Compiler Construction (CS - 636)

Sadaf Manzoor

Outline

1. LL(1) Parsing
2. Error-Recovery in Top-Down Parsers
3. Summary

Top-Down Parsing

Lecture: 13 & 14

LL(1) Parsing

- LL(1) is top-down parsing algorithm
  - First ‘L’ means it processes input from left to right
  - Second ‘L’ means it traces out a leftmost derivation for the input string
  - The ‘(1)’ means it uses only one symbol of input to predict the direction of the parse
- LL(1) parsing uses an explicit stack rather than recursive calls to perform a parse
- LL(1) performs two actions
  1. Replace a nonterminal A from top of stack using grammar
  2. Match a token on top of stack with the next input token

LL(1) Parsing (Continue…)

- LL(1) parser uses following table information during parsing
- Grammar: S → (S) S | ε
- Input: ()

No	Parsing Stack	Input	Action
1	$ S	( ) $	S → (S) S
2	$ S) S (	( ) $	Match
3	$ S) S	) $	S → ε
4	$ S)	) $	Match
5	$ S	$	S → ε
6	$	$	Accept

Removing Left Recursion

- LL(1) suffers the same problem due to left recursion as RD does
- EBNF is not a solution for LL(1) hence we need to rewrite grammar and remove left recursion from it
- Consider the following case

  $A \rightarrow A\alpha \mid \beta$

  - Here α and β are combination of terminals and nonterminals where β does not begin with A
  - This type of grammar will generate string of type β[αα….]
  - The resultant will be: $A \rightarrow \beta A'$

    $A' \rightarrow \alpha A' \mid \varepsilon$

Left Factoring

- Left factoring is required when two or more grammar rule choices share a common prefix string, as in the rule

$$A \rightarrow \alpha\beta \mid \alpha\gamma$$

- Obviously an LL(1) parser cannot distinguish between the production choices in such a situation
- The solution in this case is to ‘factor’ the α out in the left and rewrite the rule as two rules;

$$A \rightarrow \alpha A'$$

$$A' \rightarrow \beta \mid \gamma$$

Error Recovery in Top-Down Parsers

- A parser should try to determine that an error has occurred *as soon as possible*. Waiting too long before declaring error means the location of the actual error may have been lost

- After an error has occurred, the parser must pick a likely place to resume the parse. A parser should always try to parse as much of the code as possible, in order to find as many real errors as possible during a single translation

Error Recovery in Top-Down Parsers (Continue…)

- A parser should try to avoid the error cascade problem, in which one error generates a lengthy sequence of spurious error messages

- A parser must avoid infinite loops on errors, in which an unending cascade of error messages is generated without consuming any input

Summary

Any Questions?