COMPILER CONSTRUCTION (Predictive parser) **Fall 2019**

Lecture 16: Aug. 19,21 2019

Instructor: K.R. Chowdhary *: Professor of CS*

16.1 *FIRST* and *FOLLOW* Functions

Construction of top-down and bottom-up parsers are aided by two functions: *FIRST* and *FOLLOW*, associated with grammar. During the top-down, they allow to choose which production to apply based on the next input symbol. During the panic mode error recovery, sets of tokens produced by *FOLLOW* can be used as synchronizations tokens.

First we define $FIRST(\alpha)$. Let α is any string of grammar symbols, to be the set of terminals that begin strings derived from α. If $\alpha \Rightarrow^* \varepsilon$, then ε is in $FIRST(\alpha)$. For example, if $A \Rightarrow^* c\gamma$, then c is in $FIRST(A)$ (see Fig. 16.1).

Regarding using *FIRST* in predictive-parsing, consider A-productions, $A \to \alpha \mid \beta$, where $FIRST(\alpha)$ and $FIRST(\beta)$ are disjoint sets. We can then choose between A-productions by looking at the next input symbol, say a. Since a can be in at most one of $FIRST(\alpha)$ or $FIRST(\beta)$, not both. For instance, if a is in $FIRST(\beta)$, choose $A \to \beta$.

$FOLLOW(A)$ for non-terminal A is set of terminals a that can appear immediately to the right of A, in some sentential form, e.g., $S \Rightarrow^* \alpha A a \beta$ for some α, β, as in Fig. 16.1. Note that there might be symbols between A and a, at some time during the derivation, but if they exists, they derived ε and disappeared. If A can be right most symbol in some sentential form, then \$ is in $FOLLOW(A)$. The \$ is special end-marker symbol, that is assumed to be a symbol of any grammar.

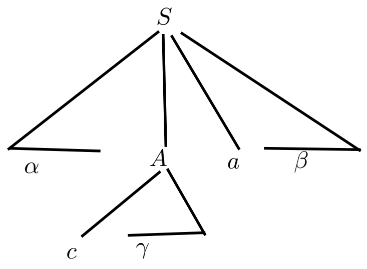

Figure 16.1: c is in $FIRST(A)$, a is in $FOLLOW(A)$

To compute $FIRST(A)$ for all grammar symbols X, we apply the following rules until no more terminals or ε can be added to any *FIRST* set.

1. If X is terminal then $FIRST(X) = \{X\}$.

2. If X is non-terminal, and $X \to Y_1 Y_2 ... Y_k$ is a production for $k \geq 1$, then place a in $FIRST(X)$ if for some i, a is in $FIRST(Y_i)$, and ε is in all of $FIRST(Y_1), ..., FIRST(Y_{i-1})$; i.e., $Y_1 ... Y_{i-1} \Rightarrow^* \varepsilon$. If ε is in $FIRST(Y_j)$ for all $j = 1, 2, ..., k$, then add ε to $FIRST(X)$. For example, every thing in $FIRST(Y_1)$ is surely in $FIRST(X)$. If Y_1 does not derive ε, then we add nothing more to $FIRST(X)$, but if $Y_1 \Rightarrow^* \varepsilon$, then we add $FIRST(Y_2)$, and so on.

3. If $X \to \varepsilon$ is a production, then add ε to $FIRST(X)$.

Now, we compute $FIRST$ for any string $X_1 X_2 ... X_n$ as follows. Add to $FIRST(X_1 X_2 ... X_n)$ all non-ε symbols of $FIRST(X_1)$. Also, add the non-ε symbols of $FIRST(X_2)$, if ε is in $FIRST(X_1)$, and so on. Finally, add ε to $FIRST(X_1 X_2 ... X_n)$ if, for all i, ε is in $FIRST(X_i)$.

To compute $FOLLOW(A)$ for all non-terminals A, apply the following rules until nothing can be added to any $FOLLOW$ set.

1. Place \$ in $FOLLOW(S)$, where S is start symbol, and \$ is the input right end-marker.

2. If there is a production $A \to \alpha B \beta$, then everything in $FIRST(\beta)$, except ε, is in $FOLLOW(B)$.

3. If there is a production $A \to \alpha B$, or production $A \to \alpha B \beta$, where $FIRST(\beta)$ contains ε, then every thing in $FOLLOW(A)$ is in $FOLLOW(B)$.

Example 16.1 *Find the FIRST and FOLLOW for given non-recursive grammar.*

$$\begin{aligned} E &\to T\ E' \\ E' &\to +T\ E' \mid \varepsilon \\ T &\to F\ T' \\ T' &\to \times F\ T' \mid \varepsilon \\ F &\to (E) \mid id \end{aligned} \tag{16.1}$$

1. $FIRST(F) = FIRST(T) = \{(, id\}$, because, the two productions for F have bodies that start with "(" and "id". The T has only one production, and its body starts with F. Since, F does not derive ε, FIRST(T) must be same as FIRST(F). The same argument covers FIRST(E) = $\{(, id\}$.

2. $FOLLOW(E) = FOLLOW(E') = \{), \$\}$. Since E is start symbol, $FOLLOW(E)$ must contain \$. The production body (E) explains why the right parenthesis is in $FOLLOW(E)$. For E', this non-terminal appears only at the ends of the bodies of E-productions. Thus $FOLLOW(E')$ must be same as $FOLLOW(E)$. □

16.2 LL(1) Grammars

Predictive parsers, i.e., recursive-descent parsers requires no backtracking. They can be constructed for a class of grammars called LL(1). The first "L" in the LL(1) grammar stand for scanning the input from left to right, and the second "L" for producing a leftmost derivation, and "1" stands for using one input symbol for look ahead at each step to make parsing action decisions.

The LL(1) grammars are rich enough to cover most programming constructs, although it requires writing suitable grammar for the source language. A non-left-recursive or ambiguous grammar can be LL(1). A grammar G is LL(1) if and only if whenever $A \to \alpha \mid \beta$ are two different productions of G, the following conditions hold:

1. For no terminal a do both α and β derive strings beginning with a.
2. At the most one of α and β can derive the empty string.
3. If $\beta \Rightarrow^* \varepsilon$, then α does not derive any string beginning with a terminal in $FOLLOW(A)$. Similarly, if $\alpha \Rightarrow^* \varepsilon$, then β does not derive any string beginning with a terminal in $FOLLOW(A)$.

The first two conditions are equivalent to the statement that $FIRST(\alpha)$ and $FIRST(\beta)$ are disjoint sets. The third condition is equivalent to stating that if ε is in $FIRST(\beta)$, then $FIRST(\alpha)$ and $FOLLOW(A)$ are disjoint sets, and likewise if ε is in $FIRST(\alpha)$.

The predictive parsers can be constructed for LL(1) grammars since the production to apply for a non-terminal can be selected by looking only at the current input symbol. Flow-of-control constructs, with their distinguishing keywords, generally satisfy the LL(1) constraints. For example, we have productions:

$$\begin{aligned} \textit{stmt} \to\ & \textbf{if}\ (\ \textit{expr}\ )\ \textit{stmt}\ \textbf{else}\ \textit{stmt} \\ & \mid \textbf{while}\ (\ \textit{expr}\ )\ \textit{stmt} \\ & \mid \{\ \textit{stmt-list}\ \} \end{aligned}$$

then keywords **if, while** and the symbol '{', tell us which alternative is the only one that could be successor if we have to find the correct statement. The reader may verify in every high-level language any two keywords are never starting with same first character. This is because the language designers choose keywords such that every keyword starts with unique first character.

16.3 Predictive parsing

A predictive parser is a recursive descent parser that has no backtracking. The predictive parser can also be automatically generated using tools like, ANTLR [**?**]. ANTLR is a successor to the Purdue Compiler Constructor Tool Set (PCCTS), first developed in 1989. It is written in Java.

The predictive parsing algorithm collects the information from $FIRST$ and $FOLLOW$ sets into a *predictive parsing table* $M[A, a]$ – a two-dimensional array, where A is a non-terminal, and a is a terminal or the the end-marker symbol \$. The algorithm for construction of this table is based on following idea:

1. The production $A \rightarrow \alpha$ is chosen as entry in $M[A, a]$ if a is the next input symbol in $FIRST(\alpha)$.
2. The only complication occurs when $\alpha = \varepsilon$ or, more generally $\alpha \Rightarrow^* \varepsilon$. In this case, we should again choose $A \rightarrow \alpha$, if the current input symbol is in $FOLLOW(A)$, or if the input pointer has reached to \$, hence \$ is in $FOLLOW(A)$.

The Algorithm 1 is the algorithm for construction of predictive parsing table.

Algorithm 1 Construction of predictive parsing table(Input: Grammar G, Output: Parsing table M)

```
1: For each terminal $a$ in $FIRST(A)$(where $A$ is current input), add $A \rightarrow \alpha$ to $M[A, a]$
2: If $\varepsilon \in FIRST(\alpha)$, then for each terminal $b$ in $FOLLOW(A)$, add $A \rightarrow \alpha$ to $M[A, b]$.
3: If $\varepsilon \in FIRST(\alpha)$ and $\$ \in FOLLOW(A)$, add $A \rightarrow \alpha$ to $M[A, \$]$.
```

If, after performing the above, there is no production at all in position $M[A, a]$, then set $M[A, a]$ to *error*. The error is usually represented by empty entry in the table. The other alternatives we will discuss later.

Example 16.2 *Construct parsing table for the expression grammar represented by Equation (16.1), using Algorithm-1 (page no. 1).*

In the parsing-table 16.1, the blanks are error entries, and non-blanks indicate a production with which to expand a non-terminal.

Table 16.1: predictive Parsing table $M[A, a]$, where A is terminal and a is non-terminal.

Non-terminal	$+$	$\times$	(	id	)	\$
E			$E \rightarrow TE'$	$E \rightarrow TE'$		
E'	$E' \rightarrow +TE'$				$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T			$T \rightarrow FT'$	$T \rightarrow FT'$		
T'	$T' \rightarrow \varepsilon$	$T' \rightarrow \times FT'$			$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F			$F \rightarrow (E)$	$F \rightarrow id$		

The explanation of entries in table above are as follows:

For the production $E \rightarrow TE'$, since $FIRST(TE') = FIRST(T) = \{(, id\}$, this production is added to $M[E, (]$ and $M[E, id]$. The production $E' \rightarrow +TE'$ is added $M[E', +]$, since $FIRST(+TE') = \{+\}$. Since $FOLLOW(E') = \{), \$\}$, the production $E' \rightarrow \varepsilon$ is added to $M[E',)]$ and $M[E', \$]$.

For production $T' \rightarrow \times FT'$, the $\times \in FIRST(T')$, when we put $T' \rightarrow \times FT'$ in $M[T', \times]$, we note that there is also other production $T' \rightarrow \varepsilon$. For this, we require $FOLLOW$ as follows: $FOLLOW(T') = FOLLOW(FT') = FOLLOW(T) = FIRST(E') =$ '+', as per grammar 16.1. Also, $FOLLOW(E') = FOLLOW(TE')$, hence $\$ \in FOLLOW(TE')$. □

16.4 Review Questions

1. Can the values in $FIRST$ and $FOLLOW$ be non-terminals?

2. Find out the $FIRST(A)$ in the following cases:

   (a) $A \rightarrow aA$
   (b) $At\ aAB$
   (c) $A \rightarrow BbA$
   (d) $A \rightarrow \varepsilon$

16.5 Exercises

1. Construct predictive parsing table for following grammars:
   (a) $S \rightarrow aSa \mid bSb \mid \varepsilon$
   (b) $S \rightarrow aSb \mid \varepsilon$
2. Given the productions for a grammar $S \rightarrow AS \mid a$, $A \rightarrow aA \mid b$, construct the predictive parser table. Indicate the errors explicitly, if any, and the reasons for their occurrence.

References

[1] Earley, Jay (1970), "An efficient context-free parsing algorithm" (PDF), Communications of the ACM, 13 (2): 94-102, doi:10.1145/362007.362035

[2] Compilers: Principles, Techniques, and Tools (2nd Edition) by Alfred V. Aho, Monica S. Lam, et al., Sep 10, 2006.

[3] Compiler design in C (Prentice-Hall software series) by Allen I Holub, Jan 1, 1990.

[4] Engineering a Compiler, by Keith D. Cooper and Linda Torczon, Morgan Kaufmann Publishers, 2004.

[5] Tools for Large-scale Parser Development, Proceedings of the COLING-2000 Workshop on Efficiency In Large-Scale Parsing Systems, 2000, pp. 54-54, `http://dl.acm.org/citation.cfm?id=2387596.2387604`.