Recursive Descent Parsers

Lecture 6

Robb T. Koether

Hampden-Sydney College

Mon, Feb 2, 2009
Outline

1. Recursive Descent Parser
2. Example
Recursive Descent Parser

- Each nonterminal in the grammar is implemented as a function.
- Begin with the start symbol $S$ of the grammar by calling the function $S()$.
- Based on the first token received, apply the appropriate grammar rule for $S$.
- Continue in this manner until $S$ is “satisfied.” That is, all nonterminals have been replaced with terminals.
Recursive Descent Parsers

- The first Pascal compiler used a recursive descent parser.
- Recursive descent parsers have the benefit of being very simple to implement.
- However,
  - Error-recovery is difficult.
  - They are not able to handle as large a set of grammars as other parsing methods.
Error Recovery

- When a syntax error occurs, in order for the compiler to recover, it usually has to discard the last few tokens, move to the end of the line, and resume.
- In a recursive descent parser, discarding tokens involves returning from several nested function calls.
- In a table-driven parser, discarding tokens requires simply clearing part of the stack.
Example (Recursive Descent)

- Write a parser for the following grammar.

\[
S \rightarrow \text{if } C \text{ then } S \;; \\
\quad \mid \text{while } C \text{ do } S \;; \\
\quad \mid \text{id }=\text{ num} \\
\quad \mid \text{id }++
\]

\[
C \rightarrow \text{id }==\text{ num} \mid \text{id }!=\text{ num}
\]

where \(S\) represents a statement and \(C\) represents a condition.
Example (Recursive Descent)

- Run the example `SimpleParser.java`. 
Example (Recursive Descent)

Modify the previous example by adding the production

\[ S' \rightarrow SS' | \varepsilon \]

where \( S' \) represents a sequence of statements.
Example (Recursive Descent)

Modify the previous example by adding the productions

\[
S \rightarrow \text{do } S \text{ while } C \ ; \\
C \rightarrow \text{id } < \text{num}
\]