بسم اللّه الرّحمن الرّحیم

دانشگاه صنعتی اصفهان ـ دانشکدهٔ مهندسی برق و کامپیوتر
(نیم‌سال تحصیلی ۴۰۲۲)

کامپایلر

حسین فلسفین

☞ *Predictive parsing relies on information about the first symbols that can be generated by a production body.*

☞ *If we can always choose a unique production based only on the next input symbol, we are able to do predictive parsing without backtracking.*

In simple cases, the right-hand sides of all productions for any given nonterminal start with distinct terminals, except at most one production whose right-hand side does not start with a terminal (i.e., it is an empty production, or the right-hand side of the production starts with a nonterminal). We chose the production whose right-hand side does not start with a terminal whenever the input symbol does not match any of the terminal symbols that start the right-hand sides other productions. We can extend the method to work also for grammars where more than one production for a given nonterminal have right-hand sides that do not start with terminals.

We just need to be able to select between these productions based on the input symbol, even when the right-hand sides do not start with terminal symbols.

نیاز به تعریف چند مفهوم داریم

With respect to a particular grammar, given a string γ of terminals and nonterminals,

- nullable(X) is true if X can derive the empty string.
- FIRST(γ) is the set of terminals that can begin strings derived from γ.
- FOLLOW(X) is the set of terminals that can immediately follow X. That is, $t \in$ FOLLOW(X) if there is any derivation containing Xt. This can occur if the derivation contains $XYZt$ where Y and Z both derive ϵ.

نکته مهم دربارهٔ انتخاب رول مناسب

We choose a production $N \to \alpha$ on input symbol c if either

- ☞ $c \in \mathrm{FIRST}(\alpha)$, ***or***
- ☞ $Nullable(\alpha)$ ***and*** $c \in \mathrm{FOLLOW}(N)$.

If we can always choose a production uniquely by using these rules, this is called $\mathrm{LL}(1)$ parsing—the first L indicates the reading direction (left-to-right), the second L indicates the derivation order (left), and the (1) indicates that there is a one-symbol lookahead, i.e., that decisions require looking only at one input symbol (the next input symbol). A grammar where strings can be unambiguously parsed or rejected using $\mathrm{LL}(1)$ parsing is called an$\mathrm{LL}(1)$ grammar.

مثال‌های دیگری از تجزیهٔ پیش‌بین به‌صورت بازگشتی (کتاب اپل)

$$S \rightarrow E\ \$$$

	$T \rightarrow T * F$	$F \rightarrow \mathrm{id}$
$E \rightarrow E + T$	$T \rightarrow T\ /\ F$	$F \rightarrow \mathrm{num}$
$E \rightarrow E - T$	$T \rightarrow F$	$F \rightarrow (\ E\ )$
$E \rightarrow T$		

GRAMMAR 3.10.

	$L \rightarrow \mathrm{end}$
$S \rightarrow \mathrm{if}\ E\ \mathrm{then}\ S\ \mathrm{else}\ S$	$L \rightarrow\ ;\ S\ L$
$S \rightarrow \mathrm{begin}\ S\ L$	
$S \rightarrow \mathrm{print}\ E$	$E \rightarrow \mathrm{num} = \mathrm{num}$

GRAMMAR 3.11.

Grammar 3.11 (استفاده از رویکرد موفق است)

```
final int IF=1, THEN=2, ELSE=3, BEGIN=4, END=5, PRINT=6,
          SEMI=7, NUM=8, EQ=9;

int tok = getToken();

void advance() {tok=getToken();}
void eat(int t) {if (tok==t) advance(); else error();}

void S() {switch(tok) {
        case IF:     eat(IF); E(); eat(THEN); S();
                     eat(ELSE); S(); break;
        case BEGIN: eat(BEGIN); S(); L(); break;
        case PRINT: eat(PRINT); E(); break;
        default:     error();
       }}
void L() {switch(tok) {
        case END:    eat(END); break;
        case SEMI:   eat(SEMI); S(); L(); break;
        default:     error();
       }}
void E() {  eat(NUM); eat(EQ); eat(NUM); }
```

Grammar 3.10 (استفاده از رویکرد موفق نیست)

```
void S() {  E(); eat(EOF); }
void E() {switch (tok) {
          case ?: E(); eat(PLUS); T(); break;
          case ?: E(); eat(MINUS); T(); break;
          case ?: T(); break;
          default: error();
         }}
void T() {switch (tok) {
          case ?: T(); eat(TIMES); F(); break;
          case ?: T(); eat(DIV); F(); break;
          case ?: F(); break;
          default: error();
         }}
```

تابع نظیر متغیر F را هم به همین نحو می‌توان نوشت. اما رویکرد تجزیهٔ پیش‌بین برای این گرامر قابل‌استفاده نیست. چرا؟

There is a conflict here (i.e., in the recursive descent parser for Grammar 3.10): The E function has no way to know which clause to use. Consider the strings $(1 * 2 - 3) + 4$ and $(1 * 2 - 3)$. In the former case, the initial call to E should use the $E \rightarrow E + T$ production, but the latter case should use $E \rightarrow T$.

Recursive-descent, or predictive, parsing works only on grammars where the first terminal symbol of each subexpression provides enough information to choose which production to use.

در ادامه، به‌صورت دقیق‌تری مفاهیم FIRST و FOLLOW را معرفی کرده، پیاده‌سازی مبتنی بر جدول را وصف می‌کنیم.

FIRST *and* FOLLOW

From the FIRST ***and*** FOLLOW ***sets for a grammar, we shall construct "predictive parsing tables," which make explicit the choice of production during top-down parsing.***

The construction of both top-down and bottom-up parsers is aided by two functions, FIRST ***and*** FOLLOW, ***associated with a grammar*** G. ***During top-down parsing,*** FIRST ***and*** FOLLOW ***allow us to choose which production to apply, based on the next input symbol.***

$\mathrm{FIRST}(\alpha)$

Define** $\mathrm{FIRST}(\alpha)$, **where** 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 also in** $\mathrm{FIRST}(\alpha)$**.

For a preview of how** FIRST **can be used during predictive parsing, consider two** A**-productions** $A \rightarrow \alpha|\beta$**, where** $\mathrm{FIRST}(\alpha)$ **and** $\mathrm{FIRST}(\beta)$ **are disjoint sets. We can then choose between these** A**-productions by looking at the next input symbol** a**, since** a **can be in at most one of** $\mathrm{FIRST}(\alpha)$ **and** $\mathrm{FIRST}(\beta)$**, not both. For instance, if** a **is in** $\mathrm{FIRST}(\beta)$ **choose the production** $A \rightarrow \beta$**.

نکتهٔ مهم

If two different productions $X \rightarrow \gamma_1$ and $X \rightarrow \gamma_2$ have the same left-hand-side symbol (X) and their right-hand sides have overlapping FIRST sets, then the grammar cannot be parsed using predictive parsing. If some terminal symbol I is in $\mathrm{FIRST}(\gamma_1)$ and also in $\mathrm{FIRST}(\gamma_2)$, then the X function in a recursive-descent parser will not know what to do if the input token is I.

$\mathrm{FOLLOW}(A)$

Define $\mathrm{FOLLOW}(A)$, for nonterminal A, to be the set of terminals a that can appear immediately to the right of A in some sentential form; that is, the set of terminals a such that there exists a derivation of the form $S \to \alpha A a \beta$, for some α and β. Note that there may have been symbols between A and a, at some time during the derivation, but if so, they derived ε and disappeared. In addition, if A can be the rightmost symbol in some sentential form, then \$ is in $\mathrm{FOLLOW}(A)$; recall that \$ is a special "endmarker" symbol that is assumed not to be a symbol of any grammar.

Example

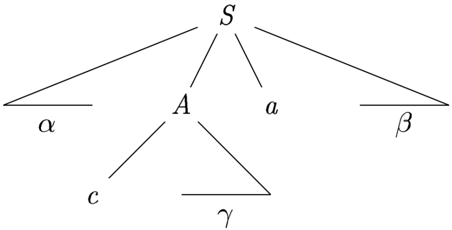

Figure 4.15: Terminal c is in FIRST(A) and a is in FOLLOW(A)

محاسبهٔ FIRST برای سمبل‌های گرامر

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

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

2. If X is a nonterminal and $X \to Y_1 Y_2 \cdots Y_k$ is a production for some $k \geq 1$, then place a in $\mathrm{FIRST}(X)$ if for some i, a is in $\mathrm{FIRST}(Y_i)$, and ϵ is in all of $\mathrm{FIRST}(Y_1), \ldots, \mathrm{FIRST}(Y_{i-1})$; that is, $Y_1 \cdots Y_{i-1} \stackrel{*}{\Rightarrow} \epsilon$. If ϵ is in $\mathrm{FIRST}(Y_j)$ for all $j = 1, 2, \ldots, k$, then add ϵ to $\mathrm{FIRST}(X)$. For example, everything in $\mathrm{FIRST}(Y_1)$ is surely in $\mathrm{FIRST}(X)$. If Y_1 does not derive ϵ, then we add nothing more to $\mathrm{FIRST}(X)$, but if $Y_1 \stackrel{*}{\Rightarrow} \epsilon$, then we add $\mathrm{FIRST}(Y_2)$, and so on.

3. If $X \to \epsilon$ is a production, then add ϵ to $\mathrm{FIRST}(X)$.

محاسبهٔ FIRST برای یک رشتهٔ دلخواه در $(V \cup \Sigma)^*$

Now, we can compute FIRST ***for any string*** $X_1X_2\cdots X_n$ ***as follows. Add to*** $\mathrm{FIRST}(X_1X_2\cdots X_n)$ ***all non-***ε ***symbols of*** $\mathrm{FIRST}(X_1)$***. Also add the non-***ε ***symbols of*** $\mathrm{FIRST}(X_2)$***, if*** ε ***is in*** $\mathrm{FIRST}(X_1)$***; the non-***ε ***symbols of*** $\mathrm{FIRST}(X_3)$***, if*** ε ***is in*** $\mathrm{FIRST}(X_1)$ ***and*** $\mathrm{FIRST}(X_2)$***; and so on. Finally, add*** ε ***to*** $\mathrm{FIRST}(X_1X_2\cdots X_n)$ ***if, for all*** i***,*** ε ***is in*** $\mathrm{FIRST}(X_i)$***.***

محاسبهٔ FOLLOW برای یک متغیر (غیرترمینال)

To compute $\mathrm{FOLLOW}(A)$ ***for all nonterminals*** A***, apply the following rules until nothing can be added to any*** FOLLOW ***set.***

1. Place \$ in $\mathrm{FOLLOW}(S)$, where S is the start symbol, and \$ is the input right endmarker.
2. If there is a production $A \to \alpha B \beta$, then everything in $\mathrm{FIRST}(\beta)$ except ϵ is in $\mathrm{FOLLOW}(B)$.
3. If there is a production $A \to \alpha B$, or a production $A \to \alpha B \beta$, where $\mathrm{FIRST}(\beta)$ contains ϵ, then everything in $\mathrm{FOLLOW}(A)$ is in $\mathrm{FOLLOW}(B)$.

Example:

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

1. $\text{FIRST}(F) = \text{FIRST}(T) = \text{FIRST}(E) = \{(, \mathbf{id}\}$. To see why, note that the two productions for F have bodies that start with these two terminal symbols, **id** and the left parenthesis. T has only one production, and its body starts with F. Since F does not derive ϵ, $\text{FIRST}(T)$ must be the same as $\text{FIRST}(F)$. The same argument covers $\text{FIRST}(E)$.

2. $\text{FIRST}(E') = \{+, \epsilon\}$. The reason is that one of the two productions for E' has a body that begins with terminal $+$, and the other's body is ϵ. Whenever a nonterminal derives ϵ, we place ϵ in FIRST for that nonterminal.

3. $\text{FIRST}(T') = \{*, \epsilon\}$. The reasoning is analogous to that for $\text{FIRST}(E')$.

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

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

5. $\text{FOLLOW}(T) = \text{FOLLOW}(T') = \{+,), \$\}$. Notice that T appears in bodies only followed by E'. Thus, everything except ϵ that is in $\text{FIRST}(E')$ must be in $\text{FOLLOW}(T)$; that explains the symbol $+$. However, since $\text{FIRST}(E')$ contains ϵ (i.e., $E' \overset{*}{\Rightarrow} \epsilon$), and E' is the entire string following T in the bodies of the E-productions, everything in $\text{FOLLOW}(E)$ must also be in $\text{FOLLOW}(T)$. That explains the symbols \$ and the right parenthesis. As for T', since it appears only at the ends of the T-productions, it must be that $\text{FOLLOW}(T') = \text{FOLLOW}(T)$.

6. $\text{FOLLOW}(F) = \{+, *,), \$\}$. The reasoning is analogous to that for T in point (5).

Iterative computation of FIRST, FOLLOW, *and* nullable

Algorithm to compute FIRST, FOLLOW, *and* nullable.

Initialize FIRST and FOLLOW to all empty sets, and nullable to all false.

for each terminal symbol Z

    FIRST[Z] $\leftarrow$ {Z}

repeat

    for each production $X \rightarrow Y_1 Y_2 \cdots Y_k$

        if $Y_1 \ldots Y_k$ are all nullable (or if $k = 0$)

          then nullable[X] $\leftarrow$ true

        for each i from 1 to k, each j from $i + 1$ to k

            if $Y_1 \cdots Y_{i-1}$ are all nullable (or if $i = 1$)

              then FIRST[X] $\leftarrow$ FIRST[X] $\cup$ FIRST[Y_i]

            if $Y_{i+1} \cdots Y_k$ are all nullable (or if $i = k$)

              then FOLLOW[Y_i] $\leftarrow$ FOLLOW[Y_i] $\cup$ FOLLOW[X]

            if $Y_{i+1} \cdots Y_{j-1}$ are all nullable (or if $i + 1 = j$)

              then FOLLOW[Y_i] $\leftarrow$ FOLLOW[Y_i] $\cup$ FIRST[Y_j]

until FIRST, FOLLOW, and nullable did not change in this iteration.