**Assignment 1**

**Course:** Topics in Compiler Design

**Course Code:** CS890DO

**Instructor:** Dr. Maher Elshakankiri

**Submitted By:** Betrand Nnamdi

**Submission Date:** 30th September 2018

**UNIVERSITY OF REGINA**

**Exercise 2 . 2 . 1 : Consider the context-free grammar**

**S -> SS+ | SS\* | a**

1. **Show how the string aa+a\* can be generated by this grammar.**

**Solution:**

S -> SS\*

-> SS+S\*

-> aS+S\*

-> aa+S\*

-> aa+a\*

1. **Construct a parse tree for this string.**

**Solution:**



1. **What language does this grammar generate? Justify your answer.**

**Solution:**

The language generated by this grammar is one that is made up of any possible arithmetic operations involving a series of a’s with the operators ‘+’ and ‘\*‘ in postfix notation.

**Exercise 2.2.2: What language is generated by the following grammars? In each case justify your answer.**

1. **S -> 0S1 | 01**

**Solution:**

The language generated by this grammar is one that is made up of a sequence of 0’s followed by a sequence of 1’s and the number of 0’s equal the number of 1’s.

1. **S -> +SS | -SS | a**

**Solution:**

The language generated by this grammar is one that is made up of any possible arithmetic operations involving ‘a’ using the ‘+’ and ‘-‘ operators in prefix notation.

1. **S -> S (S) S | ε**

**Solution:**

The language generated by this grammar is one that is made of up a series of matched pairs of parentheses of arbitrary arrangement and nesting.

1. **S -> a S b S | b S a S | ε**

**Solution:**

The language generated by this grammar accepts an empty string or all strings made up of the same number of a’s and b’s (i.e. equal number of a’s and b’s) in no definite order.

1. **S -> a | S + S | S S | S\* | (S)**

**Solution:**

The language generated by this grammar is one that is made up of any possible arithmetic operations involving ‘a’ using only the + and \* operations and a series of matched parentheses. Postfix and infix notations are used in the representation of this grammar.

**Exercise 2.2.3: Which of the grammars in Exercise 2.2.2 are ambiguous?**

The grammar **S -> S (S) S | ε** is ambiguous.

 

The grammar **S -> a S b S | b S a S | ε** is ambiguous. For example, the string abab can be parsed in more than one way:

 

The grammar **S -> a | S + S | S S | S\* | (S)** is ambiguous. For example, the string a+a\* can be parsed in more than one way:

 

**Exercise 2.2.4.: Construct unambiguous context-free grammars for each of the following languages. In each case show that your grammar is correct.**

1. **Arithmetic expressions in postfix notation.**

expr -> expr expr op | digit

digit -> [0 -9]

1. **Left-associative lists of identifiers separated by commas.**

list -> list, id | id

1. **Right-associative lists of identifiers separated by commas.**

list -> id, list | id

**d) Arithmetic expressions of integers and identifiers with the four binary** **operators +, -, \*, /**

expr -> expr + term | expr – term | term

term -> term \* factor | term / factor | factor

factor -> num | id | (expr)

1. **Add unary plus and minus to the arithmetic operators of (d).**

expr -> expr + term | expr – term | term

term -> term \* factor | term / factor | factor

factor -> num | id | (expr) | +factor | -factor

**Exercise 2.2.6.: Construct a context-free grammar for roman numerals**

**Solution:**

Roman numerals (as used today), are based on the following symbols:

**Symbol I V X L C D M**

**Value 1 5 10 50 100 500 1,000**

**1-9 = I, II, III, IIII, V, VI, VII, VIII, VIIII.**

**10-90 = X, XX, XXX, XL, L, LX, LXX, LXXX, XC**

**100-900 = C, CC, CCC, CD, D, DC, DCC, DCCC, CM**

**1000 – 3000 = M, MM, MMM**

**e.g. 3999 = MMMCMXCIX**

numeralList -> tenCubed tenSquared ten digit

tenCubed -> M | M M | M M M | ε

tenSquared - > factor | C D | D factor | C M

factor - > C | C C | C C C | ε

ten - > values | X L | L values| X C

values -> X | X X | X X X | ε

digit -> otherDigit | I V | V otherDigit | I X

otherDigit -> I | I I | I I I | ε

**Exercise 2.3.2.: Construct a syntax-directed translation scheme that translates arithmetic expressions from postfix notation into infix notation. Give annotated parse trees for the inputs 95-2\* and 952\*-.**

**Solution:**

Productions for postfix notation:

expr -> expr expr + | expr expr - | expr expr \* | expr expr / | num

expr -> expr {print("+")} expr +

| expr {print("-")} expr –

| {print("(")} expr {print(")\*(")} expr {print(")")} \*

| {print("(")} expr {print(")/(")} expr {print(")")} /

| num {print(num)}





**Exercise 2.4.1.: Construct recursive descent parsers starting with the following grammars**

1. **S -> + S S | - S S | a**

**Solution:**

void S()

{

switch(lookahead)

{

case “+”: match("+");

S();

S();

break;

case “-”: match("-");

S();

S();

break;

case “a”: match("a");

break;

default: report (“syntax error”);

}

}

void match(terminal t)

{

if (“lookahead == “t”)

{

lookahead = nextTerminal;

}

else

{

report (“syntax error”);

}

}

1. **S -> S ( S ) S | S |** **ε**

**Solution:**

void S()

{

if (lookahead == "(")

{

match("(");

S();

match(")");

S();

}

}

void match(terminal t)

{

if (“lookahead == “t”)

{

lookahead = nextTerminal;

}

else

{

report (“syntax error”);

}

}

1. **S -> 0 S 1 | 0 1**

**Solution:**

void S()

{

if (lookahead == "0")

{

match("0");

S();

match("1");

}

else

{

report (“syntax error”);

}

}

void match(terminal t)

{

if (“lookahead == “t”)

{

lookahead = nextTerminal;

}

else

{

report (“syntax error”);

}

}