Lecture Notes 5

Parsing Problem

- Syntax Analysis (or Parsing) – Process of analyzing syntax
- Top-Down Parsing – Syntax Tree is built from the root down
- Bottom-Up Parsing – Syntax Tree is built from the leaves up
- Parsing Notation
  - Terminal Symbols – (a, b, c, …) lower beginning of alphabet
  - Non-Terminal Symbols – (A, B, C, …) upper beginning of alphabet
  - Terminals or Non-Terminals – (W, X, Y, Z) upper end of alphabet
  - Strings of Terminals – (w, x, y, z) lower end of alphabet
  - Mixed Strings (Terminals and/or Non-Terminals) – (α, β, δ, γ) lower Greek letters
- Left Sentential Form ($xA\alpha$) – A is the left most non-terminal to be parsed, x is string of parsed terminals with string mix of terminals following.
  - Parsing Problem Example:
    - Assume $xA\alpha$ current state
    - Rules: $A \rightarrow bB$, $A \rightarrow cBb$, and $A \rightarrow a$
    - Next Step: $xbB\alpha$, $xcBb\alpha$, or $xa\alpha$
    - Which way to parse? There are three possibilities for the rule.
    - Top-Down usually solve by looking at next token to see if first of RHS rule.
- Recursive Decent Parser – Top-Down Parser that is based on the BNF description
- Parsing Table – A lookup table to decide the next rule instead of coding in recursive function calls
- LL Algorithms – Left-to-right scan with Leftmost derivation
- Handle – Correct RHS to match for reducing in a Bottom-Up Parser.
- LR Algorithms – Left-to-right scan with Rightmost derivation
- General Parsing Complexity – General algorithm for parsing unambiguous grammar $O(n^3)$ for string length of n, but in practice for programming languages is $O(n)$
- Direct Left Recursion – Occurs with rules where the RHS has the LHS at the beginning
  - Example: $A \rightarrow A + B$
- Direct Left Recursion Removal Algorithm: (ϵ – f Erasure Rule)

  For each nonterminal, A,

  1. Group the A-rules as $A \rightarrow A\alpha_1 \mid \ldots \mid A\alpha_m \mid \beta_1 \mid \ldots \mid \beta_n$ where none of the β's begins with A
  2. Replace the original A-rules with
     $A \rightarrow \beta_1 A' \mid \ldots \mid \beta_n A'$
     $A' \rightarrow \alpha_1 A' \mid \ldots \mid \alpha_m A' \mid \epsilon$

- Indirect Left Recursion – Occurs when a cycle of rules can cause infinite recursion
  - Example:
    $A \rightarrow BaA$
    $B \rightarrow A\ b$
- Pairwise Disjointness Test – Test to determine if Top-Down parsing can choose correct RHS by looking at next token
- Left Factoring – Separating the rule into two rules, the second may have an ϵ so that the rules pass the Pairwise Disjointness Test
- Handle Formal Definition – Where $=>_{rm}$ is a right most derivation step and $=>*_{rm}$ is zero or more steps
  β is the handle of the right sentential form $\gamma = \alpha\beta w$ iff $S => *_{rm}\ \alpha A w =>_{rm} \alpha\beta w$
- Phrase Definition – Where =>+ is one or more derivation steps
  β is a phrase of the right sentential form γ iff $S =>* \gamma = \alpha_1 A \alpha_2 =>+ \alpha_1 \beta \alpha_2$
- Simple Phrase Definition – **NOTE BOOK HAS TYPO**
  β is a phrase of the right sentential form γ iff $S =>* \gamma = \alpha_1 A \alpha_2 => \alpha_1 \beta \alpha_2$
- Selecting Correct Handle - The handle of rightmost sentential form is the leftmost simple phrase
- Shift-Reduce Algorithm – Algorithm used for bottom-up parsing
  - Shift Action – Moves the next input token onto stack
  - Reduce Action – Replaces a RHS (handle) on top of stack with LHS
- LR Parser Advantages
  - Can be used for all programming languages
  - Can detect syntax errors as soon as possible in left-to-right scan
  - LR class of grammars is a proper superset of the class of parsable by LL parsers
- LR Parser Disadvantages
  - Parsing table is difficult to construct by hand
  - Detected syntax errors lack full context of where they occur (sometimes difficult to generate meaningful error messages)