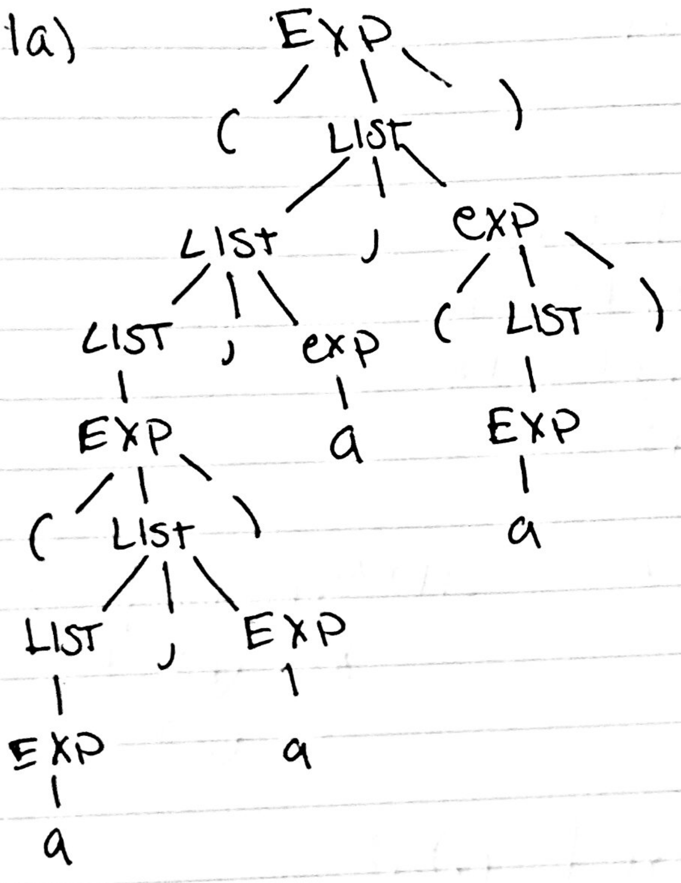
Jaskarn Jagpal
CSC 135 T,TH 130-245
Assignment 1

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

1a)

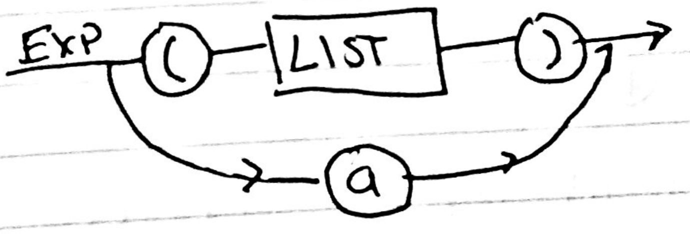


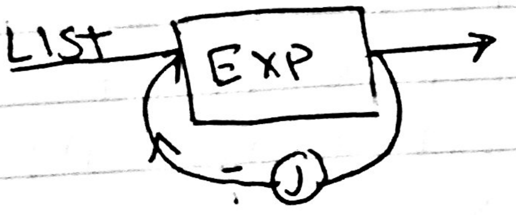
1d) FIRST(EXP) = { a, (}
~~FIRST(LIST) = {) }~~
FIRST(LIST) = FIRST(EXP)
= { a, (}
FOLLOW(LIST) = {) }

FOLLOW(EXP) = FOLLOW(LIST) ∪ { , }
= {), }

1b) EBNF
EXP::= (LIST) {a}
LIST::= <exp> {, exp}

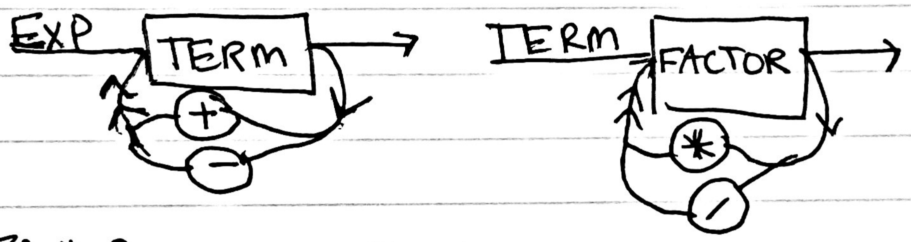
1c)

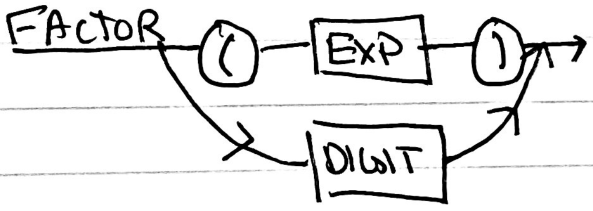


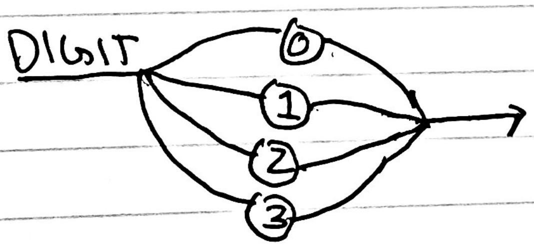


2) EXP::= EXP+TERM | EXP-TERM | TERM
TERM::= TERM * FACTOR | TERM / FACTOR | FACTOR
FACTOR::= (EXP) | DIGIT
DIGIT::= 0|1|2|3

2a) <u>EBNF</u>
EXP::=TERM{(+|-)TERM}
TERM::= FACTOR{(*|/) FACTOR}
FACTOR::= (EXP) | DIGIT
DIGIT::= 0|1|2|3

2b)






2c) First set of any two choices must not have any tokens in common.
ex) FACTOR::=(EXP) | DIGIT
First(EXP) ∩ FIRST(DIGIT) = ∅

- When structures are optional
  ex) S → B[A]D
  If A is optional, then
  FIRST(A) ∩ FOLLOW(A) = ∅

2d) FIRST(DIGIT) = { 0 1 2 3 }

FIRST (FACTOR) = FIRST(EXP) ∪ FIRST(DIGIT)

= { (} ∪ { 0 1 2 3 } = { (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) = { * / + -) }

2e) FIRST(DIGIT) ∩ FOLLOW(DIGIT) =

{ 0 1 2 3 } ∩ { * / + -) } = ∅

)

```
Exp()
    Term()
    If token == '+'
        Match('+')
    Else if token == '-'
        Match('-')
    Term()
    Match($)
```

```
Digit()
    If token in [0, 1, 2, 3]
        Match(token)
    Else
  Error
```

```
Term()
    Factor()
    If token == '*'
        Match('*')
    Else if token =='/'
        Match('/')
    Factor()
    Match($)
```

```
Factor()
    If token =='('
        Match('(')
        Exp()
        If token == ')'
            Match(')')
    else
        Digit()
    Match($)
```

• Valid Input:
  • 2/3$ ✓
  • (1-2)*3$ ✓

• Invalid Input:
  22-3$ X
  2/$ X

```
Match()
    If token ==t
        advanceTokenPtr
    else
        error
```

• This code implements both left and right associative
operators. It uses different levels of non-terminals
to express operator precedence.

• User Information:
1) enter string using numbers ∅ to 3 and the
symbols +, -, *, /, (, and)
2) The end string variable will be the dollar sign ($
3) Program determines whether or not the input
string is a valid expression.