CSE302: Compiler Design

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

February 27, 2007

Outline

- Recap
  - Writing a grammar (Section 4.3)
- Top-down parsing (Section 4.4)
- Summary and homework

Writing A Grammar

- Eliminating ambiguity
- Elimination of left recursion
  - For top-down parsing
- Left factoring
  - For top-down parsing

Outline

- Recap
- Top-down parsing (Section 4.4)
- Summary and homework

Top-Down Parsing

- At each step the key problem is determining the production to be applied for a nonterminal, say *A*
  - Recursive-descent parsing
    - May require backtracking to find the correct A-production
  - Predictive parsing
    - No backtracking is required
      - Look ahead at the input a fixed number (*k*) of symbols
      - LL(*k*) class grammars

Recursive-Descent Parsing

- void A() {

  Choose an A-production, $A \rightarrow X_1 X_2 \ldots X_n$

  for (i=1 to n) {

  if (X_i is a nonterminal) call X_i();

  else if (X_i equals the current input a)

  advance the input to the next symbol;

  else /* an error occurred, backtrack */

  }

  }

Predictive Parsers

- Recursive-descent parsers with one input symbol lookahead that requires no backtracking
  - No backtracking: being deterministic in choosing a production
  - Can be constructed for a class of grammars called LL(1)
  - 1st L: scanning the input from left to right
  - 2nd L: producing a leftmost derivation

FIRST Function and Set

- During top-down parsing, FIRST and FOLLOW allow us to choose which production to apply
  - FIRST(α) is the set of terminals that begin strings derived from α
  - If $\alpha \overset{*}{\Rightarrow} \varepsilon$, then ε is also in FIRST(α)
- Compute the FIRST set of a symbol X
  - If X is a terminal, then FIRST(X)={X}
  - If X is a nonterminal and $X \rightarrow Y_1 Y_2 \ldots Y_k$
    - If $X \rightarrow \varepsilon$ is a production, then add ε to FIRST(X)
    - Place a in FIRST(X) if for some i, a is in FIRST(Y_i) and ε is in all of FIRST(Y_1), ... , FIRST(Y_{i-1})
    - If ε is in all of FIRST(Y_1), ... , FIRST(Y_k), then add ε to FIRST(X)

Compute FIRST(X)

- $X \rightarrow Y_1 Y_2 \ldots Y_k$
  - Everything in FIRST(Y_1) is in FIRST(X)
  - If Y_1 does not derive ε, then stop
  - If Y_1 does derive ε, then add FIRST(Y_2) to FIRST(X)
  - If Y_2 does not derive ε, then stop
  - If Y_2 does derive ε, then add FIRST(Y_3) to FIRST(X)
  - ...
- Examples

Compute FIRST(α)

- α is a string of symbols $X_1X_2...X_n$
  - All non-ε symbols in FIRST(X_1) are in FIRST(α)
  - If ε is not in FIRST(X_1), then stop
  - If ε is in FIRST(X_1), then add FIRST(X_2) to FIRST(α)
  - If ε is not in FIRST(X_2), then stop
  - If ε is in FIRST(X_2), then add FIRST(X_3) to FIRST(α)
  - ...
  - If ε is in all FIRST(X_i), then add ε in FIRST(α)

- Examples

Usefulness of FIRST Sets

- In top-down parsing
  - At each step the key problem is determining the production to be applied for a nonterminal, say A
- $S \underset{lm}{\overset{*}{\Rightarrow}} \gamma A \lambda$
- $A \rightarrow \alpha$ and $A \rightarrow \beta$
  - FIRST(α) and FIRST(β) are disjoint sets
  - If a is in FIRST(α) then choose $A \rightarrow \alpha$
  - If a is in FIRST(β) then choose $A \rightarrow \beta$
  - How about a is neither in FIRST(α) nor in FIRST(β) ?

FOLLOW Function and Set

- FOLLOW(A) for nonterminal A is the set of terminals a that can appear immediately to the right of A in some sentential form
  - The set of terminals a such that there exists a derivation of the form

    $$S \overset{*}{\Rightarrow} \alpha A a \beta$$

  - If A can be the rightmost symbol in some sentential form, then \$ is in FOLLOW(A)
    - \$ is the input right endmarker and it is NOT a symbol of any grammar

Compute FOLLOW Sets For ALL Nonterminals *A*

- Place \$ in FOLLOW(*S*), where *S* is the start symbol, and \$ is the input right endmarker
  - \$ is not a symbol of any grammar
- If there is a production $A \rightarrow \alpha B \beta$, then everything in FIRST(β) except ε is in FOLLOW(*B*)
- If there is a production $A \rightarrow \alpha B$, or a production $A \rightarrow \alpha B \beta$, where FIRST($\beta$) contains ε (i.e. $\beta \Rightarrow^* \varepsilon$), then everything in FOLLOW(*A*) is is in FOLLOW(*B*)
  - Whatever followed A must follow B, since we can see from the production rule that nothing may follow B

Examples

- $E \rightarrow T\,E'$
- $E' \rightarrow +\ T\,E' \mid \varepsilon$
- $T \rightarrow F\,T'$
- $T' \rightarrow *\ F\,T' \mid \varepsilon$
- $F \rightarrow (\ E\ ) \mid \mathbf{id}$

Predictive Parsers

- Recursive-descent parsers with one input symbol lookahead that requires no backtracking
  - No backtracking: being deterministic in choosing a production
  - Can be constructed for a class of grammars called LL(1)
  - 1st L: scanning the input from left to right
  - 2nd L: producing a leftmost derivation

LL(1) Grammars

- Whenever $A \rightarrow \alpha$ and $A \rightarrow \beta$ are two distinct A-productions of G, the following conditions hold
  - For no terminal a do both α and β derive strings beginning with a
  - At most one of α and β can derive the empty string
  - If $\beta \overset{*}{\Rightarrow} \varepsilon$, then α does not derive any string beginning with a terminal in FOLLOW(A)
  - If $\alpha \overset{*}{\Rightarrow} \varepsilon$, then β does not derive any string beginning with a terminal in FOLLOW(A)

Why Such Conditions?

- In top-down parsing
  - At each step the key problem is determining the production to be applied for a nonterminal, say A
- $S \underset{lm}{\overset{*}{\Rightarrow}} \gamma A \lambda$
- $A \rightarrow \alpha$ and $A \rightarrow \beta$
  - FIRST(α) and FIRST(β) should be disjoint sets
  - If ε is in First(α), then FOLLOW(A) should be different from FIRST(β)

Predictive Parsing For LL(1) Grammar

- The production $A \rightarrow \alpha$ is chosen if
  - The next input symbol *a* is in FIRST(α)
  - The next input symbol *a* (or $) is in FOLLOW(A) and ε is in FIRST(α)
    - The next symbol could be $
- Thus we should construct a parsing table M where M[*A*,*a*] = $A \rightarrow \alpha$
  - In function *A* if the input is *a*, then call functions and/or match terminals of α

Constructing A Predictive Parsing Table M For ANY Grammar G

- For each production $A \rightarrow \alpha$
  - For each terminal a in FIRST(A), add $A \rightarrow \alpha$ to M[A,a]
  - If ε is in FIRST(α), then for each terminal b in FOLLOW(A), add $A \rightarrow \alpha$ to M[A,b]
  - If ε is in FIRST(α) and if \$ is in FOLLOW(A), add $A \rightarrow \alpha$ to M[A,\$]
- If, after performing the above, there is no production at all in M[A,a], then set M[A,a] to **error**

Non-recursive Predictive Parsing

- A stack storing symbols
- A input pointer *ip*
- A parsing table M for grammar G

- Set *ip* to point to the 1st symbol of input
- Set *X* to the top stack symbol
- **while**($X \neq \$$) {

```
        if (X is a) pop the stack and advance ip
        else if (X is a terminal) error();
        else if (M[X,a] is an error entry) error();
        else if (M[X,a] = X → Y1 Y2 ... Yk) {
                output the production or other actions;
                pop the stack;
                push Yk, ... , Y2 , Y1 onto the stack with Y1 on top;
        }
        Set X to the top stack symbol;
}
```

Examples

Outline

- Recap
  - Syntax analysis basics (Sections 4.1 & 4.2)
- Top-down parsing (Section 4.4)
- Summary and homework