Baheem Ferrell

Assignment 1
CSC 301

Overview

Here we look at grammars. We will look at working with predefined grammars such as determining whether or not a particular sentence belongs to the language of a grammar.

Problem 1

Working with grammars - assume that you have the following grammar:

<exp>* ::= <exp> + <exp>
| <exp> * <exp>
| (<exp>)
| a | b | c

Determine whether the following strings belong to the language of the above grammar. Construct an explicit parse tree where possible.

- a*b*c
- (a+b)(a+b)
- ((a + b))
- (a b)
- (c)+(b)
- b++

1.) a * b * c

<exp> → <exp> * <exp>

→ <exp> * <exp> * <exp>

→ a * <exp> * <exp>

→ a * b * <exp>

→ a * b * c

Belongs to the grammar

<u>Parse tree</u>

```
              <exp>
         /      |      \
      <exp>     *     <exp>
   /    |    \           |
<exp>   *  <exp>       <exp>
  |          |           |
  a          b           c
```

2.) (a+b) (a+b)

<exp> → <exp> + <exp>

→ (<exp>) + (<exp>)

→ (<exp> + <exp>) + (<exp> + <exp>)

→ (a+b) (a+b)

(a+b) (a+b) does not belong to the grammar

3.) ((a+b))

<exp> → (<exp>)

→ ((<exp>)) + (<exp>)

→ ((<exp> + <exp>))

→ ((a+b))

Parse tree

```
          <exp>
        /   |   \
       (  <exp>  )
        /   |    \
           <exp>  +  <exp>
             |         |
             a         b
```

4.) (ab) does not belong to the grammar because there is no operator

5.) (c) + (b)

<exp> → <exp> + <exp>

→ ((<exp>)) + (<exp>)

→ (c) +(b)

Parse tree

```
             <exp>
          /    |    \
     <exp>     +     <exp>
    /  |  \         /  |  \
   ( <exp> )       ( <exp> )
       |               |
```

c b

It belongs to the grammar.

6.) b++ does not belong to the grammar

Problem 2

- Construct a grammar for the set of all strings consisting of zero or more a's with a semicolon after each one. What exactly do the strings look like? What is the simplest string the grammar should generate? Try at least three derivations showing that your grammar derives strings in the above language.

1. a; a; a; a; a;
   → <exp>;
   → <exp>;<T>;
   → <exp>;<T>;<T>;
   → <exp>;<T>;<T>;<T>;
   → <exp>;<T>;<T>;<T>;<T>;
   → <exp>;<T>;<T>;<T>;<T>;<T>;

2. a; a; a;
   → <exp>
   → <exp>;<T>;
   → <exp>;<T>;<T>;
   → <T>;<T>;<T>;
   → a; a; a;

3. a; a;
   → <exp>
   → <exp>;<T>
   → <T>;<T>;
   → a; a;

Given the grammar,

G: <E>* ::= <E> + <T> | <T>
<T> ::= <T> * <P> | <P>
<P> ::= a | b | c

Add production → <T> ::= <T> + <T>|<T>*<T>
|<T>;<T>

String → a+b; a+c
→ <E>
→ <T>
→<T>+<T>
→<T>+<T>;<T>
→<T>+<T>;<T>+<T>
→ a+b : a+c

Add productions to the grammar that describe the binary ';' operator which has the lowest precedence of all the operators in the grammar and is left-associative. The idea behind this operator is that it allows you to list multiple expressions in one program like the following set of expressions
(A) a + b ; b *c
(B) a ; b ; c
(C) b ; b * b ; b * b * b ; b * b * b * b

- you get the idea. Show that your grammar can derive the three examples given above.

A.) a+b; b*c
→ <E>
→ <T>
→ <F>
→ <F>;<F>
→ <F>+<F>;<F>
→ <F>+<F>;<F>*<F>
→ <P>+<P>;<P>*<P>
-->a+b; b*c

B.) a; b; c

→ <E>
→ <T>
→ <F>
→ <F>;<F>
→ <F>;<F>;<F>
→ <P>;<P>;<P>
→ a; b; c

C.) b; b*b; b*b*b; b*b*b*b

→ <E>
→ <T>
→ <F>
→ <F>;<F>
→ <F>;<F>*<F>
→ <F>;<F>*<F>:<F>
→ <F>;<F>*<F>:<F>*<F>
→ <F>;<F>*<F>;<F>*<F>*<F>;
→ <F>;<F>*<F>;<F>*<F>*<F>;<F>
→ <F>;<F>*<F>;<F>*<F>*<F>;<F>*<F>
→ <F>;<F>*<F>;<F>*<F>*<F>;<F>*<F>*<F>
→ <F>;<F>*<F>;<F>*<F>*<F>;<F>*<F>*<F>*<F>
→ <P>;<P>*<P>;<P>*<P>*<P>;<P>*<P>*<P><P>
→ b; b*b; b*b*b; b*b*b*b