CS 4300: Compiler Theory

Chapter 4
Syntax Analysis

Dr. Xuejun Liang

Outlines (Sections)

1. Introduction
2. Context-Free Grammars
3. Writing a Grammar
4. Top-Down Parsing
5. Bottom-Up Parsing
6. Introduction to LR Parsing: Simple LR
7. More Powerful LR Parsers
8. Using Ambiguous Grammars
9. Parser Generators

Quick Review of Last Lecture

- Introduction
    - The role of the Parser
    - Many levels of Programming Errors
    - Error Recovery Strategies
    - Representative Grammars
- Context-Free Grammars
    - Derivations and Languages
- Writing a Grammar
    - Lexical Versus Syntactic Analysis
    - Eliminating Ambiguity
    - Eliminating left recursion

Left Recursion

- A grammar is **left recursive** if it has a nonterminal *A* such that there is a derivation $A \stackrel{+}{\Longrightarrow} A\ \alpha$ for some string α.
- When a grammar is left recursive then a predictive parser loops forever on certain inputs.
- **Immediate left recursion**, where there is a production of the form $A \rightarrow A\ \alpha$.

$$
\begin{aligned}
A &\rightarrow A\ \alpha \\
&\mid \beta \\
&\mid \gamma
\end{aligned}
\qquad\qquad
\begin{aligned}
A &\rightarrow \beta\ R \\
&\mid \gamma\ R \\
R &\rightarrow \alpha\ R \\
&\mid \varepsilon
\end{aligned}
$$

Algorithm to eliminate left recursion

Input: Grammar G with no cycles or ε-productions

Arrange the nonterminals in some order $A_1, A_2, \ldots, A_n$

for $i = 1, \ldots, n$ {

    for $j = 1, \ldots, i\text{-}1$ {

        replace each

$$A_i \to A_j\, \gamma$$

        with

$$A_i \to \delta_1\, \gamma \mid \delta_2\, \gamma \mid \ldots \mid \delta_k\, \gamma$$

        where

$$A_j \to \delta_1 \mid \delta_2 \mid \ldots \mid \delta_k$$

    }

    eliminate the *immediate left recursion* in A_i

}

Immediate Left-Recursion Elimination

Rewrite every left-recursive production

$$A \to A\alpha_1 \mid A\alpha_2 \mid \cdots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \cdots \mid \beta_n$$

into a right-recursive production:

$$A \to \beta_1 A' \mid \beta_2 A' \mid \cdots \mid \beta_n A'$$
$$A' \to \alpha_1 A' \mid \alpha_2 A' \mid \cdots \mid \alpha_m A' \mid \epsilon$$

$$A \to A\,\alpha \mid A\,\delta \mid \beta \mid \gamma$$

⇒

$$A \to \beta\,A' \mid \gamma\,A'$$
$$A' \to \alpha\,A' \mid \delta\,A' \mid \varepsilon$$

Example Left Recursion Elim.

$$S \to A\, a \mid b$$
$$A \to A\, c \mid S\, d \mid \epsilon$$

Choose arrangement: *S*, *A*

i = 1: Nothing to do

i = 2, *j* = 1: *Replace S in A → S d with A a | b*

$$A \to A\, c \mid A\, a\, d \mid b\, d \mid \epsilon$$

Eliminate the *immediate left recursion* in *A*

$$S \to A\, a \mid b$$
$$A \to b\, d\, A' \mid A'$$
$$A' \to c\, A' \mid a\, d\, A' \mid \epsilon$$

Example Left Recursion Elim.

$$A \rightarrow B\ C \mid \mathbf{a}$$
$$B \rightarrow C\ A \mid A\ \mathbf{b}$$
$$C \rightarrow A\ B \mid C\ C \mid \mathbf{a}$$

Choose arrangement: A, B, C

$i = 1$: nothing to do

$i = 2, j = 1$: $B \rightarrow C\ A \mid \underline{A}\ \mathbf{b}$

$\Rightarrow \quad B \rightarrow C\ A \mid \underline{B\ C}\ \mathbf{b} \mid \underline{\mathbf{a}}\ \mathbf{b}$

$\Rightarrow_{\text{(imm)}} \quad B \rightarrow C\ A\ B_R \mid \mathbf{a}\ \mathbf{b}\ B_R$

$\quad B_R \rightarrow C\ \mathbf{b}\ B_R \mid \varepsilon$

$i = 3, j = 1$: $C \rightarrow \underline{A}\ B \mid C\ C \mid \mathbf{a}$

$\Rightarrow \quad C \rightarrow \underline{B\ C}\ B \mid \underline{\mathbf{a}}\ B \mid C\ C \mid \mathbf{a}$

$i = 3, j = 2$: $C \rightarrow \underline{B}\ C\ B \mid \mathbf{a}\ B \mid C\ C \mid \mathbf{a}$

$\Rightarrow \quad C \rightarrow \underline{C\ A\ B_R}\ C\ B \mid \underline{\mathbf{a}\ \mathbf{b}\ B_R}\ C\ B \mid \mathbf{a}\ B \mid C\ C \mid \mathbf{a}$

$\Rightarrow_{\text{(imm)}} \quad C \rightarrow \mathbf{a}\ \mathbf{b}\ B_R\ C\ B\ C_R \mid \mathbf{a}\ B\ C_R \mid \mathbf{a}\ C_R$

$\quad C_R \rightarrow A\ B_R\ C\ B\ C_R \mid C\ C_R \mid \varepsilon$

Example Left Recursion Elim.

$$
\left.\begin{aligned}
A &\rightarrow B\ C \mid \mathbf{a} \\
B &\rightarrow C\ A \mid A\ \mathbf{b} \\
C &\rightarrow A\ B \mid C\ C \mid \mathbf{a}
\end{aligned}\right\}\ \text{Choose arrangement: } A, B, C
$$

$i = 1$: nothing to do

$i = 2, j = 1$: $B \rightarrow C\ A \mid \underline{A}\ \mathbf{b}$

$\Rightarrow \quad B \rightarrow C\ A \mid \underline{B\ C}\ \mathbf{b} \mid \underline{\mathbf{a}}\ \mathbf{b}$

$\Rightarrow_{\text{(imm)}} \quad B \rightarrow C\ A\ B_R \mid \mathbf{a}\ \mathbf{b}\ B_R$

$\quad B_R \rightarrow C\ \mathbf{b}\ B_R \mid \varepsilon$

$i = 3, j = 1$: $C \rightarrow \underline{A}\ B \mid C\ C \mid \mathbf{a}$

$\Rightarrow \quad C \rightarrow \underline{B\ C}\ B \mid \underline{\mathbf{a}}\ B \mid C\ C \mid \mathbf{a}$

$i = 3, j = 2$: $C \rightarrow \underline{B}\ C\ B \mid \mathbf{a}\ B \mid C\ C \mid \mathbf{a}$

$\Rightarrow \quad C \rightarrow \underline{C\ A\ B_R}\ C\ B \mid \underline{\mathbf{a}\ \mathbf{b}\ B_R}\ C\ B \mid \mathbf{a}\ B \mid C\ C \mid \mathbf{a}$

$\Rightarrow_{\text{(imm)}} \quad C \rightarrow \mathbf{a}\ \mathbf{b}\ B_R\ C\ B\ C_R \mid \mathbf{a}\ B\ C_R \mid \mathbf{a}\ C_R$

$\quad C_R \rightarrow A\ B_R\ C\ B\ C_R \mid C\ C_R \mid \varepsilon$

Left Factoring

- When a nonterminal has two or more productions whose right-hand sides start with the same grammar symbols, the grammar is not LL(1) and cannot be used for predictive parsing
- Replace productions

$$A \rightarrow \alpha\, \beta_1 \mid \alpha\, \beta_2 \mid \ldots \mid \alpha\, \beta_n \mid \gamma$$

  with

$$A \rightarrow \alpha\, A_R \mid \gamma$$
$$A_R \rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$$

- Example:

$$S \rightarrow i\ E\ t\ S \mid i\ E\ t\ S\ e\ S \mid a \quad \Rightarrow \quad \begin{aligned} S &\rightarrow i\ E\ t\ S\ S' \mid a \\ S' &\rightarrow e\ S \mid \epsilon \end{aligned}$$

4. Top-Down Parsing

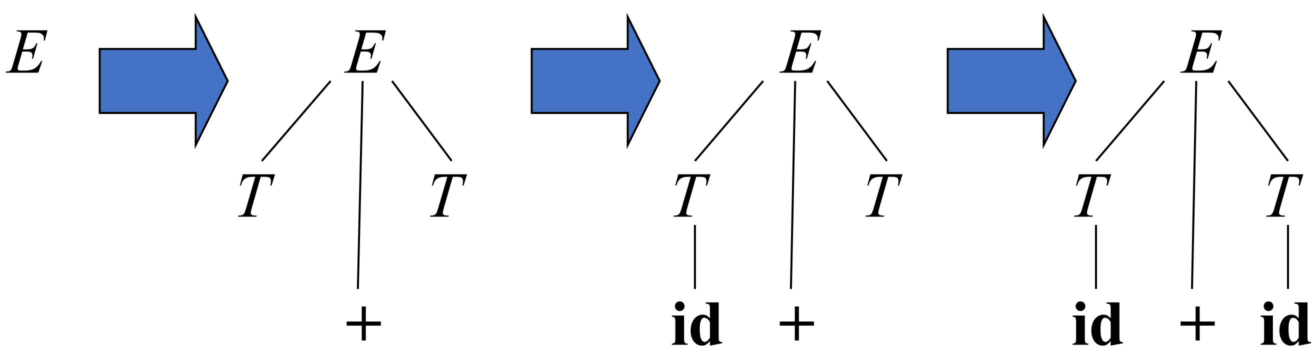
- Constructing a parse tree for the input string, starting from the root and creating the nodes of the parse tree in preorder
- Equivalently, finding the leftmost derivation for the input string

Grammar:

$$E \rightarrow T + T$$
$$T \rightarrow (\, E \,)$$
$$T \rightarrow \mathbf{-}\, E$$
$$T \rightarrow \mathbf{id}$$

Leftmost derivation:

$$E \Rightarrow_{lm} T + T$$
$$\Rightarrow_{lm} \mathbf{id} + T$$
$$\Rightarrow_{lm} \mathbf{id} + \mathbf{id}$$



Parsing Methods

- *Universal* (any C-F grammar)
  - Cocke-Younger-Kasimi
  - Earley
- *Top-down* (C-F grammar with restrictions)
  - Recursive descent (predictive parsing)
  - LL (Left-to-right, Leftmost derivation) methods
- *Bottom-up* (C-F grammar with restrictions)
  - Operator precedence parsing
  - LR (Left-to-right, Rightmost derivation) methods
    - SLR, canonical LR, LALR

Predictive Parsing

- Eliminate left recursion from grammar
- Left factor the grammar
- Compute FIRST and FOLLOW
- Two variants:
  - Recursive (recursive-descent parsing)
  - Non-recursive (table-driven parsing)
- LL(k) class of grammars
  - It can be used to construct predictive parsers looking k symbols ahead in the input.

FIRST Set

- FIRST(α) = { *terminals that begin strings derived from* α }

FIRST(a) = {a}   if $a \in T$
FIRST(ε) = {ε}
FIRST(A) = $\cup_{A\to\alpha}$ FIRST(α) for $A\to\alpha \in P$
FIRST($X_1X_2\ldots X_k$) =
  if for all j = 1, …, i-1 : $\varepsilon \in$ FIRST(X_j) **then**
    add non-ε in FIRST(X_i) to FIRST($X_1X_2\ldots X_k$)
  if for all j = 1, …, k : $\varepsilon \in$ FIRST(X_j) **then**
    add ε to FIRST($X_1X_2\ldots X_k$)

FOLLOW Set

- FOLLOW(A) = { *the set of terminals that can immediately follow nonterminal A* }

FOLLOW(A) =

for all ($B \to \alpha\ A\ \beta$) $\in P$ **do**

  add FIRST(β)\{ε} to FOLLOW(A)

for all ($B \to \alpha\ A\ \beta$) $\in P$ and $\varepsilon \in$ FIRST(β) **do**

  add FOLLOW(B) to FOLLOW(A)

for all ($B \to \alpha\ A$) $\in P$ **do**

  add FOLLOW(B) to FOLLOW(A)

if A is the start symbol S **then**

  add **$** to FOLLOW($A$)

Example

$$
\begin{aligned}
E &\rightarrow T\ E' \\
E' &\rightarrow +\ T\ E' \mid \epsilon \\
T &\rightarrow F\ T' \\
T' &\rightarrow *\ F\ T' \mid \epsilon \\
F &\rightarrow (\ E\ ) \mid \mathbf{id}
\end{aligned}
$$

FIRST(F)
= FIRST((E)) ∪ FIRST(**id**)
= FIRST(( ) ∪ {**id**}
= {**(**} ∪ {**id**} = {**(**, **id**}

FIRST(T)
= FIRST(F) = {**(**, **id**}

FIRST(E)
= FIRST(T) = {**(**, **id**}

FIRST(E')
FIRST(+TE') ∪ FIRST(ε)
= {+, **ε**}

FIRST(T')
FIRST(*FT') ∪ FIRST(ε)
= {*, **ε**}

Example

$$
\begin{array}{lcl}
E & \rightarrow & T\ E' \\
E' & \rightarrow & +\ T\ E'\ |\ \epsilon \\
T & \rightarrow & F\ T' \\
T' & \rightarrow & *\ F\ T'\ |\ \epsilon \\
F & \rightarrow & (\ E\ )\ |\ \mathbf{id}
\end{array}
$$

FIRST(E') = {+, **ε**}

FIRST(T') = {*, **ε**}

FOLLOW(E) = {**)**, **$**}

FOLLOW(E') = FOLLOW(E) = {**)**, **$**}

FOLLOW(T) = (FIRST(E')\{**ε**}) ∪ FOLLOW(E) = {+, **)**, **$**}

FOLLOW(T') = FOLLOW(T) = {+, **)**, **$**}

FOLLOW(F) = (FIRST(T')\{**ε**}) ∪ FOLLOW(T) = {+, *, **)**, **$**}

LL(1) Grammar

- Predictive parsers, that is, recursive-descent parsers needing no backtracking, can be constructed for a class of grammars called LL(1)
- A grammar *G* is LL(1) if it is not left recursive and for each collection of productions

  $$A \rightarrow \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_n$$

  for nonterminal *A* the following holds:

1. $\text{FIRST}(\alpha_i) \cap \text{FIRST}(\alpha_j) = \emptyset$ for all $i \neq j$
2. if $\alpha_i \Rightarrow^* \varepsilon$ then
   - 2.a. $\alpha_j \not\Rightarrow^* \varepsilon$ for all $j \neq i$
   - 2.b. $\text{FIRST}(\alpha_j) \cap \text{FOLLOW}(A) = \emptyset$ for all $j \neq i$

Non-LL(1) Examples

Grammar	*Not LL(1) because:*
$S \to S\ \mathbf{a} \mid \mathbf{a}$	Left recursive
$S \to \mathbf{a}\ S \mid \mathbf{a}$	FIRST($\mathbf{a}\ S$) $\cap$ FIRST($\mathbf{a}$) $\neq \emptyset$
$S \to \mathbf{a}\ R \mid \varepsilon$ $R \to S \mid \varepsilon$	For R: $S \Rightarrow^* \varepsilon$ and $\varepsilon \Rightarrow^* \varepsilon$
$S \to \mathbf{a}\ R\ \mathbf{a}$ $R \to S \mid \varepsilon$	For R: FIRST(S) $\cap$ FOLLOW(R) $\neq \emptyset$
$S \to \mathbf{i}\ E\ \mathbf{t}\ S\ S' \mid \mathbf{a}$ $S' \to \mathbf{e}\ S \mid \varepsilon$ $E \to \mathbf{b}$	For S': FIRST($\mathbf{e}\ S$) $\cap$ FOLLOW(S') $\neq \emptyset$