Глава 3. Нисходящий синтаксический анализ

3.6. Табличные методы нисходящего разбора

3.6.1. *Специальные LL(1)-таблицы разбора*

В ряде случаев для повышения эффективности синтаксического анализа разрабатывают специальные таблицы разбора. В качестве примера рассмотрим достаточно простой и понятный вид таблицы разбора для *LL*(1)-грамматик [18].

LL(1)-таблица разбора представляет собой набор строк. Каждая строка содержит поля (столбцы):

а) список терминалов – *terminals*,

б) поле переходов – *jump*,

в) поле приема – *accept*,

г) поле стека – *stack*,

д) поле возврата – *return*,

е) поле ошибки – *error*.

Область значений поля *jump* – неотрицательные целые числа (номера строк таблицы), а область значений полей *accept*, *stack*, *return* и *error* – {**true**, **false**}.

В таблице каждому шагу процесса разбора соответствует один элемент. Поэтому в нее включаются по одному элементу для каждой продукции грамматики (для нетерминала из левой части) и на каждый экземпляр терминала или нетерминала правой части продукции. Кроме того, в таблицу включаются элементы на каждую реализацию пустой строки в правой части продукции (для ε-продукций). Первая строка таблицы соответствует первой продукции с начальным нетерминалом в левой части. Для этого исходная *LL*(1)-грамматика представляется в виде специальной схемы, в которой каждому символу грамматики соответствует номер строки таблицы разбора. Очень важно, чтобы номера нетерминалов в левых частях альтернативных продукций непосредственно следовали друг за другом. Это необходимо для того, чтобы в случае, если анализируемый символ не принадлежит множеству направляющих символов одной продукции, увеличением номера на единицу перейти к проверке следующей альтернативной продукции.

Правила заполнения таблицы заключаются в следующем.

1. Поле *terminals*. Если строка таблицы соответствует продукции (т. е. нетерминалу в левой части), в поле заносится множество направляющих символов данной продукции. Если строка соответствует нетерминалу из правой части продукции, в поле записывается множество терминалов, являющееся объединением множеств направляющих символов всех продукций с данным нетерминалом в левой части. Если строка соответствует терминалу, в поле заносится этот терминал. Если строка соответствует правой части ε-продукции, в поле заносится множество направляющих символов этой ε-продукции.

2. Поле *jump*. В поле *jump* записывается номер строки следующего элемента обработки.

3. Поле *accept*. Устанавливается значение **true**, если строка таблицы соответствует терминалу, в противном случае – **false**. Означает, что если просматриваемый символ входной строки совпадает с данным терминалом, то этот символ можно принять и перейти к анализу следующего входного символа.

4. Поле *stack*. Устанавливается значение **true**, если строка таблицы соответствует нетерминалу из правой части продукции за исключением случая, когда этот нетерминал является крайним правым символом продукции. Поле показывает, что необходимо занести в стек точку возврата (номер строки для символа, непосредственно следующего в продукции за данным нетерминалом), поскольку далее осуществляется переход к обработке первой продукции, относящейся к данному нетерминалу. Очевидно, что если нетерминал является крайним правым символом продукции, нет необходимости помещать в стек адрес возврата. Во всех остальных случаях поле принимает значение **false.**

5. Поле *return*. Устанавливается значение **true**, если строка таблицы соответствует крайнему правому терминалу продукции или правой части ε-продукции, в остальных случаях – **false**. Означает, что завершена обработка продукции и необходимо вернуться к точке возврата, сохраненной в стеке. При этом значение поля *jump* во внимание не принимается и устанавливается нулевое значение.

6. Поле *error*. Если имеется несколько альтернативных продукций, например k, то в строках, соответствующих первым $k - 1$ продукциям, поле ошибки принимает значение **false**, для k-й продукции – **true**. Это связано с тем, что если входной символ не является направляющим, то это еще не означает ошибку, поскольку он может быть направляющим для какой-либо другой альтернативной продукции, проверяемой на следующем этапе. Если же анализируемый символ не будет направляющим ни для одной из альтернативных продукций (а это выяснится только после проверки последней из них), тогда можно констатировать наличие синтаксической ошибки. Во всех остальных случаях поле ошибки принимает значение **true.**

В процессе синтаксического анализа осуществляется ряд различных шагов:

1. Проверка текущего входного символа с целью выяснения, не является ли данный символ направляющим для какой-либо конкретной правой части продукции. Если этот символ не направляющий и имеется альтернативная продукция, то она проверяется на следующем этапе.

2. Проверка терминала, появляющегося в правой части продукции.

3. Проверка нетерминала. Она заключается в проверке нахождения текущего входного символа в одном из множеств направляющих символов для данного нетерминала, помещении в стек адреса возврата и переходу к первой продукции, относящейся к этому нетерминалу.

Модуль синтаксического анализатора обрабатывает элемент таблицы разбора и определяет следующий элемент для обработки. Поле перехода *jump* дает следующий элемент обработки, если значение поля возврата *return* не окажется равным **true**. В последнем случае адрес следующего элемента берется из стека (что соответствует концу продукции). Если же текущий входной символ отсутствует в списке терминалов и значение поля ошибки окажется **false**, нужно обрабатывать следующий элемент таблицы с тем же текущим входным символом.

Псевдокод алгоритма синтаксического анализа представлен алгоритмом 3.2. Переменная *nextsym* определяет, надо ли считывать новый входной символ до обработки следующего элемента таблицы разбора. Например, *nextsym* = **false**, когда текущий входной символ не является направляющим для какой-либо конкретной продукции и требуется исследовать множество направляющих символов следующей альтернативной продукции. Если символ не содержится в текущем множестве направляющих символов и поле ошибки *error* будет **true**, то выдается сообщение о синтаксической ошибке. Если поле *stack* обрабатываемой *i*-й строки таблицы разбора ($T[i]$) имеет значение **true**, то до перехода к адресу, задаваемому полем перехода *jump*, в стек помещается адрес следующей строки таблицы.

Алгоритм 3.2. Синтаксический анализ с применением специальной *LR*-таблицы разбора
Вход: *LR*-таблица разбора, анализируемая строка
Выход: Сообщения о результатах синтаксического анализа

$i := 1$ // *первая строка таблицы разбора*
$S \Leftarrow 0$ // *0 в стек*
$nextsym :=$ **true**
read(*sym*) // *в sym первый символ входной строки*
while $(sym \neq \perp)$ **and** $(i \neq 0)$ **do**
 with $T[i]$ **do** // *обработка строки i таблицы разбора*

$$\begin{cases} \textbf{if } sym \in terminals \textbf{ then } \begin{cases} nextsym := accept \\ \textbf{if } return \\ \textbf{then } i \Leftarrow S \\ \textbf{else } \begin{cases} \textbf{if } stack \textbf{ then } S \Leftarrow i+1 \\ i := jump \end{cases} \end{cases} \\ \textbf{else if } error \textbf{ then } syntax\_error \textbf{ else } \begin{cases} i := i+1 \\ nextsym := \textbf{false} \end{cases} \\ \textbf{if } nextsym \textbf{ then } read(sym) \end{cases}$$

Такой модуль синтаксического анализа совершенно не зависит от разбираемого языка и может использоваться в целом ряде компиляторов. Он относительно мал, т. к. большая часть объема памяти, занимаемого синтаксическим анализатором, приходится на таблицу разбора, размер которой пропорционален размеру грамматики.

Построим таблицу разбора для $LL(1)$-грамматики со следующими продукциями и их множествами направляющих символов (предполагая наличие продукции $S' \to S\bot$):

$S \to TC$ $\{a, c, \bot\}$
$T \to aTb$ $\{a\}$
$T \to \varepsilon$ $\{b, c, \bot\}$
$C \to cC$ $\{c\}$
$C \to \varepsilon$ $\{\bot\}$

Сначала представим грамматику в виде схемы, в которой все символы помечены номерами строк таблицы разбора:

$$\begin{array}{l} \overset{1}{S} \to \overset{2}{T}\,\overset{3}{C} \\ \overset{4}{T} \to \overset{6}{a}\,\overset{7}{T}\,\overset{8}{b} \\ \overset{5}{T} \to \overset{9}{\varepsilon} \\ \overset{10}{C} \to \overset{12}{c}\,\overset{13}{C} \\ \overset{11}{C} \to \overset{14}{\varepsilon} \end{array}$$

На основании этой схемы легко строится таблица разбора в соответствии с рассмотренными выше правилами заполнения ее полей (рис. 3.4).

	terminals	*jump*	*accept*	*stack*	*return*	*error*
1	{*a*, *c*, ⊥}	2	**false**	**false**	**false**	**true**
2	{*a*, *b*, *c*, ⊥}	4	**false**	**true**	**false**	**true**
3	{*c*, ⊥}	10	**false**	**false**	**false**	**true**
4	{*a*}	6	**false**	**false**	**false**	**false**
5	{*b*, *c*, ⊥}	9	**false**	**false**	**false**	**true**
6	{*a*}	7	**true**	**false**	**false**	**true**
7	{*a*, *b*, *c*, ⊥}	4	**false**	**true**	**false**	**true**
8	{*b*}	0	**true**	**false**	**true**	**true**
9	{*b*, *c*, ⊥}	0	**false**	**false**	**true**	**true**
10	{*c*}	12	**fase**	**false**	**false**	**false**
11	{⊥}	14	**false**	**false**	**false**	**true**
12	{*c*}	13	**true**	**false**	**false**	**true**
13	{*c*, ⊥}	10	**false**	**false**	**false**	**true**
14	{⊥}	0	**false**	**false**	**true**	**true**

Рис. 3.4. *LL*(1)-таблица разбора

Можно сократить число элементов в таблице разбора. Заметим, что имеет место взаимно однозначное соответствие между нулевым значением поля *jump* и значением **true** поля *return*. Следовательно, вместо проверки поля *return* для определения, следует ли брать номер следующей строки из стека, достаточно проверить поле *jump* на равенство нулю. Это позволяет исключить из таблицы разбора поле (столбец) *return*.

Рассмотрим разбор строки *aabbc*⊥, которая выводится в соответствии со следующей левосторонней схемой

$S' \Rightarrow S\bot \Rightarrow TC\bot \Rightarrow aTbC\bot \Rightarrow aaTbbC\bot \Rightarrow aabbC\bot \Rightarrow aabbcC\bot \Rightarrow aabbc\bot$.

Процесс разбора показан на рис. 3.5. Содержимое стека представляется строкой, в которой самый левый символ находится в вершине стека, символ ⊥ показывает дно стека, столбец *i* есть номер применяемой строки таблицы разбора.

Вводимая строка	*i*	Содержимое стека	Комментарии
aabbc⊥	1	0⊥	проверка *a*, перейти к строке 2
aabbc⊥	2	0⊥	проверка *a*, в стек поместить 2 + 1 = 3, перейти к строке 4
aabbc⊥	4	3,0⊥	проверка *a*, перейти к строке 6
aabbc⊥	6	3,0⊥	*a* принимается, перейти к строке 7
abbc⊥	7	3,0⊥	проверка *a*, в стек поместить 7 + 1 = 8, перейти к строке 4
abbc⊥	4	8,3,0⊥	проверка *a*, перейти к строке 6
abbc⊥	6	8,3,0⊥	*a* принимается, перейти к строке 7
bbc⊥	7	8,3,0⊥	проверка *a*, в стек поместить 7 + 1 = 8, перейти к строке 4
bbc⊥	4	8,8,3,0⊥	*b* не совпадает с *a*, *error* = **false**, перейти к строке 4 + 1 = 5
bbc⊥	5	8,8,3,0⊥	проверка *b*, перейти к строке 9
bbc⊥	9	8,8,3,0⊥	проверка *b*, взять из стека 8, перейти к строке 8

Вводимая строка	*i*	Содержимое стека	Комментарии
bbc⊥	8	8,3,0⊥	*b* принимается, взять из стека 8, перейти к строке 8
bc⊥	8	3,0⊥	*b* принимается, взять из стека 3, перейти к строке 3
c⊥	3	0⊥	проверка *c*, перейти к строке 10
c⊥	10	0⊥	проверка *c*, перейти к строке 12
c⊥	12	0⊥	*c* принимается, перейти к строке 13
⊥	13	0⊥	проверка *c*, перейти к строке 10
⊥	10	0⊥	⊥ не совпадает с *c*, *error* = **false**, перейти к строке 10 + 1 = 11
⊥	11	0⊥	проверка ⊥, перейти к строке 14
⊥	14	0⊥	проверка '⊥', взять из стека 0, перейти к строке 0
⊥	0	⊥	стек пуст, разбор успешно завершен

Рис. 3.5. Процесс разбора строки *aabbc*⊥

В процессе разбора некоторые терминалы проверяются несколько раз. Такой неоднократной проверки можно избежать, если отложить обнаружение некоторых синтаксических ошибок на более поздний этап (поздний в смысле шагов разбора, но не считанного текста). Тогда можно сократить число строк в таблице разбора.

С целью экономии памяти можно упаковывать элементы таблицы разбора в машинные слова, но это, очевидно, замедлит работу синтаксического анализатора.

Можно написать программу для получения соответствующей таблицы разбора. Модуль синтаксического анализа можно применять многократно для различных компиляторов, так что при наличии соответствующих программных средств можно получить *LL*(1)-анализатор из *LL*(1)-грамматики с минимальными затратами времени.

LL(1)-метод разбора имеет следующие достоинства:

1) никогда не требуется возврат, поскольку это детерминированный метод;

2) время разбора примерно пропорционально длине программы;

3) имеются хорошие диагностические характеристики и возможность исправления ошибок, т. к. синтаксические ошибки распознаются по первому неприемлемому символу, а в таблице разбора есть список возможных символов продолжения;

4) таблица разбора меньше, чем соответствующие таблицы в других методах разбора.