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 α ⇒ ε, 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 → Y₁ Y₂...Yₖ
      - If X → ε is a production, then add ε to FIRST(X)
      - Place a in FIRST(X) if for some i, a is in FIRST(Yᵢ) and ε is in all of FIRST(Y₁), ..., FIRST(Yᵢ⁻¹)
      - If ε is in all of FIRST(Y₁), ..., FIRST(Yₖ), then add ε to FIRST(X)
Compute $\text{FIRST}(X)$

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

- Examples
Compute FIRST($\alpha$)

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

- 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 $A$
  
  - $S \Rightarrow \gamma A \lambda$
    - $I_m$
  
  - $A \rightarrow \alpha$ and $A \rightarrow \beta$
    - FIRST($\alpha$) and FIRST($\beta$) are disjoint sets
    - If $a$ is in FIRST($\alpha$) then choose $A \rightarrow \alpha$
    - If $a$ is in FIRST($\beta$) then choose $A \rightarrow \beta$
    - How about $a$ is neither in FIRST($\alpha$) nor in FIRST($\beta$)?
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
    \[ S \Rightarrow \alpha A a \beta \]
  - If \(A\) can be the rightmost symbol in some sentential form, then $ is in \text{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(\beta) except \epsilon 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 \epsilon (i.e. $\beta \Rightarrow \epsilon$), then everything in FOLLOW(A) is in FOLLOW(B)
  - Whatever followed A must follow B, since we can see from the production rule that nothing may follow B
Examples

- $E \rightarrow TE$
- $E \rightarrow + TE \mid \epsilon$
- $T \rightarrow FT$
- $T' \rightarrow * FT' \mid \epsilon$
- $F \rightarrow (E) \mid \text{id}$
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 $\alpha$ and $\beta$ derive strings beginning with $a$.
  - At most one of $\alpha$ and $\beta$ can derive the empty string *
  - If $\beta \Rightarrow \varepsilon$, then $\alpha$ does not derive any string beginning with a terminal in FOLLOW($A$) *
  - If $\alpha \Rightarrow \varepsilon$, then $\beta$ 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 $A$
  
  \[ S \Rightarrow^* \gamma A \lambda \]
  
  \[ l_m \]

- $A \rightarrow \alpha$ and $A \rightarrow \beta$
  
  - FIRST($\alpha$) and FIRST($\beta$) should be disjoint sets
  - If $\epsilon$ is in First($\alpha$), then FOLLOW($A$) should be different from FIRST($\beta$)
The production $A \rightarrow \alpha$ is chosen if:

- The next input symbol $a$ is in $\text{FIRST}(\alpha)$
- The next input symbol $a$ (or $\$$) is in $\text{FOLLOW}(A)$ and $\varepsilon$ is in $\text{FIRST}(\alpha)$
  - 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 $\alpha$
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 $\epsilon$ is in FIRST($\alpha$), then for each terminal $b$ in FOLLOW($A$), add $A \rightarrow \alpha$ to $M[A, b]$
  - If $\epsilon$ is in FIRST($\alpha$) 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 \( \text{ip} \)
- A parsing table \( M \) for grammar \( G \)

- Set \( \text{ip} \) to point to the 1st symbol of input
- Set \( X \) to the top stack symbol

\[
\text{while}(X \neq \$$) \{
  \text{if} \ (X \text{ is } a) \ \text{pop the stack and advance } \text{ip}
  \text{else if} \ (X \text{ is a terminal}) \ \text{error()};
  \text{else if} \ (M[X,a] \text{ is an error entry}) \ \text{error()};
  \text{else if} \ (M[X,a] = X \rightarrow Y_1 Y_2 \ldots Y_k) \{
    \text{output the production or other actions;}
    \text{pop the stack;}
    \text{push } Y_k, \ldots, Y_2, Y_1 \text{ onto the stack with } Y_1 \text{ on top;}
  \}
  \text{Set } X \text{ to the top stack symbol;}
\}
\]
Examples
Outline

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