

Grammars and Semantics

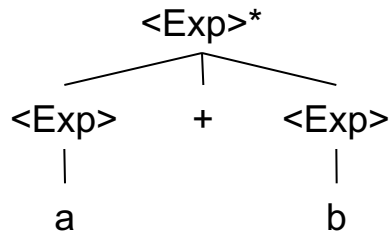
- Programming languages are used to specify computations – that is, computations are the meaning/semantics of programs.

Grammars and Semantics

Consider the simple language of expressions:

$$\begin{array}{lcl} G: <Exp>^* & ::= & <Exp> + <Exp> \\ & & | <Exp> * <Exp> \\ & & | a \\ & & | b \\ & & | c \end{array}$$

When we write the sentence $a + b$ we can build the parse tree:

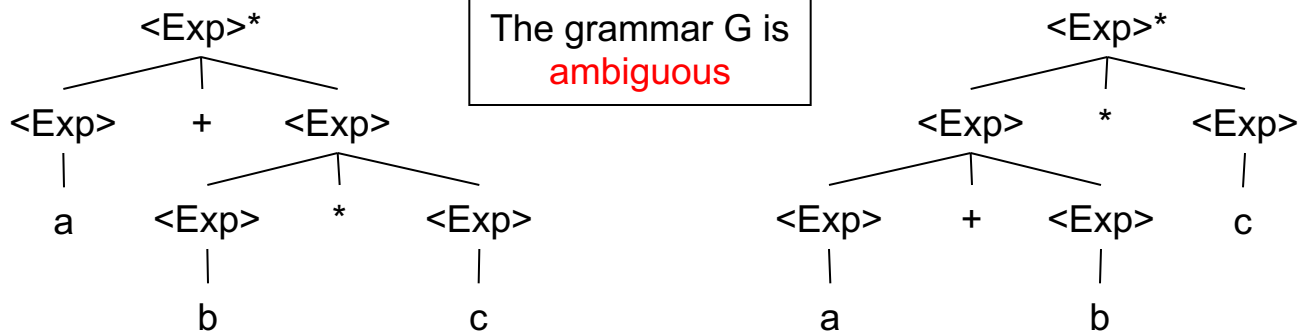


We can say that this parse tree *represents* the computation $a + b$.

If we let a and b be variables, then the parse tree gives us a procedure to compute $a + b$ by starting at the leaves of the tree: (1) lookup the values of the variables (2) pass the values up along the parse tree branches (3) use the values to compute the value of the $+$ operator.

Grammars and Semantics

Now consider the sentence $a + b * c$, for this sentence we can construct two parse trees:



Even though both parse trees derive the same terminal string, the computations they represent are very different:

- (1) left tree – first compute the product, then the addition
- (2) right tree – first compute the addition, then the product

Since we had written the original sentence without parentheses the left parse tree represents the intended computation according to algebraic conventions.

However, from a machine point of view, there is no way of knowing which parse tree to pick...

Grammars and Semantics

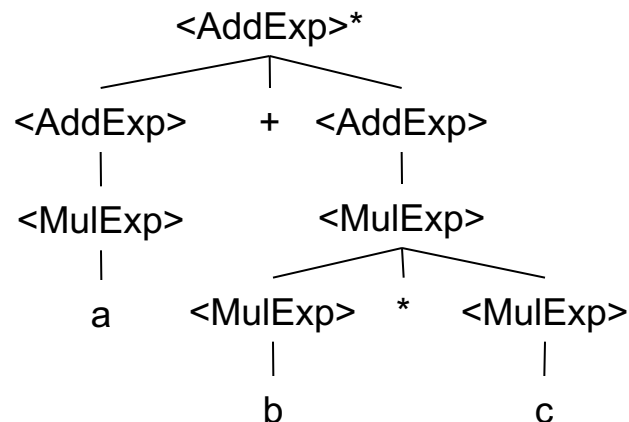
...we need additional information: operator precedence

Operator precedence means that some operators bind tighter than others, e.g. * binds tighter than +.

We can build operator precedence right into our grammar:

$$\begin{aligned} G' : & \langle \text{AddExp} \rangle^* ::= \langle \text{AddExp} \rangle + \langle \text{AddExp} \rangle \\ & \quad \quad \quad | \quad \langle \text{MulExp} \rangle \\ & \langle \text{MulExp} \rangle ::= \langle \text{MulExp} \rangle * \langle \text{MulExp} \rangle \\ & \quad \quad \quad | a \quad | b \quad | c \end{aligned}$$

Let's try our problematic sentence $a + b * c$, only one parse tree is possible:



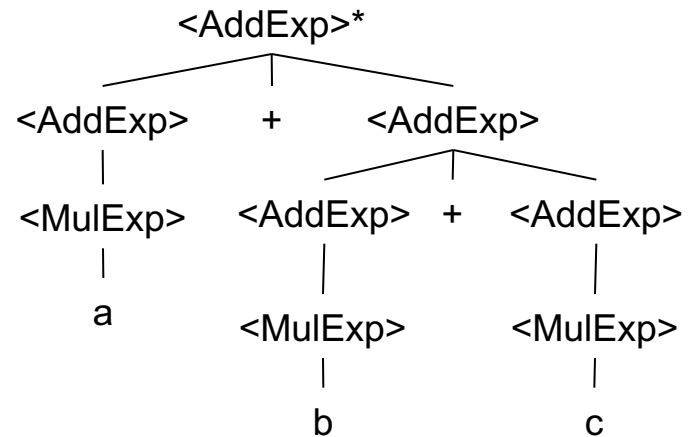
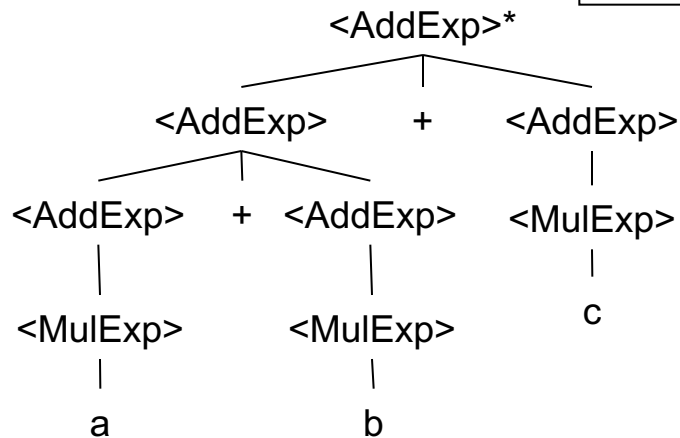
This is the only parse tree we can build, therefore, the grammar G' is not ambiguous.

Grammars and Semantics

However, our new grammar still has a problem, consider the sentence $a+b+c$; here we have two possible parse trees:

```
G' : <AddExp> ::= <AddExp> + <AddExp>
      | <MulExp>
    <MulExp> ::= <MulExpr> * <MulExp>
      | a | b | c
```

The grammar G' is
ambiguous



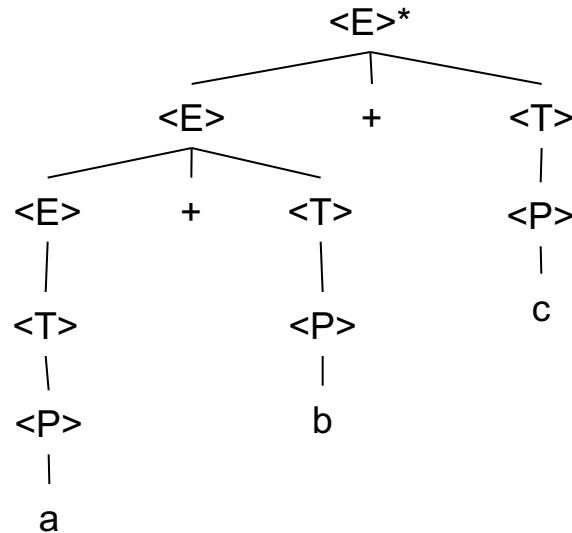
Grammars and Semantics

- Again, our grammar is ambiguous because the computation specified by the sentence $a+b+c$ can be represented by two different parse trees.
- We need more information!
- There is one more algebraic property we have not yet explored – associativity
- Most algebraic operators, including the $+$ operator, are left-associative.
- We can rewrite our grammar to take advantage of this additional information:

$$\begin{aligned} G'' : \quad & \langle E \rangle^* ::= \langle E \rangle + \langle T \rangle \mid \langle T \rangle \\ & \langle T \rangle ::= \langle T \rangle * \langle P \rangle \mid \langle P \rangle \\ & \langle P \rangle ::= a \mid b \mid c \end{aligned}$$

Grammars and Semantics

Let's try our sentence $a+b+c$ again with grammar G'' :

$$\begin{aligned} G'' : \quad & \langle E \rangle^* ::= \langle E \rangle + \langle T \rangle \mid \langle T \rangle \\ & \langle T \rangle ::= \langle T \rangle * \langle P \rangle \mid \langle P \rangle \\ & \langle P \rangle ::= a \mid b \mid c \end{aligned}$$


There is no other way to derive this string from the grammar and thus the grammar is not ambiguous.

Take Away

- Grammars can be ambiguous in the sense that a derived string can have multiple distinct parse trees.
- By taking additional information such as associativity and precedence about the operators of a language into account we can construct grammars that are not ambiguous.

Grammars and Semantics

Given the following grammar,

$$\begin{aligned} G'' : \quad & \langle E \rangle^* ::= \langle E \rangle + \langle T \rangle \mid \langle T \rangle \\ & \langle T \rangle ::= \langle T \rangle * \langle P \rangle \mid \langle P \rangle \\ & \langle P \rangle ::= a \mid b \mid c \end{aligned}$$

Add productions to the grammar that define the right-associative operator = at a lower precedence than any of the other operators.

This new operator should allow you to write expressions such as

$a = b$

$a = b = c$

$a = b = b + c$

Grammars and Semantics

a) Show that the following grammar is ambiguous.

$$\begin{array}{lcl} G: <S> & ::= & <S> <S> \\ & | & (<S>) \\ & | & () \end{array}$$

b) Rewrite the above grammar so that it is no longer ambiguous.

Class Exercise

- Let $L(G)$ be the set of all strings that start with an a followed by zero or more b 's and end with a c . Design grammar G .
- Show that the following grammar is ambiguous:

$$\begin{array}{ccccc} \langle A \rangle^* & ::= & \langle A \rangle & @ & \langle A \rangle \\ & & | & & * \end{array}$$

Assignment

- Assignment #5 – see BS