COMS 381, Summer 2005
Supplementary Handout 4
Ambiguity in Grammars
Friday, June 17th

Ambiguous grammars are undesirable in real-life compilers for a variety of good reasons. While we cannot write a program to determine in general whether a given grammar G is ambiguous (this is an example of an *undecidable problem*, similar to the halting problem), there is much we can do to remove ambiguity from a grammar once we have located it.

Here are the examples we saw in class.

Example 1 - Arithmetic expressions and operator precedence

Consider the following grammar G_1 for arithmetic expressions with addition and multiplication:

$E \rightarrow E + E \mid E * E \mid (E) \mid \text{digit}$

This is an ambiguous grammar, as for example the string $1 + 2 * 3$ has two different parse trees.

We can, however, convert it to an equivalent grammar G_2 which is unambiguous, as follows:

We introduce two new nonterminals, F (for 'factor') and T (for 'term'). Our grammar is then:

$E \rightarrow E + T \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow (E) \mid \text{digit}$

You can now check that $1 + 2 * 3$ has only one parse tree in G_2, and moreover that this parse tree correctly captures our intuition about the relative precedence of multiplication and addition.

Example 2 - The dangling else problem

This problem will arise when parsing a language with the following type of grammar for if-statements:

$S \rightarrow \text{if } (E)\ S$
$S \rightarrow \text{if } (E)\ S \text{ else } S$
$S \rightarrow \text{other}$

We can write the following type of program (here in pseudo-Java):

```
if (x < 0) if (y < 0) a = true; else a = false;
```

The problem is that it is unclear which if the else belongs to. Conceivably, it could belong to either, as the program stands. (Of course, forcing the programmer to use braces in the appropriate place would have solved the problem!).

Our original grammar needs to be fixed, as it is ambiguous. Our fix will be to restrict the nesting of if-statements. Specifically, *unmatched* if-statements (that is, those without an else) may not be nested in the if-clause of another if-statement. They can still be nested inside the else-clause if desired. A new grammar could look like this:

$S \rightarrow$ matched | unmatched
matched $\rightarrow$ if (E) matched else matched | other
unmatched $\rightarrow$ if (E) S | if (E) matched else unmatched