Глава 3. Нисходящий синтаксический анализ

3.5. *Метод рекурсивного спуска*

Метод рекурсивного спуска – хорошо известный и легко реализуемый детерминированный метод нисходящего разбора для $LL(k)$-грамматик. В простейшей форме метод рекурсивного спуска предоставляет удобную возможность построения синтаксического анализатора языка, порожденного $LL(1)$-грамматикой.

Самый простой (но не самый эффективный) способ реализации рекурсивного спуска заключается в следующем. Каждому символу из множества $V_T \cup V_N$ ставится в соответствие единственная процедура, распознающая строку, порождаемую этим символом.

Если $a \in V_T$, то соответствующая процедура (обозначим *proc_a*) обеспечивает проверку на совпадение очередного входного символа с терминалом a, в случае совпадения реализуется ввод следующего входного символа, в противном случае фиксируется синтаксическая ошибка.

Если $A \in V_N$ и имеются A-продукции вида $A \to \alpha_1 | \alpha_2 | ... | \alpha_n$, то соответствующая процедура (обозначим *proc_A*), сравнив очередной входной символ с элементами множеств направляющих символов, определяет, какой из A-продукций нужно воспользоваться для осуществления следующего шага вывода. Если выбрана продукция $A \to \alpha_i$, и $\alpha_i = X_1X_2...X_m$, $X_j \in V_T \cup V_N$, $1 \le j \le m$, то реализуется последовательность вызовов процедур *proc_*X_1; *proc_*X_2; ...; *proc_*X_m.

Рассмотрим $LL(1)$-грамматику со следующими продукциями, предполагая наличие продукции $S' \to S\perp$ (для каждой продукции справа указаны множества направляющих символов):

$S \to AbB$ $\{a, b, c, e\}$
$S \to d$ $\{d\}$
$A \to aAb$ $\{a\}$
$A \to edAb$ $\{e\}$
$A \to B$ $\{b, c\}$
$B \to cSd$ $\{c\}$
$B \to \varepsilon$ $\{b, d, \perp\}$

Пусть процедура *read*(*sym*) считывает из входной сроки очередной символ и присваивает его значение переменной *sym* (соответствует обращению к сканеру за очередным токеном), процедура *error* каким-либо образом обрабатывает синтаксическую ошибку и прекращает разбор, процедура *stop* завершает синтаксический разбор. Тогда синтаксический анализатор можно представить множеством соответствующих процедур.

Для всех терминалов $a \in V_T = \{a, b, c, d, e\}$ процедуры *proc_a* имеют вид:

```
procedure proc_a
   if sym ='a' then read(sym) else error
return
```

$S \to AbB$	$\{a, b, c, e\}$	$A \to aAb$	$\{a\}$	$B \to cSd$	$\{c\}$
$S \to d$	$\{d\}$	$A \to edAb$	$\{e\}$	$B \to \varepsilon$	$\{b, d, \perp\}$
		$A \to B$	$\{b, c\}$		

Процедуры для нетерминалов:

```
procedure proc_S
   if sym ∈ {a,b,c,e} then proc_A; proc_b; proc_B
   else proc_d
return

procedure proc_A
   if sym = a then proc_a; proc_A; proc_b
   else if sym = e then proc_e; proc_d; proc_A; proc_b
   else proc_B
return

procedure proc_B
   if sym = c then proc_c; proc_S; proc_d
return
```

Процесс разбора начинается со следующих действий (переменная *sym* после выполнения *read*(*sym*) будет содержать первый символ входной строки)

read(*sym*); *proc_S*; **if** *sym* = ⊥ **then** *stop* **else** *error*.

На практике с целью сокращения числа вызовов процедур такие процедуры ставят в соответствие только для нетерминалов, включая операции ввода и анализа терминалов в эти процедуры. При составлении процедур *proc_A* для всех $A \in V_N$ необходимо придерживаться следующих правил:

а) реализовать выбор из A-продукций вида $A \to \alpha_1 | \alpha_2 | ... | \alpha_n$ соответствующей продукции, сравнив очередной входной символ с элементами множеств направляющих символов; пусть выбрана продукция $A \to \alpha_i$, и $\alpha_i = X_1X_2...X_m$, $X_j \in V_T \cup V_N$, $1 \le j \le m$;

б) если $X_j \in V_N$ (текущим символом правой части продукции является нетерминал), ему соответствует вызов процедуры, распознающей порождаемую этим нетерминалом строку;

в) если $X_j \in V_T$ (текущим символом правой части продукции является терминал), ему соответствует действие, связанное с проверкой на совпадение этого терминала с текущим входным символом. Если символы совпадают, то входной символ принимается и реализуется ввод очередного входного символа, в противном случае фиксируется синтаксическая ошибка;

г) если $\alpha_i = \varepsilon$ (имеет место ε-продукция), реализуется переход на конец процедуры, т. е. никакие действия не производятся.

Учитывая эти правила, процедуры для нетерминалов рассматриваемой грамматики можно написать следующим образом:

$S \to AbB\ \{a, b, c, e\}$ $S \to d\ \{d\}$

procedure *proc_S*

if $sym \in \{a,b,c,e\}$ **then** $\begin{cases} proc\_A \\ \textbf{if } sym = b \textbf{ then } read(sym) \textbf{ else } error \\ proc\_B \end{cases}$

else if $sym = d$ **then** $read(sym)$ **else** *error*

return

$A \to aAb\ \{a\}$ $A \to edAb\ \{e\}$ $A \to B\ \{b, c\}$

procedure *proc_A*

if $sym = a$ **then** $\begin{cases} read(sym) \\ proc\_A \\ \textbf{if } sym = b \textbf{ then } read(sym) \textbf{ else } error \end{cases}$

else if $sym = e$ **then** $\begin{cases} read(sym) \\ \textbf{if } sym = d \textbf{ then } read(sym) \textbf{ else } error \\ proc\_A \\ \textbf{if } sym = b \textbf{ then } read(sym) \textbf{ else } error \end{cases}$

else *proc_B*

return

$B \to cSd$ {c} $B \to \varepsilon$ {$b, d, \perp$}

procedure *proc_B*

$$\textbf{if } sym = c \textbf{ then } \begin{cases} read(sym) \\ proc\_S \\ \textbf{if } sym = d \textbf{ then } read(sym) \textbf{ else } error \end{cases}$$

return

Для разбора строк языка может потребоваться много рекурсивных вызовов процедур, соответствующих нетерминалам грамматики. Если представить грамматику несколько иным способом, в ряде случаев рекурсию можно заменить итерацией. Например, пусть дана грамматика со следующими продукциями:

$S \to cAdBe$

$A \to afA \mid a$

$B \to bfB \mid b$

Ее можно представить следующим образом (используя нотацию регулярных выражений):

$S \to cAdBe$

$A \to a(fa)^*$

$B \to b(fb)^*$

Тогда процедуры для нетерминалов A и B можно представить следующим образом:

$A \to a(fa)^*$

procedure *proc_A*
  if $sym \neq 'a'$ **then** *error*
  read(*sym*)
  while $sym = 'f'$ **do** $\begin{cases} read(sym) \\ \textbf{if } sym \neq 'a' \textbf{ then } error \\ read(sym) \end{cases}$
return

$B \to b(fb)^*$

procedure *proc_B*
  if $sym \neq 'b'$ **then** *error*
  read(*sym*)
  while $sym = 'f'$ **do** $\begin{cases} read(sym) \\ \textbf{if } sym \neq 'b' \textbf{ then } error \\ read(sym) \end{cases}$
return

Такая замена рекурсии итерацией обычно позволяет делать анализатор более эффективным, за счет сокращения вызовов процедур.

Метод рекурсивного спуска является одним из наиболее эффективных способов создания компиляторов. Преимущества написания рекурсивного нисходящего анализатора очевидны. Основные из них – это скорость написания анализатора на основании соответствующей грамматики. Другое преимущество заключается в соответствии между грамматикой и анализатором, благодаря которому увеличивается вероятность того, что анализатор окажется правильным, или, по крайней мере, того, что ошибки будут носить простой характер.

Недостатки этого метода, хотя и менее очевидны, но не менее реальны. Из-за большого числа вызовов процедур во время синтаксического анализа анализатор становится относительно медленным. Кроме того, он может быть довольно большим по сравнению с анализаторами, основанными на табличных методах разбора. Несмотря на то, что данный метод способствует включению в анализатор действий по генерированию кода, это неизбежно приводит к смешиванию различных фаз компиляции. Это снижает надежность компиляторов или усложняет обращение с ними и привносит в анализатор зависимость от машины.