Syntax of Programming Languages (cont’d)


Syntactic Ambiguity

In English

Syntactically ambiguous sentences of English:

- “I saw the dog with the binoculars.”
- “The friends you praise sometimes deserve it.”
- “He seemed nice to her.”

Other kinds of ambiguity in English:

Aside: We can often “disambiguate” ambiguous sentences. Question: How?

But we can be wrong.

Example: “I put the box on the table .”

In a programming language

Example grammar excerpt (based on Sebesta, p. 119):

```
<stmt> → <assert-stmt> | <loop-stmt> | <if-stmt>
<if-stmt> → if <boolean-exp> then <stmt>
           | if <boolean-exp> then <stmt> else <stmt>
```

Example sentence:

```
if (x odd) then
if (x = 1) then
print "bleep";
else
print "boop";
```

Definition: A grammar is ambiguous iff it generates a sentence for which there are two or more distinct parse trees.

To prove that a grammar is ambiguous, give a string and two parse trees for it.

A sentence is ambiguous with respect to a grammar iff that grammar generates two or more distinct parse trees for the sentence.

Note that having two distinct derivations does not make a sentence ambiguous. A derivation corresponds to a traversal through a parse tree, and one can traverse a single tree in many orders.

Exercise: Draw the two parse trees.
Example

Grammar: if statement two slides ago.

Sentence:
if (x odd) then
  print "bleep";

One parse tree:

Two derivations:

Changing the language to include delimiters

Algol 68 if-statement grammar:

\[
\begin{align*}
<\text{stmt}> & \rightarrow <\text{assn-stmt}> \mid <\text{loop-stmt}> \mid <\text{if-stmt}> \\
<\text{if-stmt}> & \rightarrow \text{if} <\text{boolean-exp}> \text{then} <\text{stmt}> \text{fi} \\
& \quad | \text{if} <\text{boolean-exp}> \text{then} <\text{stmt}> \text{else} <\text{stmt}> \\
& \quad | \text{fi}
\end{align*}
\]

Notation and Terminology

We say that \( L(G) \) is the language generated by grammar \( G \).

So \( G \) is ambiguous if \( L(G) \) contains a sentence which has more than one parse tree, or more than one leftmost (or canonical) derivation.

Dealing with ambiguity

We have two strategies:

1. Change the language to include delimiters
2. Change the grammar to impose associativity and precedence

Example: A CFG for Arithmetic Expressions

Grammar 1:

\[
\begin{align*}
<\text{exp}> & \rightarrow <\text{exp}> + <\text{exp}> \\
& \mid <\text{exp}> - <\text{exp}> \\
& \mid <\text{exp}> \times <\text{exp}> \\
& \mid <\text{exp}> / <\text{exp}> \\
& \mid <\text{id}> \\
& \mid <\text{lit}>
\end{align*}
\]

Example: parse \( 8 - 3 \times 2 \)
Changing the language to include delimiters

Grammar 2:

\[
<\text{expn}> \rightarrow ( <\text{expn}> ) - ( <\text{expn}> ) \mid 
( <\text{expn}> ) * ( <\text{expn}> ) \mid 
<\text{identifier}> \mid 
<\text{literal}> 
\]

\((8-((3)*(2))) \in L(G)\)
\(((8)-(3))*2 \in L(G)\)
\(8 - 3 * 2 \notin L(G)\)

Grammar 3:

\[
<\text{expn}> \rightarrow <\text{expn}> - <\text{expn}> \mid 
<\text{expn}> * <\text{expn}> \mid 
<\text{identifier}> \mid 
<\text{literal}> \mid 
( <\text{expn}> ) 
\]

Accepts all expressions, but still ambiguous!

Grouping in parse tree now reflects precedence

Example: parse \(8 - 3 \times 2\)

Precedence

- Low Precedence: Addition + and Subtraction -

- Medium Precedence: Multiplication * and Division /

- Higher Precedence: Exponentiation ^

- Highest Precedence: Parenthesized expressions ( <expr> )

\(\Rightarrow\) Ordered lowest to highest in grammar.
We still have ambiguity...

Example: $3 - 2 - 1$ is still a problem.

- The grouping of operators of same precedence is not disambiguated.

- For non-commutative operators, only one parse tree is correct.

- Operators may be left or right associative.

Changing the grammar to impose associativity

Grammar 5:

<exp> -->

Associativity

- Deals with operators of same precedence

- Implicit grouping or parenthesizing

- Left associative: $\ast$, $/$, $+$, $-$

- Right associative: $-$

Dealing with Ambiguity

1. Can't always remove an ambiguity from a grammar by restructuring productions.

2. When specifying a programming language, we want the grammar to be completely unambiguous.

3. An inherently ambiguous language does not possess an unambiguous grammar.

4. There is no algorithm that can examine an arbitrary context-free grammar and tell if it is ambiguous, i.e., detecting ambiguity in context-free grammars is an undecidable problem.
An Inherently Ambiguous Language

Suppose we want to generate the following language:

\[ \mathcal{L} = \{a^ib^j c^k \mid i, j, k \geq 1, i = j \text{ or } j = k \} \]

Grammar:

Limitations of CFGs

CFGs are not powerful enough to describe some languages.

Example:

- The language consisting of strings with one or more a’s followed by the same number of b’s then the same number of c’s. 
  \( \{ a^i b^i c^i \mid i \geq 1 \} \).
- \( \{ a^m b^n c^m d^n \mid m, n \geq 1 \} \).

The Chomsky Hierarchy

There are several categories of grammar that are more and less expressive, forming a hierarchy:

- Phrase-structure grammars
- Context-sensitive grammars
- Context-free grammars
- Regular grammars

Research question: Exactly what things can and cannot be expressed with a CFG?
Using CFGs for PL Syntax

Some aspects of programming language syntax can’t be specified with CFGs:

- Cannot declare the same identifier twice in the same block.
- Must declare an identifier before using it.
- A[i,j] is valid only if A is two-dimensional.
- The number of actual parameters must equal the number of formal parameters.

Other things are awkward to say with CFGs:

- Identifier names must be no more than 50 characters long.

These aspects of a programming language are usually specified informally, separately from the formal grammar.

Applications of Formal Grammars

Specifying a programming language

REs for lexical syntax; BNF for syntax; plus extra rules for things that BNF cannot say.

Identifying strings for an operating system command

Examples (Unix commands that use extended REs):

- ls s[y-z]*
- grep Se.h syntax.tex
- Scripting languages like awk use regular expressions.
  - awk '/to[kg]e/ {print $1}' syntax.tex

Voice recognition

Problem: Given recorded speech, produce a string containing the words that were spoken.

Difficulties:

How can a grammar help?

Example: ViaVoice, by IBM. See chapter 6 of the IBM Speech Programmer’s Guide:

http://wwwhome.cs.utwente.nl/~liraraki/grammar.htm

Compiler-compilers

Examples:

- bison (the GNU replacement for yacc)
- JavaCC.
  - See: http://www.webgain.com/products/java_cc

So why does anyone still write compilers by hand?