Nhat Doan

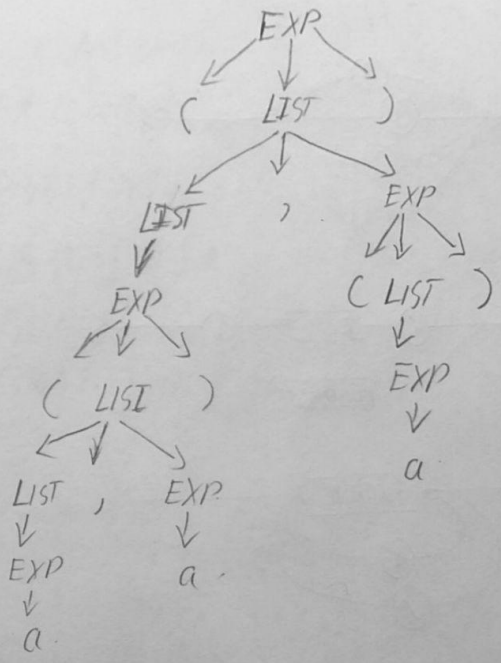
PL1

Part 2 :

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

LIST ::= LIST , EXP | EXP

a) Parse tree for ((a,a),a,(a))

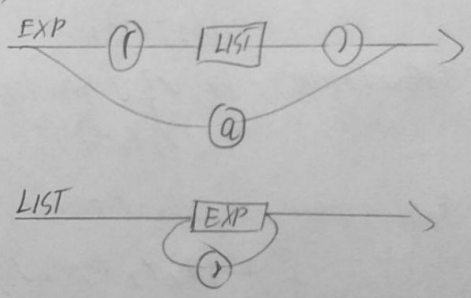


b) Translate to BNF

<EXP> ::= (LIST) | a

<LIST> ::= EXP {, EXP}

c)



d) First(EXP) = { (, a }.

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

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

(2) EXP ::= EXP + TERM | EXP - TERM | TERM

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

FACTOR ::= (EXP) | DIGIT

DIGIT ::= 0 | 1 | 2 | 3.

Nhat Doan

2) a) to EBNF

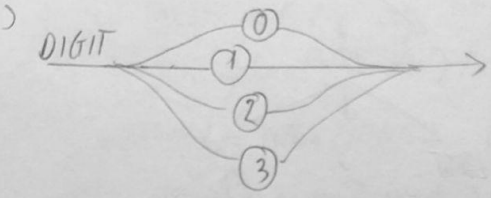
<EXP> ::= TERM {(+|-) TERM}

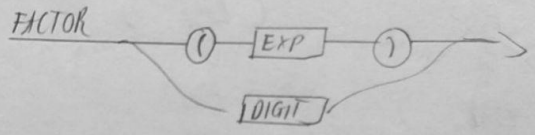
<TERM> ::= FACTOR {(*|/) FACTOR}

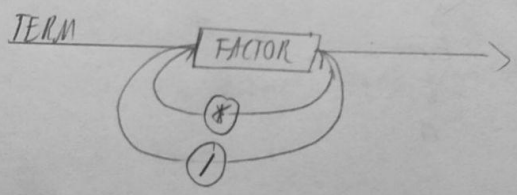
<FACTOR> ::= (EXP) | DIGIT

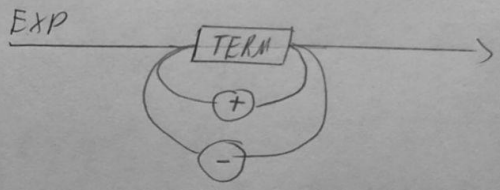
<DIGIT> ::= 0 | 1 | 2 | 3

b)









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

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

c) Two requirements on a grammar for a predictive parser to be able to make such choices:

- Firstsets of any two choices must not have any token in common.
- If A is optional, then First(A) ∩ Follow(A) should be empty.

What Doom

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

(1) First sets of any two choices must not have any token in common.

DIGIT is a trivial case

Factor : $\{(\} \cap First(DIGIT) = \{(\} \cap \{0,1,2,3\} = \phi$

TERM and EXP only have 1 first choice.

(2) If t is optional, then $First A \cap Follow(A)$ should be empty

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

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

$First(TERM) \cap Follow(TERM) = \{(,0,1,2,3\} \cap \{+, -,)\} = \phi$

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

(3) Recursive Descent recognizer Pseudocode.

```
Procedure Start:
    EXP
    if token => $error

Procedure EXP:
    TERM
    While (token is + or -)
        { match token
          TERM }

Procedure TERM
    FACTOR
    while (token is * or /)
        { match token
          FACTOR }
```

```
Procedure FACTOR:
    DIGIT
    if (token is "(")
        { match token
          DIGIT
          match ")" }
    else DIGIT
Procedure DIGIT
    if (token is 1 or 2 or 3 or 4.)
        match token.
    else $error.

Procedure match(input)
    if token match input
        advance next token.
    else $ error
```