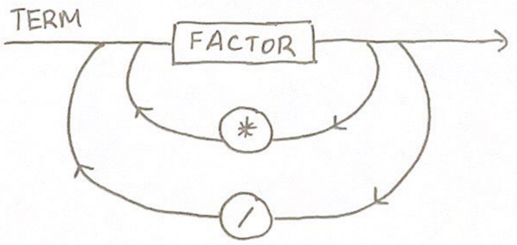
$\Rightarrow$ { (} $\cap$ {0, 1, 2, 3} = $\emptyset$

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



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

Syntax Diagram:



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

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

~~$\Rightarrow$ {[illegible]} $\cap$ {*, /, +, -,)} = $\emptyset$~~

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

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

3. EBNF Given:

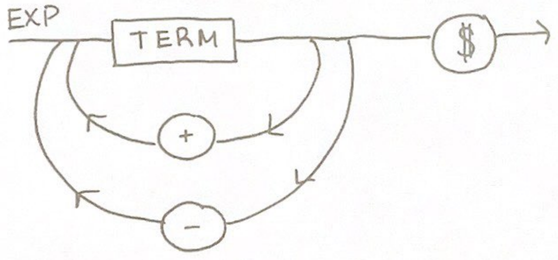
EXP ::= TERM { (+|−) TERM }

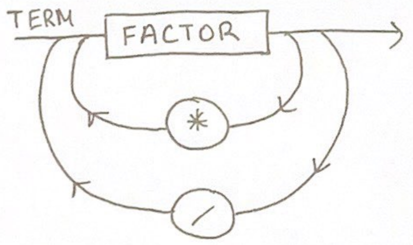
TERM ::= FACTOR { (*|/) FACTOR }

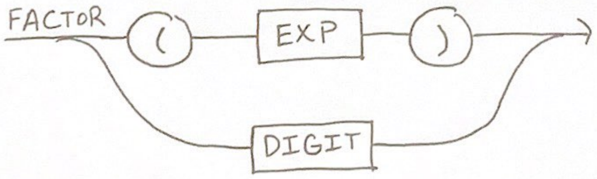
FACTOR ::= (EXP) | DIGIT

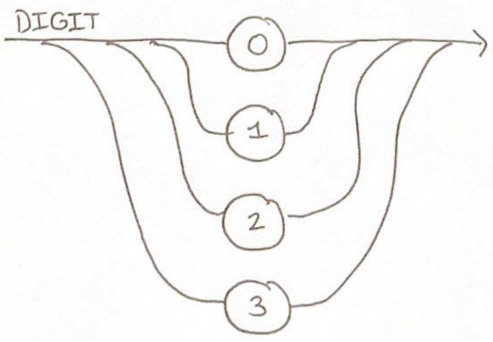
DIGIT ::= 0|1|2|3

Syntax Diagrams:









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

$\underset{\uparrow}{1} + 3 \$$

$1 \underset{\uparrow}{+} 3 \$$

$1 + \underset{\uparrow}{3} \$$

$1 + 3 \underset{\uparrow}{\$}$

---

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

$\underset{\uparrow}{(}1+3)*(2+1) \$$

$(\underset{\uparrow}{1}+3)*(2+1) \$$

$(1\underset{\uparrow}{+}3)*(2+1) \$$

$(1+\underset{\uparrow}{3})*(2+1) \$$

$(1+3\underset{\uparrow}{)}*(2+1) \$$

$(1+3)\underset{\uparrow}{*}(2+1) \$$

$(1+3)*\underset{\uparrow}{(}2+1) \$$

$(1+3)*(\underset{\uparrow}{2}+1) \$$

$(1+3)*(2\underset{\uparrow}{+}1) \$$

$(1+3)*(2+\underset{\uparrow}{1}) \$$

$(1+3)*(2+1\underset{\uparrow}{)} \$$

$(1+3)*(2+1) \underset{\uparrow}{\$}$

Illegal Test Cases

1) String = 1++3$

	String Ptr Status
EXP → TERM → FACTOR → DIGIT → 1	$\underset{\uparrow}{1}{+}{+}3\$$
1 → EXP → +	$1\underset{\uparrow}{+}{+}3\$$
+ → TERM → FACTOR → DIGIT → error	$1{+}\underset{\uparrow}{+}3\$$

∴ String = 1++3$ is invalid

2) String = (1+3$

	String Ptr Status
EXP → TERM → FACTOR → (	$\underset{\uparrow}{(}1{+}3\$$
(→ EXP → TERM → FACTOR → DIGIT → 1	$(\underset{\uparrow}{1}{+}3\$$
1 → EXP → +	$(1\underset{\uparrow}{+}3\$$
+ → TERM → FACTOR → DIGIT → 3	$(1{+}\underset{\uparrow}{3}\$$
3 → FACTOR → break → EXP → break	$(1{+}3\underset{\uparrow}{\$}$

∴ String (1+3$ is invalid

3) String = 10+3$

	String Ptr Status
EXP → TERM → FACTOR → DIGIT → 1	$\underset{\uparrow}{1}0{+}3\$$
1 → TERM → break → EXP → break	$1\underset{\uparrow}{0}{+}3\$$

∴ String 10+3$ is invalid