More CFGs, Parse Trees, Ambiguity

CS152

Chris Pollett

Sep. 17, 2008.

Outline

- More about CFGs
- Parse Trees and Abstract Syntax Trees
- Ambiguity

More about CFGs

- Recall we are learning about how to specify the syntax of a programming language.
- On Monday, we were learning about context-free grammars and gave an example grammar for a very tiny fragment of English:

1. *sentence* --> *noun-phrase verb-phrase* .
2. *noun-phrase* --> *article noun.*
3. *article* --> a | the.
4. *noun* --> girl | dog.
5. *verb-phrase* --> *verb noun-phrase.*
6. *verb* --> sees | pets.

Remarks

- Using this grammar we gave a derivation of the sentence: the girl sees a dog.
- If you look at that grammar, you'll see that you could derive sentences like: "the dog pets the girl."
- So although it is syntactically correct, it doesn't quite make sense. Syntax ≠ Semantics.
- In terms of programming, you can have syntactically correct programs which don't do anything useful.

What is a context free grammar?

- A context free grammar consists of a sequence of rules (called **productions**) of the form: some structure-name, followed by --> or ::=, followed by a string consisting of token symbols and 0 or more additional structure names.
- Structure names are sometimes called **nonterminals** and token symbols are sometimes called **terminals**.
- A context free grammar also has a distinguished nonterminal called the **start symbol**.
- The start symbol for our English example was *sentence*.
- All derivations must begin from the start symbol.
- The **language of a grammar** consists of all strings s of only terminals which have derivations beginning from the start symbol in grammar and which terminate with s.

Why are CFGs called *context-free*?

- You could imagine productions where you have more than one thing on the left hand side:

  <sentence> ::= <start-of-sentence> <noun-phrase> <verb-phrase>

  <start-of-sentence> <article> ::= The | A

  <start-of-sentence> ::= empty-string

  …

- This would allow you to capitalize the start of sentence, but the rule:

  <start-of-sentence> <article> ::= The | A

  uses the context in which the <article> appears.

Some More Example Grammars

- Here is a grammar for arithmetic expressions involving + and *:

  <expr> ::= <expr> + <expr> | <expr> * <expr> | (<expr>) | <number>

  <number> ::= <number> <digit> | <digit>

  <digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

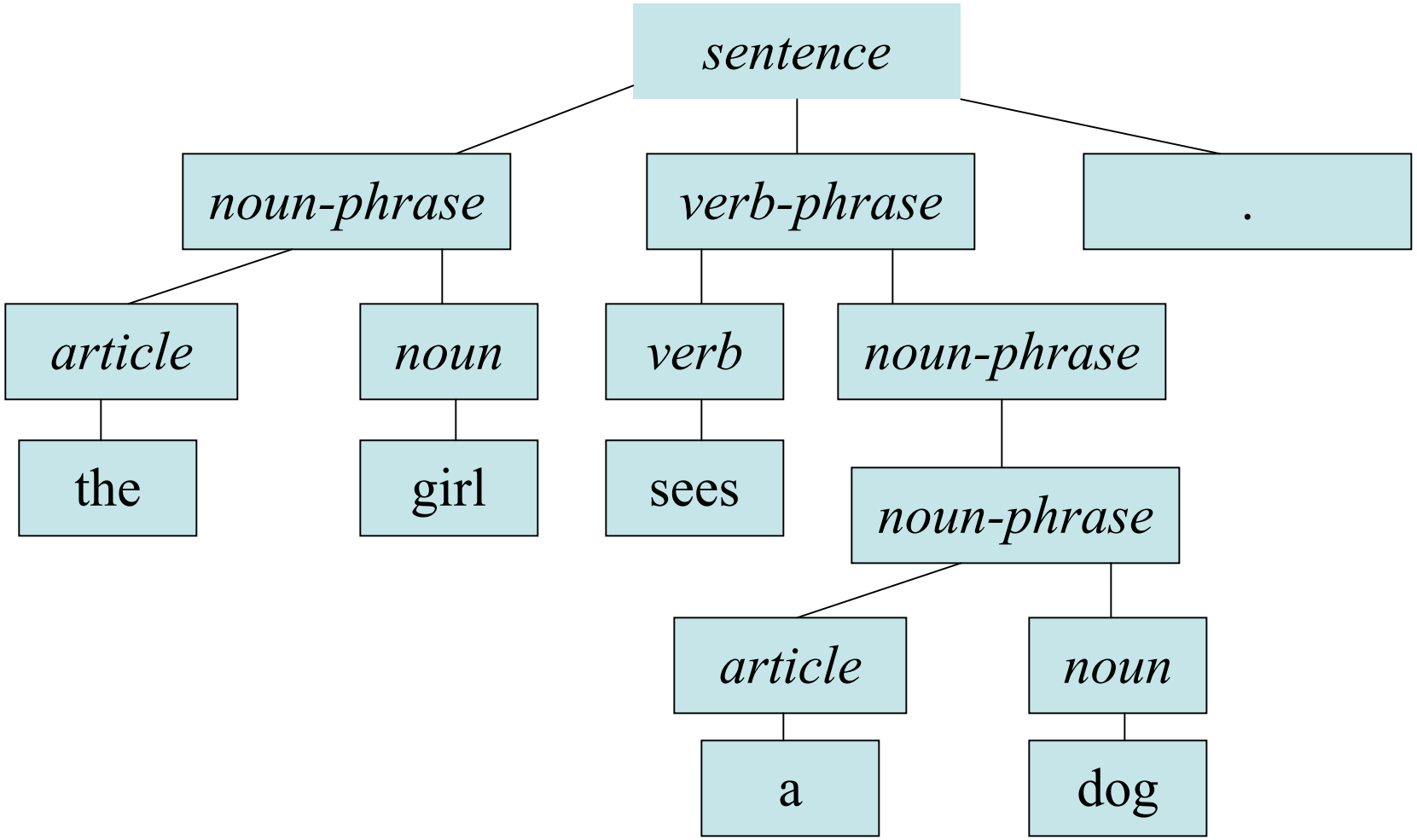
- Note the recursive nature of the rules above: <expr> and <number> appear on both sides of their rules. This is completely legal.
- Give the derivation for (2 + (11 * 5)).
- The book gives a fragment of the C's grammar.
- It has productions like:

  <external-declaration> ::= <function-definition> | <declaration>

Parse Trees and Abstract Syntax Trees

- Syntax establishes structure not meaning.
- One way of associating meaning with a particular program is to use the structure one obtains from parsing it and annotate that structure with code to give it meaning.
- This is sometimes called **syntax-directed semantics**.
- The structure typically associated with parsing a program is called its **parse tree**.

Example Parse Tree



Remarks

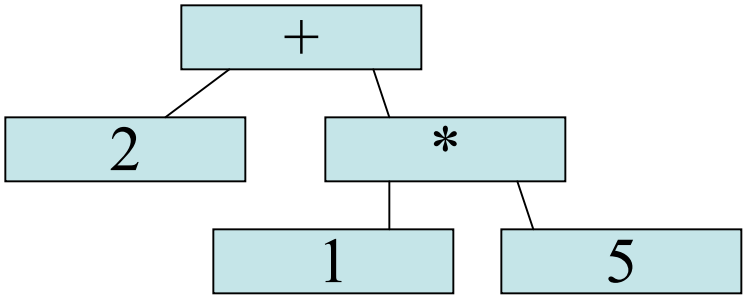
- Notice how the parse tree is completely specified by the grammar rules that were used in the derivation.
  - i.e., if a rule like

    <struct1> ::= <struct2> <struct3> terminal

    Was used then in the parse tree <struct2>, <struct3>, and terminal will be children of <struct1>
- Try to come up with the parse tree for (2 + (11*5)) in the grammar we gave a couple slides back.
- All the terminals and nonterminals in a derivation are included in a parse tree.
- Not all of these may be necessary to determine completely the syntactic structure of an expression.

More remarks

- For example, the relevant part of the tree needed to provide semantics to (2 + (1 * 5)) might look like:



- Such abbreviated trees are called **abstract syntax trees** or just **syntax trees**.

Ambiguity.

- Consider the expression 3 + 4 * 5.
- It actually has two distinct parse trees using the grammar of a couple slides back:
  - One corresponds to 3 + (4 *5).
  - The other to (3 + 4) *5.
- Worse, these two expressions evaluate to different things.
- Grammars which have two distinct parse trees for the same string are called **ambiguous**.
- We'll discuss how to avoid making such grammars on Monday.