프로그래밍언어

[4장] Problem Set

과목명	프로그래밍언어
교수명	김철홍
분 반	나반
학 과	컴퓨터학부
학 번	20211799
이 름	유진
제출일	2023.04.16

교재 *4장 Problem Set*의 다음 문제들을 푼 결과물을 *PDF 파일로 제출하세요. (문서편집기로 작업 또는 손으로 연습용지에(or 전자패드) 문제풀이 후 캡쳐 편집 가능)*

(1, 2, 3, 5, 7)

-

1. Perform the pairwise disjointness test for the following grammar rules.

- pairwise disjointness
       a. A -> aB ¦ b ¦ cBB
FIRST(aB) = {a} , FIRST(b) = {b}, FIRST(cBB) = {c}
Pass!

       b. B -> aB ¦ bA ¦ aBb
FIRST(aB) = {a} , FIRST(bA) = {b}, FIRST(aBb) = {a}
Fail!

       c. C -> aaA ¦ b ¦ caB
FIRST(aaA) = {a} , FIRST(b) = {b}, FIRST(caB) = {c}
Pass!

2. Perform the pairwise disjointness test for the following grammar rules.

- pairwise disjointness

       a. S -> aSb ¦ bAA
FIRST(aSb) = {a}, FIRST(bAA) = {b}
Pass!

b. A -> b{aB} ¦ a

FIRST(b{aB}) = {b}, FIRST(a) = {a}

Pass!

c. B -> aB ¦ a

FIRST(aB) = {a}, FIRST(a) = {a}

Fail!

3.   Show a trace of the recursive descent parser given in Section 4.4.1 for the string a + b * c.

4.4.1. Consider the following EBNF description of simple arithmetic expressions:

<expr> → <term> {(+ ¦ -) <term>}

<term> → <factor> {(* ¦ /) <factor>}

<factor> → id ¦ int_constant ¦ (<expr>)

- RD parser trace

The next token is: 10 The next lexeme is a

Enter <expr>

Enter <term>

Enter <factor>

The next token is: 21 The next lexeme is +

Exit <factor>

Exit <term>

The next token is: 10 The next lexeme is b

Enter <term>

Enter <factor>

```
The next token is: 10 The next lexeme is *
Exit <factor>


The next token is: 26 The next lexeme is c
Enter<factor>


The next token is: -1 The next lexeme is EOF
Exit <term>
Exit <expr>
```

5. Given the following grammar and the right sentential form, draw a parse tree and show the phrases and simple phrases, as well as the handle.
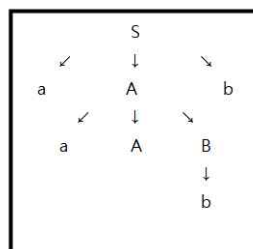
```
S -> aAb ¦   bBA
A -> ab ¦   aAB
B -> aB ¦ b
```

      a. aaAbb

Phrases: aaAbb,aAb,b

Simple Phrases: b

Handle : b
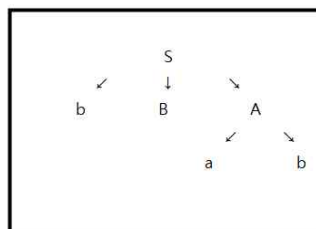

      b. bBab

Phrases: bBab,ab

Simple Phrases: ab

Handle : ab

c. aaAbBb218

Can't derive aaAbBb

7. Show a complete parse, including the parse stack contents, input string, and action for the string id * (id + id), using the grammar and parse table in Section 4.5.3.

4.5.3
1. E -> E+ T
2. E -> T
3. T -> T * F
4. T -> F
5. F ->(E)
6. F -> id

State	id	+	*	(	)	$	E	T	F
			Action					Goto	
0	S5			S4			1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

- LR 파서
id * (id + id)

stack	input	action
0	id*(id+id)$	shift 5
0id5	*(id+id)$	Reduce 6(use GOTO[0,F])
0F3	*(id+id)$	Reduce 4(use GOTO[0,T])
0T2	*(id+id)$	shift 7
0T2*7	(id+id)$	shift 4
0T2*7(4	id+id)$	shift 5
0T2*7(4id5	+id)$	Reduce 6(use GOTO[4,F])
0T2*7(4F3	+id)$	Reduce 4(use GOTO[4,T])
0T2*7(4T2	+id)$	Reduce 2(use GOTO[4,E])
0T2*7(4E8	+id)$	shift 6
0T2*7(4E8+6	id)$	shift 5
0T2*7(4E8+6id5	)$	Reduce 6(use GOTO[6,F])
0T2*7(4E8+6F3	)$	Reduce 4(use GOTO[6,T])

0T2*7(4E8+6T9	)$	Reduce 1(use GOTO[4,E])
0T2*7(4E8	)$	shift 11
0T2*7(4E8)11	$	Reduce 5(use GOTO[7,F])
0T2*7F10	$	Reduce 3(use GOTO[0,T])
0T2	$	Reduce 2(use GOTO[0,E])
0E1	$	accept