Chapter 5

LL Parsing

5.1 LL(k) Grammars

5.1.1 Introduction

- Subset of CFG’s
- Permit deterministic left-to-right top-down recognition with a look ahead of $k$ symbols
- Building a tree top down
- If the correct production can be deduced from the partially constructed tree and the next $k$ symbols in the unscanned string, for every possible top-down parsing step, then the grammar is said to be LL($k$)
- If a parse table can be constructed for a grammar then it is LL($k$), if it can’t, it is not LL($k$)

5.1.2 Properties of LL($k$)

1. Each LL($k$) grammar is unambiguous
2. An LL($k$) grammar has no left-recursion
   - Why is this a problem?
   - Keep expanding top nonterminal on the stack without consuming any input
   - An erroneous string will cause this to happen

5.1.3 Problems in LL($k$) parsing

1. Left recursion
2. Order of alternatives is important
3. Failure
Algorithm 5.1 Elimination of Left Recursion

{ Restrictions: Grammar has no cycles (A ⇒+ A) or ε-productions. }

Arrange non-terminals of G in some order A₁, A₂, ..., Aₙ
for i := 1 to n do
  for j := 1 to i - 1 do
    Replace each production of the form Aᵢ → Aⱼγ by the production
    Aᵢ → δ₁ γ | δ₂ γ | ... | δᵢ γ, where Aⱼ → δ₁ | δ₂ | ... | δᵢ are all the current Aⱼ-productions
  end for
end for

Eliminate the immediate left-recursion among the Aᵢ-productions

Figure 5.1: Algorithm for the elimination of left recursion in LL grammars

5.1.4 Left Recursion

• Consider the problems with LL(k) parsers. Consider the grammar

  \[ A \rightarrow \beta \mid A\alpha \]

  and try to parse the string β α.
  
  - Show the string β α
  - Show the erroneous string α α - gets into an infinite loop

• If we removed the left recursion the grammar becomes

  \[ A \rightarrow \beta A' \]
  \[ A' \rightarrow \alpha A' \mid \epsilon \]

  which derives the same string but towards the right instead of the left

• Now parse α α. β doesn’t match α, therefore try another alternative for A. There are none, so parse fails. With right recursion, we will be matching part of the input string as we go along

• Left recursion indicates we are building the string from right-to-left

• To eliminate left recursion, we turn it into right recursion and build the string left-to-right

• Algorithm to eliminate left recursion is shown in Figure 5.1.

• Eliminating left recursion at one level is done by

  \[ A \rightarrow A\alpha₁ \mid A\alpha₂ \mid ... \mid A\alphaₘ \mid \beta₁ \mid \beta₂ \mid ... \mid \betaₙ \]

  where no βᵢ, 1 ≤ i ≤ n, begins with A. Turn the above into

LL Parsing
\[
\begin{align*}
S & \rightarrow Aa \mid b & A & \rightarrow Ac \mid Sd \mid c \\
& & A & \rightarrow Ac \mid Aad \mid bd \mid c \\
S & \rightarrow Aa \mid b & A & \rightarrow bdA' \mid cA' \\
& & A' & \rightarrow cA' \mid adA' \mid \epsilon \\
A' & \rightarrow \beta_1A' \mid \beta_2A' \mid \ldots \mid \beta_nA' \\
& & \alpha_1A' \mid \alpha_2A' \mid \ldots \mid \alpha_mA' \mid \epsilon
\end{align*}
\]

- Applying the preceding algorithm to the grammar in Figure 5.2(a) gives Figure 5.2(b).
- Consider the grammar in Figure 5.3(a).
- Removing one level of recursion gives Figure 5.3(b).
- Remove left recursion gives Figure 5.3(c).

### 5.1.5 Left Factoring

- Left factoring is like factoring in mathematics. Take the common parts of productions and form a new nonterminal.
- This allows us to defer, until later, which alternative to take.
5.1.6 Order of Alternatives

- The order in which alternatives are considered can affect the language accepted. Consider the grammar shown in Figure 5.4(a).

- The string \( cabd \) may not be accepted. Consider the parse tree shown in Figure 5.4(b). \( ca \) has already matched. When the next input symbol does not match, it implies that the alternative \( cAd \) for \( S \) was wrong leading to the rejection of \( cabd \).

5.1.7 Failure

- When failure is reported, we have very little idea where the error actually occurred.

5.2 Deterministic LL(1) Parsers

5.2.1 Introduction

- We will only concern ourselves with LL(1) grammars for the following two reasons:
  1. Table size grows exponentially with \( k \)
  2. If a grammar is not LL(1), then it usually is not LL(k) for any \( k \).

- Why was push-down automaton previously given nondeterministic? Given a nonterminal on top of the stack, there are many choices as to which right hand side to choose, e.g., \( A \rightarrow \alpha_1 | \alpha_2 | ... | \alpha_k \).

- To make the parser deterministic, we must have a table to tell which alternative to choose. Given the nonterminal on top of the stack and the next \( k \) input symbols, we can uniquely select a production \( A \rightarrow \alpha_i \). Such a table is called an LL(k) selector table.
5.2.2 Example Selection Table

- Consider Figure 5.5 that shows a selector table for the grammar generating arithmetic expressions.

- Notice this grammar has been modified by the transformations talked about previously. For example, left recursion has been removed.

5.2.3 LL(1) Parsing Algorithm

- The algorithm for the LL(1) parse is shown in Figure 5.6.

5.2.4 Example

- Let's parse the string $a * (a + a)$. The parse is shown in Figure 5.7.

5.2.5 Selector Table Construction

- The algorithm for constructing the selector table is shown in Figure 5.8.

- If any pair $(A, x)$ maps to two or more different productions, then the grammar cannot be LL(1); we say that we have a conflict.

- Consider the grammar shown in Figure 5.9(a).

- The selector table is shown in Figure 5.9(b).

- Obviously the grammar is not LL(1)

- Let's consider another example. The grammar is shown in Figure 5.10(a), the First and Follow sets are shown in Figure 5.10(b), and the selector table is shown in Figure 5.10(c).
Algorithm 5.2 LL(1) Parse Algorithm

{ Input: A string $\omega$ and a parsing table $M$ for grammar $G$. }
{ Output: If $\omega$ is in $L(G)$, a leftmost derivation of $\omega$; otherwise, an error indication. }

Initially, the parser is in a configuration in which it has $S$ on the stack with $S$, the start symbol of $G$ on top, and $\omega$ in the input buffer.

Set ip to point to the first symbol of $\omega$.

repeat
  Let $X$ be the top stack symbol and $a$ the symbol pointed to by ip.
  if $X \in V_t$ or $\$$
    if $X = a$
      Pop $X$ from the stack and advance ip.
    else
      error()
    end if
  else
    \{ $X$ is a nonterminal \}
    if $M[X, a] = X \rightarrow Y_1 Y_2 \cdots Y_k$
      Pop $X$ from the stack
      Push $Y_k, Y_{k-1}, \cdots, Y_1$ onto the stack, with $Y_1$ on top
      Output the production $X \rightarrow Y_1 Y_2 \cdots Y_k$
    else
      error()
    end if
  end if
until $X = \$$  \{ stack is empty \}

Figure 5.6: LL(1) Parsing Algorithm
Figure 5.9: Example of selector table construction

Figure 5.10: Another example of selector table construction