Assignment Two

Grammer and its Deriverations.

Quin'darius Lyles-Woods
qlyleswo@students.kennesaw.edu

Concepts of Programming Languages.
Professor Jose Garrido
Section W01

Bachelors of Computer Science
Kennesaw State University
1100 South Marietta Pkwy SE
Marietta, GA 30060
September 13, 2021

Part I

Problems

0.1 Explain whether the following grammar is ambiguous

$$
\begin{aligned}
< assignment > &\mapsto < identifier > = < expression > \\
< identifier > &\mapsto A \\
&\mapsto B \\
&\mapsto C \\
< expression > &\mapsto < expression > + < expression > \\
&\mapsto < expression > + < expression > \\
&\mapsto < expression > * < expression > \\
&\mapsto < expression > \\
&\mapsto < identifier >
\end{aligned}
$$

Explanation of the problem , we are trying to determine if the grammar provided will be classified as ambiguous.

Answer to the problem , the statement will be classified as ambiguous. This is because the grammar will generate and sentential form with two distinct parse trees.

Why this matters , the complier chooses what code to run based off the size of its parse tree. So when there is ambiguity then the complier has a hard time choosing which code to run because it cannot be determined uniquely.

0.2 Given the following grammar specification, draw the corresponding parse tree and write a leftmost derivation for each of the following statements.

$$
\begin{aligned}
< assignment > &\mapsto < identifier > = < expression > \\
< identifier > &\mapsto A \\
&\mapsto B \\
&\mapsto C \\
< expression > &\mapsto < identifier > + < expression > \\
&\mapsto < identifier > * < expression > \\
&\mapsto (< expression >) \\
&\mapsto < identifier >
\end{aligned}
$$

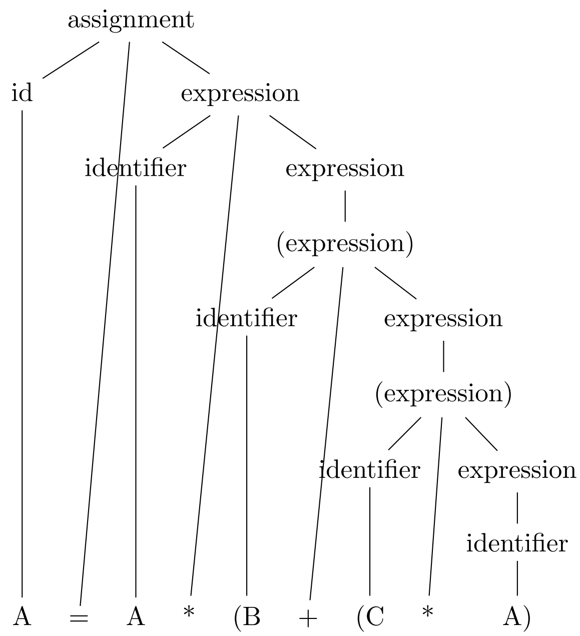
Explanation of the problem we are going to expanded out the statement in the context of the grammar to show all of its derivations in both the text format in the **leftmost** derivation, and the graphical format of the **parse tree**.

0.2.1 $A = A * (B + (C * A))$

Leftmost Derivation

$$
\begin{aligned}
< assignment > &\mapsto < identifier > = < expression > \\
&\mapsto A = < expression > \\
&\mapsto A = < identifier > * < expression > \\
&\mapsto A = A* < expression > \\
&\mapsto A = A * (< expression >) \\
&\mapsto A = A * (< identifier > + < expression >) \\
&\mapsto A = A * (B+ < expression >) \\
&\mapsto A = A * (B + (< expression >)) \\
&\mapsto A = A * (B + (< identifier > * < expression >)) \\
&\mapsto A = A * (B + (C* < expression >)) \\
&\mapsto A = A * (B + (C* < identifier >)) \\
&\mapsto A = A * (B + (C * A))
\end{aligned}
$$

Parse Tree

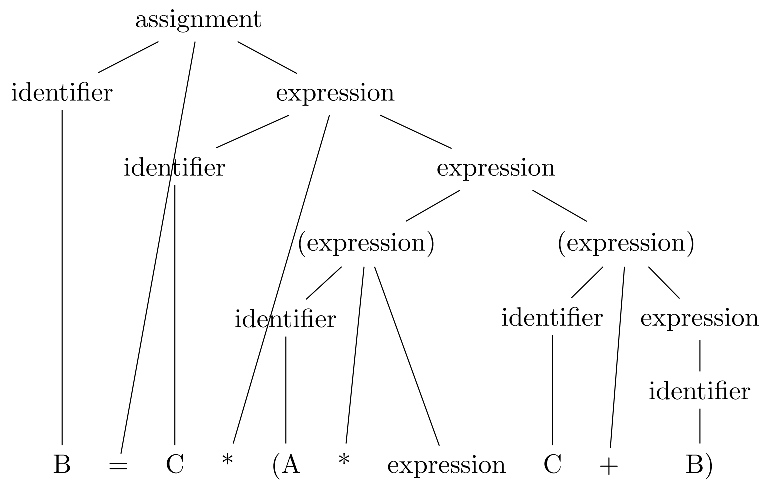


0.2.2 $B = C * (A * C + B)$

Leftmost Derivation

$$
\begin{aligned}
< assignment > &\mapsto < identifier > = < expression > \\
&\mapsto B = < expression > \\
&\mapsto B = < identifier > * < expression > \\
&\mapsto B = C* < expression > \\
&\mapsto B = C * (< expression >) \\
&\mapsto B = C * (< identifier > * < expression >) \\
&\mapsto B = C * (A* < expression >) \\
&\mapsto B = C * (A* < identifier > + < expression >)) \\
&\mapsto B = C * (A * C+ < expression >)) \\
&\mapsto B = C * (A * C+ < identifier >)) \\
&\mapsto B = C * (A * C + B))
\end{aligned}
$$

Parse Tree


assignment
identifier
expression
identifier
expression
(expression)
(expression)
identifier
identifier
expression
identifier
B = C * (A * expression C + B)

0.3 Convert the following to Backus-Naur Form

$$< Statement > \mapsto < A > \{b < A >\}$$
$$< A > \mapsto a[b] < A >$$

Answer

$$\begin{aligned} < Statement > &:= A \\ &|\{b < A >\} \\ A &:= a[b] < A > \end{aligned}$$

Explanation of the problem we are going to but the given expression in Backus-Naur Form.

Backus-Naur Form Consist of:

- terminals symbols
- non-terminals symbols

Why this matters Backus-Naur Form is the way we specify programming language currently. Invented by John Backus and Peter Naur and many others to help describe programming languages.

0.4 A language consists of strings that have n copies of the letter a followed by the same number of copies of the letter b, where $n > 0$. Define the grammar specification for the language.

Answer

$$
\begin{aligned}
< statement > &\mapsto < letter > b \\
&\mapsto < letter > b, < statement > \\
< letter > &\mapsto A \\
&\mapsto B \\
&\mapsto C
\end{aligned}
$$

Explanation of the problem We are given the task of making a recursive language that can take an arbitrary amount of arguments and also add a slight modification to such language. We must remember that whenever the syntax is on the **LHS** and the **RHS** of the equation the grammar is recursive.

Why this matters Because almost all languages are infinite so we must have a solid understanding on how to recursive recognize the language.

0.5 Using the following grammar, Find out which of the following sentences are legal in the language generated by this grammar.

$$
\begin{aligned}
< Statement > &\mapsto < A > a < B > b \\
< A > &\mapsto < A > b \\
&\mapsto b \\
< B > &\mapsto a < B > \\
&\mapsto a
\end{aligned}
$$

0.5.1 baab

Leftmost Derivation

$$
\begin{aligned}
< Statement > &\mapsto < A > a < B > b \\
&\mapsto ba < B > b \\
&\mapsto baab
\end{aligned}
$$

0.5.2 bbbab

Leftmost Derivation

$$
\begin{aligned}
< Statement > &\mapsto < A > a < B > b \\
&\mapsto b < A > a < B > b \\
&\mapsto bb < A > a < B > b \\
&\mapsto bbba < B > b \\
&\mapsto bbbaab
\end{aligned}
$$

0.5.3 bbaaaaaS

Leftmost Derivation

$$\begin{aligned} < Statement > &\mapsto < A > a < B > b \\ &\mapsto b < A > a < B > b \\ &\mapsto bb < A > a < B > b \\ &\mapsto bbba < B > b \\ &\mapsto bbbaa < B > b \\ &\mapsto bbbaaa < B > b \\ &\mapsto bbbaaaa < B > b \\ bbbaaaaa &\neq bbaaaaaS \end{aligned}$$

0.5.4 bbaab

Leftmost Derivation

$$\begin{aligned} < Statement > &\mapsto < A > a < B > b \\ &\mapsto b < A > a < B > b \\ &\mapsto bba < B > b \\ &\mapsto bbaab \end{aligned}$$

Explanation of the problem We are tasked with the function for being a language recognizer. Giving input statements we are asked whether they are valid statements.

Why this matters We need to understand what the computer is doing when it is running our code so the recognition of syntax is an essential things to know in the understanding computers.

Part II

Summary and Conclusions