CS 420 - Compilers

Dr. Chen-Yeou (Charles) Yu

- Writing a Grammar (4.3)
  - ~~Lexical Versus Syntactic Analysis (4.3.1)~~
  - ~~Eliminating Ambiguity (4.3.2)~~
  - Elimination of Left Recursion (4.3.3)
  - Left Factoring (4.3.4)
- Top-Down Parsing (4.4)
  - **Recursive Decent Parsing (4.4.1) (TBD, in Part6)**
- **Bottom-up Parsing (4.5) (TBD, in Part6)**

Elimination of Left Recursion (4.3.3)

- Here is the formal definition of left recursive

A grammar is *left recursive* if it has a nonterminal A such that there is a derivation $A \stackrel{+}{\Rightarrow} A\alpha$ for some string α. Top-down parsing methods cannot handle left-recursive grammars, so a transformation is needed to eliminate left recursion

- Still remember this?

$\stackrel{+}{\Rightarrow}$ means, "derives in one or more steps."

- We used to study the case in section 2.4.5 about the immediate left recursion, but this time we are going to study a more general case

Elimination of Left Recursion (4.3.3)

- In section 2.4.5, we showed how the left-recursive pair of productions

$$A \to A\alpha \mid \beta$$

could be replaced by the non-left-recursive productions

$$\begin{aligned} A &\to \beta A' \\ A' &\to \alpha A' \mid \epsilon \end{aligned}$$

, by introducing A’ and epsilon, without changing the strings derivable from A

Elimination of Left Recursion (4.3.3)

- Rule1: (epsilon is a good tool to tell the derivation, when to stop)
  - Immediate left recursion can be eliminated by the following technique, which works for any number of A-productions.
  - The original production may looks like this: (we have alpha_m, beta_n)

$$A \rightarrow A\alpha_1 \;|\; A\alpha_2 \;|\; \cdots \;|\; A\alpha_m \;|\; \beta_1 \;|\; \beta_2 \;|\; \cdots \;|\; \beta_n$$

, where no “beta_i” begins with an A.

  - Then, we can replace the A-productions by:

$$\begin{aligned} A &\rightarrow \beta_1 A' \;|\; \beta_2 A' \;|\; \cdots \;|\; \beta_n A' \\ A' &\rightarrow \alpha_1 A' \;|\; \alpha_2 A' \;|\; \cdots \;|\; \alpha_m A' \;|\; \epsilon \end{aligned}$$

Elimination of Left Recursion (4.3.3)

- Example1: considering this simple grammar

$E \to E + T \mid T$

  - This is obviously left recursive, because from E on the RHS, it can grow some other T if I put some other derivations starts with T
  - The way to eliminate the recursion is to introduce an E’, as well as an “epsilon”

    $E \to E + T \mid T$ are replaced

    by $E \to T\ E'$ and $E' \to +\ T\ E' \mid \epsilon$.

Elimination of Left Recursion (4.3.3)

- Another example (Example2)

Example: Assume n=m=1, α_1 is + and β_1 is *. So the left recursive grammar is

$$A \rightarrow A + \mid *$$

and the non-left recursive grammar is

$$A \rightarrow * A'$$
$$A' \rightarrow + A' \mid \varepsilon$$

With the recursive grammar, we have the following derivation.

$$A \Rightarrow A + \Rightarrow A + + \Rightarrow * + +$$

With the non-recursive grammar we have

$$A \Rightarrow * A' \Rightarrow * + A' \Rightarrow * + + A' \Rightarrow * + +$$

This procedure removes direct left recursion where a production with A on the left hand side begins with A on the right.

Left Factoring (4.3.4)

- Left factoring is a grammar transformation (trick), that is useful for producing a grammar suitable for predictive, or top-down, parsing
- When the choice between two alternative A-productions is not clear, we may be able to rewrite the productions to defer the decision, until enough of the input has been seen that we can make the right choice
- Here is an example on the next page

Left Factoring (4.3.4)

- Example1: (It is just a kind of re-writing skill)

if $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2$ are two A-productions, and the input begins with a nonempty string derived from α, we do not know whether to expand A to $\alpha\beta_1$ or $\alpha\beta_2$. However, we may defer the decision by expanding A to $\alpha A'$.

Then, after seeing the input derived from α, we expand A' to β_1 or to β_2. That is, left-factored, the original productions become

$$
\begin{aligned}
&A \rightarrow \alpha A' \\
&A' \rightarrow \beta_1 \mid \beta_2
\end{aligned}
$$

Top-Down Parsing (4.4)

- We did an example of top down parsing, namely predictive parsing, in chapter 2.
- For top down parsing, we
  - Start with the root of the parse tree, which is always the start symbol of the grammar. That is, initially the parse tree is just the start symbol.
  - Choose a nonterminal in the frontier.
    - Choose a production having that nonterminal as LHS.
    - Expand the tree by making the RHS the children of the LHS.
  - Repeat above until the frontier is all terminals.
  - Hope that the frontier equals the input string.

Top-Down Parsing (4.4)

- Another problem is that the procedure “may not terminate.”