CSE302: Compiler Design

Instructor: Dr. Liang Cheng
Department of Computer Science and Engineering
P.C. Rossin College of Engineering & Applied Science
Lehigh University

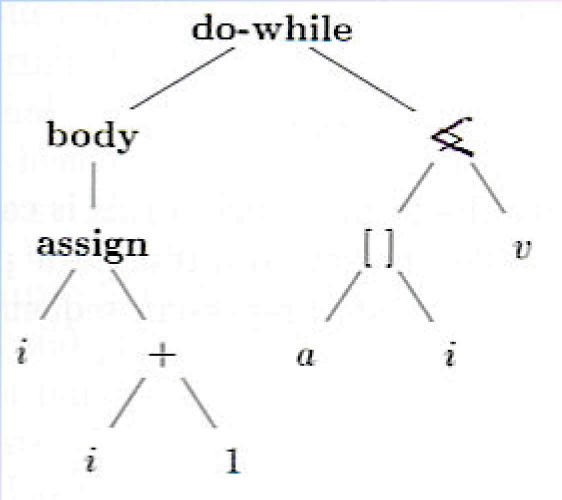
February 20, 2007

Outline

- Recap
  - Implementing lexical-analyzer generators (Sections 3.6, 3.7, and 3.8)
- Syntax analysis (Chapter 4)
- Summary and homework

Three Types Of Parsers

- Universal
  - Cocke-Younger-Kasami algorithm and Earley's algorithm
- Top-down
  - From the root to leaves
- Bottom-up
  - From leaves to the root



Input And Output Of Parsers

- A stream of tokens coming from lexer
- Generate some representation of the parse tree
  - Collecting information about tokens into the symbol table
  - Type checking and static semantic analysis
  - Error handling

Error Handling: Error Types

- Common types of errors
  - Lexical errors
    - Misspelling, missing quotes around string texts
  - Syntactic errors
    - Misplaced semicolons
    - Extra or missing braces
    - Missing matching keywords
  - Static semantic errors
    - Type mismatches
    - Return values for void return method
  - Logical errors
    - = vs. ==

Error Handling: Error Recovery

- Print the offending line with a pointer to the error position
- Panic-mode recovery
  - Discard input symbols one at a time until one of a designated set of synchronizing tokens is found
    - Delimiters
- Phrase-level recovery
  - Replace a prefix of the remaining input by some string that allows the parser to continue
    - , $\rightarrow$; delete an extraneous ; insert a missing ;
- Global correction
  - A minimal sequence of changes to obtain a globally least-cost correction
- Error productions
  - Add error productions in the grammar

Context-free Grammar

- *stmt* $\rightarrow$ **if** (*expr*) *stmt* **else** *stmt*
- Terminals
  - Token names
- Nonterminals
- A start symbol
- Productions
  - Head or left side
  - $\rightarrow$ or ::=
  - Body or right side

Notations for Context-free Grammar

- *stmt* → **if** (*expr*) *stmt* **else** *stmt*
- Terminals
  - Lowercase letters early in the alphabet (*a*,*b*,*c*)
  - Operator symbols
  - Punctuation symbols
  - The digits 0,1,...,9
  - Boldface strings
- Nonterminals
  - Uppercase letters early in the alphabet (*A*,*B*,*C*,*D*,*E*,*F*) & *T*
    - *E*: expressions; *T*: terms; *F*: factors
  - Letter *S* or the head of the 1st production: start symbol
  - Lowercase, italic names

More Notations for Context-free Grammar

- Uppercase letter late in the alphabet (X, Y, Z) represent grammar symbols
  - Either nonterminals or terminals
- Lowercase Greek letters ($\alpha, \beta, \gamma, \ldots$) represent strings of grammar symbols
  - $A \rightarrow \alpha$
- Lowercase letter late in the alphabet (u, v, w, x, y, z) represent strings of terminals
- A set of productions $A \rightarrow \alpha_1$, $A \rightarrow \alpha_2$, ... , $A \rightarrow \alpha_k$, with a common head A, may be written as
  - $A \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_k$

Derivations

- Leftmost
  - The top-down construction of the parse trees
- Rightmost
  - The bottom-up construction of the parse trees
- The symbol $\overset{*}{\Rightarrow}$ means “derives in zero or more steps”
  -
  - $program \overset{*}{\Rightarrow} a = b + \textbf{const}$
- The symbol $\overset{+}{\Rightarrow}$ means “derives in one or more steps”

Sentential Form and A Language

- $S \overset{*}{\Rightarrow} \alpha$ and S is the start symbol of a grammar G
  -
  - α is a sentential form of G
  - A sentence of G is a sentential form without nonterminals
- The language $L(G)$ generated by G is its set of sentences
- $S \underset{lm}{\overset{*}{\Rightarrow}} \alpha$ then α is a *left-sentential* form of a grammar

More Terminologies

- If S $\stackrel{*}{\Rightarrow}$ means "derives in zero or more steps"
  -
  - *program* $\stackrel{*}{\Rightarrow}$ *a* = *b* + **const**

- The symbol $\stackrel{+}{\Rightarrow}$ means "derives in one or more steps"

Derivations and Parse Trees

- The leaves of a parse tree are labeled by nonterminals and terminals, which constitute a sentential form
  - The yield or frontier of the parse tree
- $\alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow \alpha_n$ where α_1 is A
  - For each sentential form α_i, we can construct a parse tree whose yield is α_i

Ambiguity

- A grammar that produces more than one parse tree for some sentence is ambiguous

Verifying Language Generated

- A proof that a grammar G generates language L has two parts
  - Every string generated by G is in L
  - Every string in L can be generated by G
    - $S \rightarrow (S) S \mid \varepsilon$

BNF vs. Regular Expressions

- Every construct that can be described by a regular expression can be described by a BNF grammar
  - Convert a NFA to BNF
    - For each state *i* of NFA, create a nonterminal A_i
    - If state *i* has a transition to state *j* on *a*

      $A_i \to aA_j$; if *a* is ε, add $A_i \to A_j$
    - If *i* is an accepting state

      $A_i \to \varepsilon$
    - If *i* is the start state, make A_i the start symbol

BNF vs. Regular Expressions

- A regular expression may not be able to define a language that can be defined by a BNF.
  - L = $\{a^n b^n \mid n \geq 1\}$

Outline

- Recap
- Syntax analysis (Chapter 4)
- Summary and homework