Precedence, EBNFs and Syntax Diagrams

CS152

Chris Pollett

Sep. 22, 2008.

Outline

- Disambiguiting rules, Precedence, Associativity
- EBNFs and Syntax Diagrams

Recalling Ambiguity

- Recall last Wednesday we had the grammar:
  <expr> ::= <expr> + <expr> | <expr> * <expr> | (<expr>)
  | <number>
  <number> ::= <number> <digit> | <digit>
  <digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
- 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**.

Leftmost Derivations

- If a derivation in each step always operates on its leftmost non-terminal, then it is called a **leftmost derivation**.
- It turns out that having distinct parse trees for the same string is equivalent to having two distinct leftmost derivations for the same string.
- In the example above, one derivation begins

  <expr> => <expr> * <expr> => <expr> + <expr> *<expr>

  the other as

<expr> => <expr> + <expr> => <number> + <expr>=> <digit> + <expr> => 3 + <expr> => 3+ <expr> * <expr>

and the rest of the derivations are the same.

PDAs

- There are algorithms (such as CYK) which work for parsing any CFG ambiguous or not.
- They are typically slow -- $O(n^3)$ -- and they don't address the problem of the fact that ambiguous grammars often yield strings with two "meanings".
- To do parsing people instead, prefer to use a machine model like the finite automata model we briefly discussed for regular expressions.
- For CFGs, this model is basically a finite automata together with a stack, a push down automata. (PDA).
- When trying to parse a grammar, the approach is to initially **shift** the start symbol for the grammar unto the stack.
- Then in each step we check is the top symbol of the stack a non-terminal? If it is, we pop it and replace it with a right hand side of a rule with involving that non-terminal.
- If there a terminal on the top of the stack we check if the input has that terminal. If it does we read the terminal/token from the input and pop the terminal from the stack.
- We keep going till the string is parsed.

Disambiguating Rules

- The problem with ambiguous grammars is that there may be more than one rule that could be pushed onto the stack in a given step.
- One way to solve this problem (and this can be done in YACC) is to give a **precedence** to the rules.
- I.e., we could say do rule <expr> ::= <expr> + <expr> before <expr> ::= <expr> * <expr>.
- This yields the parenthesization 3 + (4 * 5).
- Alternatively, we could modify our grammar to remove the problem:

  <expr> ::= <expr> + <expr> | <term>

  <term> ::= <term> * <expr> | (<expr>) | <number>
- This has the same effect as giving precedence to the rules.

Associativity

- Consider 3 + 4 + 5. This could be viewed as either (3 + 4) +5 or 3 + (4 +5).
- The first would say + is **left associative**, the second **right associative**.
- Our current grammar, using leftmost derivations, favors a left associative parse trees for +.
- For +, it doesn't really matter; however, for -, notice (3 - 4) - 5 ≠ 3 - (4 - 5).
- We can modify our grammar to make + either left or right associative, by replacing <expr> ::= <expr> + <expr> with either <expr> ::= <term> + <expr> or <expr> ::= <expr> + <term>

EBNFs

- EBNF stands for extended BNF.
- It allows us slightly more general rules to make it easier to write down grammars.
- For example, rather than have to write

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

  to say that a <number> a string of one or more <digits>, one can write instead

  <number> ::= digit {digit}

  here {} is used to denote zero of more repetitions.
- Another abbreviation is [ ] for optional. So one can write

  if (<expr>) <statement> [else <statement>]

  to indicate the else clause is optional.

Syntax Diagrams

- Sometimes a diagramming notation called **syntax diagrams** is used to indicate grammar rules. For instance, Oracle documentation often uses this.
- In syntax diagrams a circle is used for a terminal and a box for a non-terminal.
- The left hand side of the rule is indicated by a word above an arc coming into the diagram. Arcs are used to indicate connections between parts of the rule.
- So <noun-phrase> ::= <article> <noun> and <article> ::= a | the might be draws as:

Example Diagram