МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Чувашский государственный университет имени И.Н.Ульянова

Л.А.ПАВЛОВ

ВОСХОДЯЩИЙ СИНТАКСИЧЕСКИЙ АНАЛИЗ

Конспект лекций

Чебоксары 2004

УДК 004.4’422(075.8)
П12

Павлов Л.А.

П12 **Восходящий синтаксический анализ:** Конспект лекций/ Чуваш. ун-т. Чебоксары, 2004. 44 с.

Даны основы теории восходящего синтаксического анализа, приведены методы восходящего разбора для грамматик простого предшествования и *LR*(1)-грамматик.

Для студентов III курса факультета информатики и вычислительной техники специальности 230105 «Программное обеспечение вычислительной техники и автоматизированных систем», а также других инженерных специальностей для более глубокого изучения проблем информатики и вычислительной техники.

Утверждено Методическим советом университета

Отв. редактор канд. техн. наук, доцент А.Л. Симаков

УДК 004.4’422(075.8)

Учебное издание

ПАВЛОВ Леонид Александрович

ВОСХОДЯЩИЙ СИНТАКСИЧЕСКИЙ АНАЛИЗ

Конспект лекций

Редактор Т.В. Пенкина

Подписано в печать 2.11.2004. Формат 60×84/16. Бумага газетная. Печать оперативная. Гарнитура Times New Roman. Усл. печ. л. _,__. Уч.-изд. л. _,_. Тираж 300 экз. Заказ № 636.

Чувашский государственный университет
Типография университета
428015 Чебоксары, Московский просп., 15

ФОРМАЛЬНЫЕ ГРАММАТИКИ

Язык – это множество строк (предложений, цепочек), представляющих собой последовательность символов, каждый из которых принадлежит некоторому конечному *алфавиту* (*словарю языка*). Каждая строка языка формируется из словаря в соответствии с заданными правилами. Совокупность таких правил формирования называется *грамматикой* языка и определяет его структуру.

Строка есть конечная последовательность символов, каждый из которых принадлежит некоторому конечному алфавиту V; при этом символы в строке могут повторяться. Если строка содержит m символов, то говорят, что она имеет длину m. Строка длины 0, т.е. не содержащая ни одного символа, называется пустой и обозначается ε.

Пусть V – некоторый алфавит. Тогда через V^* (рефлексивно-транзитивное замыкание V) обозначается множество всех строк (включая пустую строку), составленных из символов, входящих в V, т. е. это множество определенных над алфавитом V строк. Через V^+ обозначается множество всех строк, исключая пустую строку, т. е. $V^* = V^+ \cup \{\varepsilon\}$.

Формально грамматика определяется как $G = (V_T, V_N, P, S)$, где V_T – конечное множество *терминальных* символов (*терминалов*); V_N – конечное множество *нетерминальных* символов (*нетерминалов*), $V_T \cap V_N = \varnothing$; P – конечное множество *продукций* вида $\alpha \to \beta$, где α – левая часть продукции – строка такая, что $\alpha \in (V_T \cup V_N)^+$, а β – правая часть – строка такая, что $\beta \in (V_T \cup V_N)^*$; $S \in V_N$ – *начальный символ* грамматики.

Пусть $\gamma_1\alpha\gamma_2 \in (V_T \cup V_N)^+$ – строка терминальных и нетерминальных символов длиной ≥ 1. Если $\alpha \to \beta$ – продукция из P, то подстрока α в строке может быть заменена строкой β, и в результате получится $\gamma_1\beta\gamma_2$ (обозначается $\gamma_1\alpha\gamma_2 \Rightarrow \gamma_1\beta\gamma_2$), при этом говорят, что строка $\gamma_1\alpha\gamma_2$ *генерирует* (*порождает*) строку $\gamma_1\beta\gamma_2$, или строка $\gamma_1\beta\gamma_2$ *выводится* из строки $\gamma_1\alpha\gamma_2$.

Если $\alpha_1, \alpha_2, ..., \alpha_n \in (V_T \cup V_N)^*$ и $\alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow \alpha_{n-1} \Rightarrow \alpha_n$ ($n \geq 1$), то обычно пишут сокращенно $\alpha_1 \overset{+}{\Rightarrow} \alpha_n$ и при этом говорят, что строка α_n выводится из строки α_1 за один или более ша-

гов. Аналогично $\alpha_1 \overset{*}{\Rightarrow} \alpha_n$ означает, что строку α_n можно вывести из строки α_1 с помощью нуля или более применений правил грамматики.

Если строка $\alpha \in (V_T \cup V_N)^*$ такая, что $S \overset{*}{\Rightarrow} \alpha$, то строку α называют *сентенциальной формой* грамматики *G*. *Сентенцией* грамматики *G* называют сентенциальную форму, представляющую собой строку, состоящую только из терминалов, которая может быть выведена из начального символа *S*. Тогда множество всех сентенций грамматики *G* называется языком, порожденным грамматикой *G*, и обозначается *L*(*G*).

Таким образом, $L(G) = \{x \in V_T^* \mid S \overset{*}{\Rightarrow} x\}$. Если две различные грамматики *G* и *G*' порождают один и тот же язык, т.е. *L*(*G*) = *L*(*G*'), то грамматики *G* и *G*' *эквивалентны*.

Если в грамматике все левые части продукций состоят из одного нетерминала, т. е. имеют вид $A \to \beta$, где $A \in V_N$, $\beta \in (V_T \cup V_N)^*$, то ее называют *контекстно-свободной*. Любой язык, порожденный контекстно-свободной грамматикой, называется контекстно-свободным языком.

В контекстно-свободной грамматике любая строка $x \in L(G)$ может быть выведена из начального символа *S*. Общепринятым методом представления такого вывода является *дерево вывода* (*дерево грамматического разбора*, *синтаксическое дерево*). Наибольшее распространение получили *левосторонняя* схема вывода, если при выводе строки $x \in L(G)$ раскрывается всегда самый левый нетерминал сентенциальной формы, и *правосторонняя* схема вывода, когда раскрывается самый правый нетерминал сентенциальной формы.

Синтаксическое дерево строится по порождению

$$S \Rightarrow A_1A_2...A_n \Rightarrow ... \Rightarrow T_1T_2...T_k,$$

где $A_i \in (V_T \cup V_N)$, $i = 1, 2, \ldots, n$; $T_j \in V_T$, $j = 1, 2, \ldots, k$, следующим образом. Из корня дерева, помеченного начальным нетерминалом *S*, отходит *n* ветвей по числу символов в строке, непосредственно порожденной начальным нетерминалом. Каждая из *n* ветвей заканчивается вершиной, помеченной символом A_i. Вершины метятся слева направо в порядке возрастания номера индекса. Если в процессе порождения к нетерминалу A_i применена

продукция $A_i \to B_1B_2...B_t$, то вершина A_i становится корнем поддерева, из которого выходит t ветвей, каждая из которых заканчивается вершиной, помеченной символом B_i, $i = 1, 2, \ldots, t$. Такое построение производится для всех вершин, помеченных нетерминальными символами. Построение дерева заканчивается, когда все листья оказываются помеченными терминалами $T_1, T_2, ..., T_k$. Совокупность этих меток при просмотре вершин слева направо образует терминальную строку (сентенцию) $T_1T_2...T_k$. Очевидно, что в полностью построенном синтаксическом дереве листья соответствуют терминалам, а внутренние вершины – нетерминалам.

Пример построения синтаксического дерева для грамматики G_1 с продукциями

$S \to AB$,

$A \to aA \mid a$,

$B \to bB \mid b$

при выводе строки *aaabb* представлен на рис. 1. Это дерево соответствует левосторонней схеме вывода

$$S \Rightarrow AB \Rightarrow aAB \Rightarrow aaAB \Rightarrow aaaB \Rightarrow aaabB \Rightarrow aaabb$$

или правосторонней схеме вывода

$$S \Rightarrow AB \Rightarrow AbB \Rightarrow Abb \Rightarrow aAbb \Rightarrow aaAbb \Rightarrow aaabb.$$

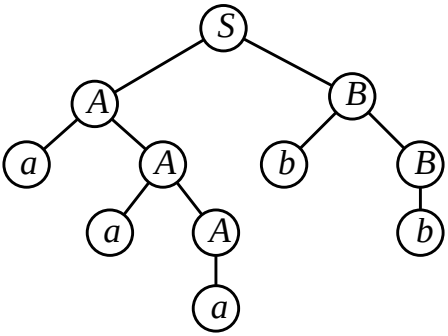

Рис. 1. Дерево вывода строки *aaabb*

Одним из важных этапов компиляции программ является этап синтаксического анализа, в процессе которого распознается синтаксическая правильность программ. Синтаксис большинства языков программирования в своей основной части может быть

определен с помощью контекстно-свободных грамматик, которые являются основой для синтаксического анализа. При решении задачи синтаксического анализа используют, как правило, один из двух основных подходов, которые различаются порядком построения дерева грамматического разбора. Нисходящие методы строят дерево разбора, начиная с корня и завершая листьями, обычно в соответствии с левосторонней схемой вывода и подробно изложены в [6]. В данном конспекте лекций рассматриваются восходящие методы синтаксического анализа языков, синтаксис которых определен с помощью контекстно-свободной грамматики.

ПОСТРОЕНИЕ ДЕРЕВА РАЗБОРА

При восходящем синтаксическом анализе дерево разбора для входной строки строится снизу вверх, начиная с листьев и завершая корнем дерева. Этот процесс можно рассматривать как свертку входной строки к начальному символу грамматики. Входная строка анализируется слева направо в поисках подстроки, которая может быть свернута, т. е. ищется правая часть продукции грамматики, совпадающая с подстрокой. Если такая продукция найдена, то соответствующая подстрока заменяется нетерминалом левой части продукции. В результате такой замены может быть получена сентенциальная форма грамматики, и процесс повторяется до тех пор, пока входная строка не преобразуется в начальный символ грамматики (корень дерева разбора).

Действие замены подстроки нетерминалом левой части продукции называется *сверткой* или *приведением*, а свертываемая подстрока – *основой правосторонней сентенциальной формы*. Таким образом, основа – это подстрока, совпадающая с правой частью продукции, свертка которой в левую часть продукции представляет собой один шаг правосторонней схемы вывода в обратном направлении. Формально основой правосторонней сентенциальной формы $\alpha\beta w$ ($\alpha, \beta \in (V_T \cup V_N)^*$, $w \in V_T^*$) является подстрока β, если существует продукция $A \to \beta$ такая, что β может быть заменена нетерминалом A для получения предыдущей сентенциальной формы в правосторонней схеме вывода

$$S \overset{*}{\Rightarrow} \alpha A w \Rightarrow \alpha\beta w .$$

Следует обратить внимание на то, что подстрока *w* может состоять только из терминалов. Из определения следует, что не любая подстрока β, соответствующая правой части некоторой продукции $A \to \beta$, является основой, поскольку свертка для этой продукции может привести к строке, которая не может быть в последующем свернута к начальному символу грамматики.

Пример построения дерева разбора снизу вверх для грамматики G_1 с продукциями

$S \to AB$,

$A \to aA \mid a$,

$B \to bB \mid b$

при разборе строки *aabb* представлен на рис. 2. Это дерево соответствует правосторонней схеме вывода

$$S \Rightarrow AB \Rightarrow AbB \Rightarrow Abb \Rightarrow aAbb \Rightarrow aabb\ .$$

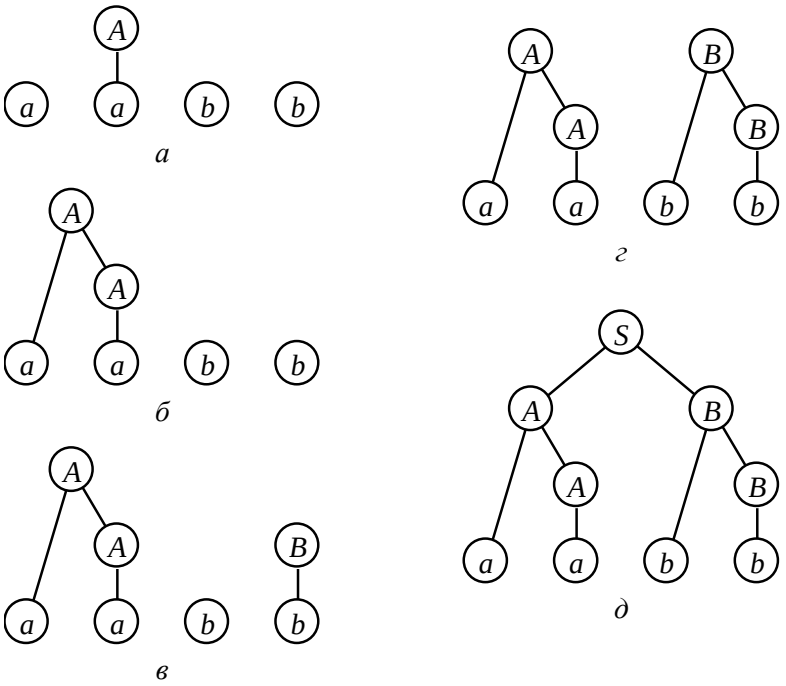

Рис. 2. Процесс построения дерева разбора снизу вверх

В приведенном примере строка *aabb* представляет собой правостороннюю сентенциальную форму, основой которой является второй символ *a* (но не первый!) в соответствии с продукцией $A \to a$. В результате приведения получается правосторонняя сентенциальная форма *aAbb* (рис. 2, *а*), основой которой является подстрока *aA* в соответствии с продукцией $A \to aA$. В результате приведения получается строка *Abb* (рис. 2, *б*). Такая последовательность операций определения основы и приведения продолжается до полного построения дерева разбора: основа сентенциальной формы *Abb* – второй символ *b* (но не первый!) в соответствии с продукцией $B \to b$, результат приведения – строка *AbB* (рис. 2, *в*); основа сентенциальной формы *AbB* – подстрока *bB* в соответствии с продукцией $B \to bB$, результат приведения – строка *AB* (рис. 2, *г*); сентенциальная форма *AB* сама является основой в соответствии с продукцией $S \to AB$, результат приведения – начальный символ грамматики *S*, который является корнем построенного дерева разбора (рис. 2, *д*).

Следует заметить, что если бы в строке *aabb* заменили первый символ *a* на нетерминал *A*, то получили бы строку *Aabb*, которую невозможно свернуть в *S*, т. е. первый символ *a* не является основой, хотя и имеется продукция $A \to a$. По аналогичной причине и первый символ *b* не является основой для строки *Abb*.

Таким образом, при восходящем синтаксическом анализе основной проблемой является поиск и своевременное приведение основы сентенциальной формы на каждом шаге построения дерева разбора.

В синтаксических анализаторах процесс восходящего анализа реализуется с помощью стека, который соответствует части магазинного автомата. При этом анализатор выполняет две основные операции:

1) *сдвиг* (*перенос*), во время которого считывается символ входной строки и помещается в вершину стека (поэтому основа будет находиться в верхней части стека);

2) *приведение* (*свертка*) заключается в замене множества элементов в верхней части стека (основы) на один элемент – нетерминал из левой части соответствующей продукции.

В связи с этим построенные на таких принципах синтаксические анализаторы часто называют анализаторами типа «сдвиг-приведение» или «перенос-свертка».

Процесс анализа строки *aabb* для приведенной выше грамматики G_1 показан в табл. 1. Содержимое стека представляется строкой, в которой самый правый символ находится в вершине стека, символ # показывает дно стека.

Таблица 1

Процесс разбора строки *aabb*

Вводимая строка	Содержимое стека	Выполняемые действия
aabb	#	Сдвиг стека
abb	#*a*	Сдвиг стека
bb	#*aa*	Приведение для продукции $A \to a$
bb	#*aA*	Приведение для продукции $A \to aA$
bb	#*A*	Сдвиг стека
b	#*Ab*	Сдвиг стека
ε	#*Abb*	Приведение для продукции $B \to b$
ε	#*AbB*	Приведение для продукции $B \to bB$
ε	#*AB*	Приведение для продукции $S \to AB$
ε	#*S*	Разбор успешно завершен

Разбор считается успешно завершенным, когда в стеке остается только начальный символ грамматики и полностью считана входная строка.

При такой организации восходящего анализа проблема поиска основы и ее своевременного приведения сводится к проблеме разрешения конфликтов типа сдвиг/приведение (анализатор в конкретной ситуации должен знать, какое из этих действий необходимо выполнить) и приведение/приведение (если возможны разные приведения, необходимо знать, какое из них выполнять и выполнять ли их вообще). Очевидно, что необходимым условием для выполнения приведения является то, чтобы правая часть какой-либо продукции появилась в верхней части стека. Но это условие не является достаточным. Такие ситуации встречались и в рассмотренном выше примере. Поэтому для разрешения конфликтов анализатор должен иметь дополнительную информацию.

ГРАММАТИКИ ПРОСТОГО ПРЕДШЕСТВОВАНИЯ

Одним из наиболее легких подходов к решению проблемы поиска и своевременного приведения основы является реализация восходящего синтаксического анализа для небольшого класса контекстно-свободных грамматик, называемых *грамматиками простого предшествования*. Технология синтаксического анализа для таких грамматик предполагает введение специальных бинарных отношений между каждой парой символов грамматики (как терминалов, так и нетерминалов). Эти отношения управляют выбором основ сентенциальных форм для последующего их приведения.

Основным недостатком этого метода разбора является применимость его лишь в узком классе грамматик простого предшествования.

Отношения предшествования

Пусть $G = (V_T, V_N, P, S)$ – контекстно-свободная грамматика, а строка $\alpha XY\beta$ – правосторонняя сентенциальная форма, где $\alpha, \beta \in (V_T \cup V_N)^*$, $X, Y \in V_T \cup V_N$. В некоторый момент (на одном из этапов процесса последовательных сверток сентенциальной формы) возникает одна из следующих возможных ситуаций:

1. Y – самый левый символ (*заголовок*) основы сентенциальной формы, а X не является последним символом (*окончанием*) другой основы. В этом случае говорят, что X предшествует Y, и записывают в виде $X \lessdot Y$.

2. X и Y входят в одну и ту же основу. В этом случае говорят, что X и Y имеют равное предшествование, и записывают в виде $X \doteq Y$.

3. X – самый правый символ (окончание) основы. При этом Y может являться заголовком другой основы сентенциальной формы. В этом случае говорят, что Y следует за X, и записывают в виде $X \gtrdot Y$.

Отношения $\lessdot$, $\doteq$, $\gtrdot$ называются отношениями простого предшествования. Следует заметить, что, хотя эти отношения похожи на арифметические отношения $<$, $=$, $>$, они имеют совершенно иные свойства. Например, для одной и той же грамма-

тики может быть так, что $X \lessdot Y$ и $X \gtrdot Y$, или для некоторых пар символов не выполняется ни одно из отношений простого предшествования.

Формально эти отношения для символов $X, Y \in V_T \cup V_N$ грамматики определяются следующим образом:

1. $X \doteq Y$, если существует некоторая продукция $A \to \gamma_1 X Y \gamma_2$, $A \in V_N$, $\gamma_1, \gamma_2 \in (V_T \cup V_N)^*$. Это значит, что в правосторонней сентенциальной форме X и Y входят в одну и ту же основу.

2. $X \lessdot Y$, если существует некоторая продукция $A \to \gamma_1 X B \gamma_2$, $A, B \in V_N$, $\gamma_1, \gamma_2 \in (V_T \cup V_N)^*$, такая, что $B \overset{+}{\Rightarrow} Y\delta$, $\delta \in (V_T \cup V_N)^*$. Это значит, что в правосторонней сентенциальной форме основа начинается с символа Y (Y является заголовком), при этом X не должен быть окончанием другой основы.

3. $X \gtrdot Y$, если существует некоторая продукция $A \to \gamma_1 B Z \gamma_2$, $A, B \in V_N$, $Z \in V_T \cup V_N$, $\gamma_1, \gamma_2 \in (V_T \cup V_N)^*$, такая, что $B \overset{+}{\Rightarrow} \delta_1 X$ и $Z \overset{*}{\Rightarrow} Y\delta_2$, $\delta_1, \delta_2 \in (V_T \cup V_N)^*$. Это значит, что в правосторонней сентенциальной форме основа кончается символом X (X является окончанием основы). При этом Y может быть заголовком другой основы, непосредственно следующей за основой с символом X, т. е. две основы могут следовать друг за другом.

Контекстно-свободная грамматика $G = (V_T, V_N, P, S)$ называется грамматикой простого предшествования, если:

1) не содержит ε-продукций;

2) никакие две продукции грамматики не имеют совпадающих правых частей;

3) любые два символа, составляющие элемент множества $(V_T \cup V_N) \times (V_T \cup V_N)$, связаны одним и тем же отношением предшествования.

Отношения предшествования обычно записывают в виде *матрицы предшествований*, строки и столбцы которой соответствуют символам грамматики. На пересечении i-й строки и j-го столбца записывается отношение предшествования между соответствующими символами грамматики. Элементами матрицы являются знаки $\lessdot$, $\doteq$, $\gtrdot$ или «пусто». Последний случай означает, что соответствующие символы не могут стоять рядом ни в одной сентенциальной форме.

Вычисление отношений предшествования

Формальный процесс вычисления отношений предшествования для символов $X, Y \in V_T \cup V_N$ заданной контекстно-свободной грамматики можно представить такой последовательностью действий:

1. Определить для каждого нетерминала X грамматики множество $L(X) = \{Y \mid X \overset{+}{\Rightarrow} Y\alpha\}$, где $\alpha \in (V_T \cup V_N)^*$, т. е. множество символов грамматики (как терминалов, так и нетерминалов), с которых могут начинаться строки, выводимые из нетерминала X. Для этого необходимо построить отношение <LEFT>, определяемое следующим образом: X <LEFT> Y, если в грамматике существует продукция вида $X \to Y\beta$, где $\beta \in (V_T \cup V_N)^*$. Затем вычислить отношение <LEFT>$^+$ как транзитивное замыкание отношения <LEFT>. Тогда $L(X)$ есть множество символов Y, для которых выполняется отношение X <LEFT>$^+$ Y.

2. Вычислить отношение <LEFT>* как рефлексивно-транзитивное замыкание отношения <LEFT>. Очевидно, что отношение <LEFT>* легко вычисляется по отношению <LEFT>$^+$, поскольку имеет место соотношение <LEFT>* = <LEFT>$^+ \cup I$, где I – отношение тождественности. Отношение <LEFT>* понадобится для вычисления отношения $\gtrdot$.

3. Определить для каждого нетерминала X грамматики множество $R(X) = \{Y \mid X \overset{+}{\Rightarrow} \alpha Y\}$, где $\alpha \in (V_T \cup V_N)^*$, т. е. множество символов грамматики, являющихся крайними справа в строках, выводимых из нетерминала X. Для этого необходимо построить отношение <RIGHT>, определяемое следующим образом: Y <RIGHT> X, если в грамматике существует продукция вида $X \to \beta Y$, где $\beta \in (V_T \cup V_N)^*$. Затем вычислить отношение <RIGHT>$^+$ как транзитивное замыкание отношения <RIGHT>. Тогда $R(X)$ есть множество символов Y, для которых выполняется отношение Y <RIGHT>$^+$ X.

4. Построить для всех символов грамматики отношение $\doteq$ по его определению, т. е. $X \doteq Y$, если в грамматике существует продукция вида $A \to \gamma_1 XY\gamma_2$, $A \in V_N$, $\gamma_1, \gamma_2 \in (V_T \cup V_N)^*$.

5. Вычислить отношение $\lessdot$. Из его формального определения следует, что $X \lessdot Y$, если в грамматике имеется продукция вида $A \to \gamma_1 XB\gamma_2$ и $Y \in L(B)$, т. е. $(\lessdot) = (\doteq)\cdot(<\text{LEFT}>^+)$.

6. Вычислить отношение $\gtrdot$. Из его формального определения следует, что $X \gtrdot Y$, если существует продукция вида $A \to \gamma_1 B Y \gamma_2$ и $X \in R(B)$ или имеется продукция вида $A \to \gamma_1 B Z \gamma_2$ и $X \in R(B)$ и $Y \in L(Z)$, т. е. $(\gtrdot) = (\text{<RIGHT>}^+)\cdot(\doteq)\cdot(\text{<LEFT>}^*)$.

7. Построить матрицу предшествования, объединив матрицы отношений $\doteq$, $\lessdot$ и $\gtrdot$ в одну и заменив единицы на соответствующие обозначения ($\doteq$, $\lessdot$, $\gtrdot$) отношений.

Пример процесса вычисления отношений простого предшествования для грамматики G_1 с продукциями

$S \to AB$,

$A \to aA \mid a$,

$B \to bB \mid b$

представлен на рис. 3. Из отношений <LEFT>^+ и <RIGHT>^+ следует, что

$L(S) = \{A, a\}$, $L(A) = \{a\}$, $L(B) = \{b\}$,

$R(S) = \{B, b\}$, $R(A) = \{A, a\}$, $R(B) = \{B, b\}$.

Грамматика G_1 не является грамматикой простого предшествования, поскольку она не удовлетворяет третьему условию, требующему, чтобы любые два символа грамматики были связаны одним и тем же отношением предшествования. В нашем случае одновременно выполняются $A \lessdot b$ и $A \gtrdot b$, а также $A \doteq B$ и $A \gtrdot B$. Действительно $A \lessdot b$, так как имеется продукция $S \to AB$ и $b \in L(B)$, т. е. $B \overset{+}{\Rightarrow} b$, а $A \gtrdot b$, так как существует продукция $S \to AB$, такая, что $A \in R(A)$ и $b \in L(B)$, т. е. $A \overset{+}{\Rightarrow} aA$ и $B \overset{*}{\Rightarrow} b$. Аналогично $A \doteq B$, поскольку имеется продукция $S \to AB$, а $A \gtrdot B$, так как существует продукция $S \to AB$ и $A \in R(A)$, т. е. $A \overset{+}{\Rightarrow} aA$ и $B \overset{*}{\Rightarrow} B$.

Синтаксический анализ

Если грамматика G является грамматикой простого предшествования, то отношения предшествования позволяют легко определить границы основы правосторонней сентенциальной формы. Для этого достаточно проанализировать сентенциальную форму слева направо и найти самую левую пару символов X_j и X_{j+1}, таких, что $X_j \gtrdot X_{j+1}$, т. е. X_j – окончание основы, а отношение $\gtrdot$ отмечает ее правую границу. Затем сентенциальная

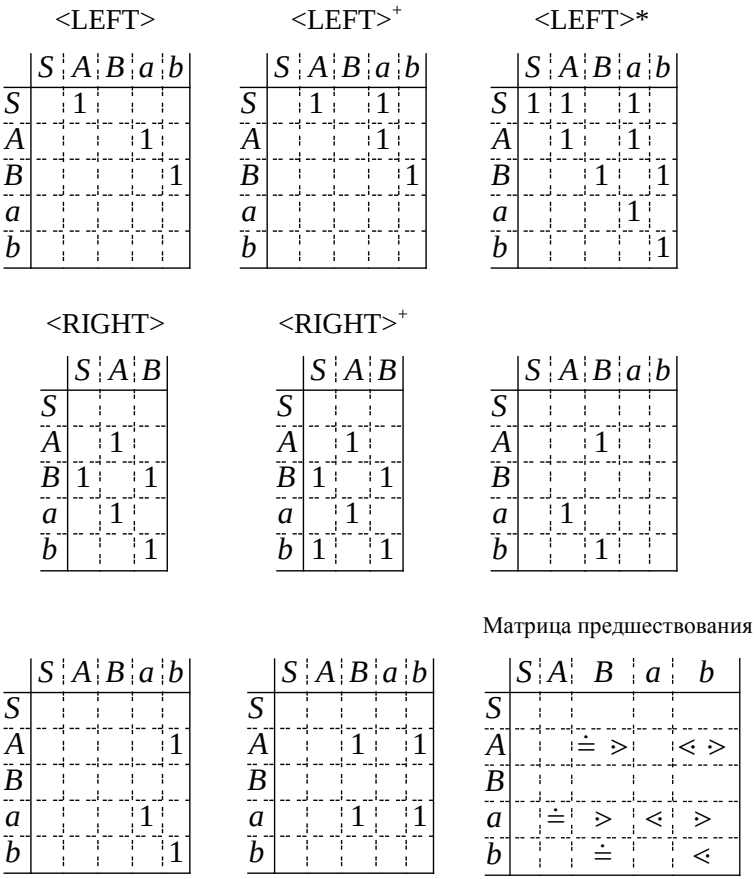

Рис. 3. Процесс вычисления отношений предшествования

форма просматривается справа налево, начиная с символа X_j, до тех пор, пока не будет найдена пара символов X_{i-1} и X_i, таких, что $X_{i-1} \lessdot X_i$, т. е. X_i – заголовок основы, а отношение $\lessdot$ отмечает ее левую границу. Ясно, что между всеми соседними символами внутри основы выполняется отношение $\doteq$. В результате будет найдена основа, имеющая вид $\beta = X_i \ldots X_j$, и должна существовать продукция вида $A \to \beta$, т. е. основа может быть приведена к

нетерминалу A. Например, если сентенциальная форма с отношениями предшествования имеет вид

$$X_1 \lessdot X_2 \doteq X_3 \doteq X_4 \gtrdot X_5,$$

то $X_2X_3X_4$ является ее основой (в грамматике должна быть продукция вида $A \to X_2X_3X_4$), а в результате приведения получится сентенциальная форма X_1AX_5.

Возникает трудность, если заголовок основы является первым символом или окончание основы является последним символом сентенциальной формы. Для преодоления этой трудности вводится специальный символ $\perp$, принадлежащий множеству терминалов, который используется для обозначения начала и конца сентенциальной формы. При этом предполагается, что $\perp \lessdot X$ и $X \gtrdot \perp$ для всех $X \in V_T \cup V_N$, и результатом последовательности приведения входной строки $\perp x \perp$, где $x \in L(G)$, будет строка вида $\perp S \perp$. Другой подход основан на том, что отношения предшествования вычисляются в предположении, что в грамматике имеется продукция вида $S' \to \perp S \perp$. Это позволяет выявлять ошибки на более ранних этапах синтаксического анализа.

Как уже отмечалось, для реализации приведения удобным средством является стек. В начале анализа входной строки в вершину стека записывается символ $\perp$. Затем осуществляется поиск окончания самой левой основы анализируемой строки. Для этого по матрице предшествования проверяются отношения между символом X, находящимся в вершине стека, и очередным входным символом Y. Если условие $X \gtrdot Y$ не выполняется, то очередной входной символ заносится в вершину стека и процесс продолжается до тех пор, пока не будет выполнено условие $X \gtrdot Y$, т. е. не будет найдено окончание самой левой основы. Таким образом, если между символом в вершине стека и очередным входным символом выполняется отношение $\lessdot$ или $\doteq$, то синтаксический анализатор выполняет операцию сдвига стека (окончание основы еще не найдено). После обнаружения окончания основы (выполняется отношение $\gtrdot$) в верхней части стека будет содержаться основа сентенциальной формы.

Затем, чтобы выполнить приведение, производится поиск заголовка основы. Для этого из стека исключаются все символы, имеющие равное предшествование $\doteq$ (начиная с вершины стека), до тех пор, пока между символом в вершине стека и послед-

ним исключенным из стека символом не выполнится отношение $\lessdot$, т. е. последний исключенный из стека символ будет являться заголовком основы. Завершается приведение поиском продукции с соответствующей правой частью и записью в стек нетерминала из левой части этой продукции. Если такая продукция не найдена (т. е. подстрока не является основой), то фиксируется синтаксическая ошибка.

Разбор входной строки успешно завершен, если в результате работы стек будет содержать строку $\perp S$ (содержимое стека представляется строкой, в которой самый правый символ находится в вершине стека), а входной буфер будет содержать символ $\perp$. Если основа не будет найдена или между парами соседних символов не выполняется ни одно из отношений предшествования, то фиксируется синтаксическая ошибка.

В качестве примера рассмотрим грамматику простого предшествования G_2 с продукциями

$S \to DB$,

$D \to A$,

$A \to aA \mid a$,

$B \to bB \mid b$,

матрица предшествования для которой приведена на рис. 4. Отношения предшествования вычислены в предположении, что в грамматике имеется продукция $S' \to \perp S \perp$.

	S	D	A	B	a	b	$\perp$
S							$\doteq$
D				$\doteq$		$\lessdot$	
A				$\gtrdot$		$\gtrdot$	
B							$\gtrdot$
a			$\doteq$	$\gtrdot$	$\lessdot$	$\gtrdot$	
b				$\doteq$		$\lessdot$	$\gtrdot$
$\perp$	$\doteq$	$\lessdot$	$\lessdot$		$\lessdot$		

Рис. 4. Матрица предшествования для грамматики G_2

Рассмотрим работу анализатора на примере разбора строки $aabb\perp$, которая выводится в соответствии со следующей правосторонней схемой (символ $\perp$ начала строки опущен)

$S\bot \Rightarrow DB\bot \Rightarrow DbB\bot \Rightarrow Dbb\bot \Rightarrow Abb\bot \Rightarrow aAbb\bot \Rightarrow aabb\bot$.

Процесс разбора показан в табл. 2. Содержимое стека представлено строкой, в которой для наглядности между каждой парой соседних символов указано отношение предшествования (реально стек содержит только символы грамматики).

Таблица 2

Процесс разбора строки *aabb*⊥

Входной буфер	Содержимое стека	Основа	Выполняемое действие
$aabb\bot$	$\bot$		Сдвиг стека, т. к. $\bot \lessdot a$
$abb\bot$	$\bot \lessdot a$		Сдвиг стека, т. к. $a \lessdot a$
$bb\bot$	$\bot \lessdot a \lessdot a$	a	Приведение для $A \to a$, т. к. $a \gtrdot b$
$bb\bot$	$\bot \lessdot a \doteq A$	aA	Приведение для $A \to aA$, т. к. $A \gtrdot b$
$bb\bot$	$\bot \lessdot A$	A	Приведение для $D \to A$, т. к. $A \gtrdot b$
$bb\bot$	$\bot \lessdot D$		Сдвиг стека, т. к. $D \lessdot b$
$b\bot$	$\bot \lessdot D \lessdot b$		Сдвиг стека, т. к. $b \lessdot b$
$\bot$	$\bot \lessdot D \lessdot b \lessdot b$	b	Приведение для $B \to b$, т. к. $b \gtrdot \bot$
$\bot$	$\bot \lessdot D \lessdot b \doteq B$	bB	Приведение для $B \to bB$, т. к. $B \gtrdot \bot$
$\bot$	$\bot \lessdot D \doteq B$	DB	Приведение для $S \to DB$, т. к. $B \gtrdot \bot$
$\bot$	$\bot \doteq S$		Разбор успешно завершен

Функции предшествования

Если грамматика состоит из n символов, то матрица предшествования имеет размер $n \times n$. Причем многие элементы будут иметь значение «пусто». Можно применить известные методы компактного хранения разреженных массивов, что обычно приводит к дополнительным временным затратам. Поэтому чаще вместо матриц предшествования синтаксические анализаторы используют так называемые *функции предшествования* f и g, отображающие символы грамматики в целые числа. При этом должны выполняться следующие соотношения:

$f(X) = g(Y)$, если $X \doteq Y$,

$f(X) < g(Y)$, если $X \lessdot Y$,

$f(X) > g(Y)$, если $X \gtrdot Y$.

Однако не всякая матрица предшествования может иметь функции предшествования, хотя обычно на практике они существу-

ют. Тогда вместо n^2 элементов достаточно хранить только $2n$ элементов.

Процедура построения функций предшествования по заданной матрице предшествования заключается в следующем.

1. Создать символы f_X и g_X для каждого символа X грамматики, включая и символ $\perp$.

2. Сгруппировать созданные символы на как можно большее число групп таким образом, что если $X \doteq Y$, то f_X и g_Y должны входить в одну группу. Следует заметить, что в одну группу могут попасть символы, не связанные отношением $\doteq$. Например, если $a \doteq b$ и $c \doteq b$, то f_a и f_c должны находиться в одной и той же группе, поскольку оба находятся в той же группе, что и g_b. Если, кроме того, $c \doteq d$, то f_a и g_d находятся в одной группе, даже если не выполняется условие $a \doteq d$.

3. Создать ориентированный граф, вершины которого представляют собой определенные в п. 2 группы. Для всех X и Y, если $X \lessdot Y$, проводится дуга из группы, в которой находится g_Y, в группу с f_X. Если $X \gtrdot Y$, дуга проводится из группы с f_X в группу с g_Y. Таким образом, дуга (путь) от группы с f_X к группе с g_Y означает, что $f(X)$ превосходит $g(Y)$; путь от группы с g_Y к группе с f_X означает, что $g(Y)$ должно превосходить $f(X)$.

4. Если построенный граф имеет циклы, то для заданной таблицы предшествования не существуют функции предшествования. Если циклов нет, то значением $f(X)$ является длина самого длинного пути, начинающегося в группе с f_X; значение $g(X)$ равно длине самого длинного пути из группы с g_X.

В качестве примера определим функции предшествования для матрицы предшествования на рис. 4. Построенный граф показан на рис. 5. Элементы f_S и $g_\perp$ объединены в одну группу, поскольку $S \doteq \perp$, элементы f_a и g_A в одной группе, т. к. $a \doteq A$, и, поскольку $\perp \doteq S$, элементы $f_\perp$ и g_S включены в одну группу. Элементы f_b, f_D и g_B объединены в одну группу, т. к. $b \doteq B$ и $D \doteq B$, несмотря на то, что символы b и D не связаны никаким отношением предшествования.

Этот граф не имеет циклов, следовательно, функции предшествования существуют. Определим самые длинные пути от каждой вершины графа:

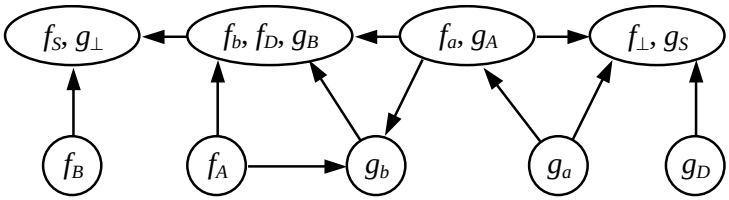

Рис. 5. Граф отношений предшествования

группы $\{f_S, g_\perp\}$ и $\{f_\perp, g_S\}$ не имеют исходящих дуг, т. е. длины путей равны 0,

$\{f_B\} \to \{f_S, g_\perp\}$ длины 1,

$\{f_A\} \to \{g_b\} \to \{f_b, f_D, g_B\} \to \{f_S, g_\perp\}$ длины 3,

$\{f_b, f_D, g_B\} \to \{f_S, g_\perp\}$ длины 1,

$\{g_b\} \to \{f_b, f_D, g_B\} \to \{f_S, g_\perp\}$ длины 2,

$\{f_a, g_A\} \to \{g_b\} \to \{f_b, f_D, g_B\} \to \{f_S, g_\perp\}$ длины 3,

$\{g_a\} \to \{f_a, g_A\} \to \{g_b\} \to \{f_b, f_D, g_B\} \to \{f_S, g_\perp\}$ длины 4,

$\{g_D\} \to \{f_\perp, g_S\}$ длины 1.

Полученные длины путей и будут являться значениями соответствующих функций предшествования. Например, поскольку группы $\{f_S, g_\perp\}$ и $\{f_\perp, g_S\}$ не имеют исходящих дуг (длины путей равны 0), $f(S) = g(\perp) = f(\perp) = g(S) = 0$. Самый длинный путь из $\{f_a, g_A\}$ имеет длину 3, поэтому $f(a) = g(A) = 3$. Аналогично определяются остальные значения функций. Результат построения функций предшествования выглядит следующим образом:

	S	D	A	B	a	b	$\perp$
f	0	1	3	1	3	1	0
g	0	1	3	1	4	2	0

Недостатком использования функций предшествования является то, что теряется информация о несуществующих отношениях, которые могут быть использованы для обнаружения синтаксических ошибок. На практике потеря возможности обнаружения ошибок не считается достаточно серьезной, поскольку в большинстве случаев они могут быть обнаружены при попытке выполнения приведения для необнаруженной основы, но на более поздних этапах разбора.

В качестве примера обнаружения ошибки рассмотрим разбор строки $aba\perp$, не принадлежащей языку, порождаемому грамматикой G_2. При использовании матрицы предшествования (табл. 3) ошибка фиксируется на более раннем этапе разбора по несуществующему отношению для символов b и a. При использовании функций предшествования (табл. 4) ошибка обнаруживается на более позднем этапе при попытке выполнения операции приведения подстроки bD, поскольку в грамматике нет продукции с данной правой частью, т. е. основа не обнаружена.

Таблица 3

Процесс разбора строки $aba\perp$ по матрице предшествования

Входной буфер	Содержимое стека	Основа	Выполняемое действие
$aba\perp$	$\perp$		Сдвиг стека, т. к. $\perp \lessdot a$
$ba\perp$	$\perp \lessdot a$	a	Приведение для $A \to a$, т. к. $a \gtrdot b$
$ba\perp$	$\perp \lessdot A$	A	Приведение для $D \to A$, т. к. $A \gtrdot b$
$ba\perp$	$\perp \lessdot D$		Сдвиг стека, т. к. $D \lessdot b$
$a\perp$	$\perp \lessdot D \lessdot b$		Синтаксическая ошибка, т. к. для символов b и a не существует отношение предшествования

Таблица 4

Процесс разбора строки $aba\perp$ по функциям предшествования

Входной буфер	Содержимое стека	Основа	Выполняемое действие
$aba\perp$	$\perp$		Сдвиг стека, т. к. $f(\perp) < g(a)$
$ba\perp$	$\perp < a$	a	Приведение для $A \to a$, т. к. $f(a) > g(b)$
$ba\perp$	$\perp < A$	A	Приведение для $D \to A$, т. к. $f(A) > g(b)$
$ba\perp$	$\perp < D$		Сдвиг стека, т. к. $f(D) < g(b)$
$a\perp$	$\perp < D < b$		Сдвиг стека, т. к. $f(b) < g(a)$
$\perp$	$\perp < D < b < a$	a	Приведение для $A \to a$, т. к. $f(a) > g(\perp)$
$\perp$	$\perp < D < b < A$	A	Приведение для $D \to A$, т. к. $f(A) > g(\perp)$
$\perp$	$\perp < D < b = D$	bD	Попытка приведения подстроки bD, т. к. $f(D) > g(\perp)$. Синтаксическая ошибка, поскольку нет продукции с правой частью bD

LR(*k*)-ГРАММАТИКИ

Наиболее общим случаем восходящего синтаксического анализа является *LR*-разбор, который применим к широкому классу *LR*(*k*)-грамматик.

LR(*k*)-*грамматикой* называется грамматика, при использовании которой всякая основа сентенциальной формы однозначно определяется (т. е. все конфликты типа сдвиг/приведение и приведение/приведение можно разрешать) на основании уже прочитанного текста (левый контекст) и фиксированного числа предварительно просматриваемых символов (максимум *k*). Аббревиатура *LR* означает «левосторонний ввод – правосторонний вывод», т. е. входная строка анализируется слева направо и используется правосторонняя схема вывода.

Практический интерес представляют случаи $k = 0$ и $k = 1$. Рассмотрение даже двух предварительно просматриваемых символов делает *LR*-метод разбора довольно громоздким. Кроме того, из теории формальных грамматик известно, что любой *LR*(*k*)-язык является также *LR*(1)-языком, т. е. может генерироваться *LR*(1)-грамматикой. Это означает, что для любой *LR*(*k*)-грамматики можно построить эквивалентную *LR*(1)-грамматику. Если *LR*(*k*)-язык удовлетворяет *условию собственности префиксов*, то он может генерироваться *LR*(0)-грамматикой. Это условие означает, что если *x* – строка языка, то никакой ее собственный префикс не принадлежит этому же языку. Поэтому, если используется маркер конца ввода $\perp$, язык удовлетворяет условию собственности префиксов и, следовательно, может генерироваться *LR*(0)-грамматикой.

Таким образом, в отличие от *LL*(*k*)-грамматик, где увеличение значения *k* позволяет представлять больший класс языков, в *LR*(*k*)-грамматиках этого не происходит. На практике обычно используются *LR*(1)-грамматики (или ее более простые подклассы), поскольку многие типичные свойства грамматик языков программирования относятся к *LR*(1)-свойствам, а не к *LR*(0)-свойствам, т. е. использование *LR*(1)-грамматик вместо *LR*(0)-грамматик позволяет избежать затруднений, связанных с преобразованием грамматик.

Основными достоинствами *LR*-разбора являются:

1) метод применим ко всем языкам, которые можно разбирать детерминированно, т. е. имеет универсальный характер и охватывает широкий класс языков и грамматик;

2) относительно редко возникает необходимость в преобразованиях грамматик;

3) имеются хорошие диагностические характеристики и возможность исправления ошибок.

LR-таблицы разбора

Чтобы ссылаться на конкретную позицию в продукции грамматики, вводится понятие конфигурации. В общем случае *LR*(*k*)-*конфигурация* представляет собой пару $[A \to \alpha \bullet \beta, w]$, где $A \in V_N$, $\alpha, \beta \in (V_T \cup V_N)^*$, $w \in V_T^*$. Первый элемент пары – это продукция грамматики, в правой части которой перед одним из символов стоит маркерная точка, а второй элемент w – терминальная строка, которая может следовать за продукцией (т. е. за нетерминалом A). Строка w (ее длина не должна превышать k символов) называется *контекстом* конфигурации или *предпросмотром*. Очевидно, что в *LR*(1)-конфигурации $[A \to \alpha \bullet \beta, a]$ контекст представляет собой единственный терминальный символ $a \in V_T$, а в *LR*(0)-конфигурации $[A \to \alpha \bullet \beta]$ контекст отсутствует (поэтому первый элемент конфигурации часто называют *LR*(0)-конфигурацией).

Первый элемент конфигурации (т. е. *LR*(0)-конфигурация) указывает, какая часть продукции уже исследована в данной точке синтаксическим анализатором. Например, продукция $A \to BCD$ дает четыре варианта расположения маркерной точки (четыре *LR*(0)-конфигурации):

$A \to \bullet BCD$,

$A \to B \bullet CD$,

$A \to BC \bullet D$,

$A \to BCD \bullet$.

Первый вариант определяет, что во входном потоке ожидается строка, порождаемая *BCD*. Второй вариант указывает, что порожденная *B* строка уже исследована, а во входном потоке ожидается строка, порождаемая *CD*. Следует заметить, что продукция вида $A \to \varepsilon$ дает только одну *LR*(0)-конфигурацию $A \to \bullet$.

Что касается контекста конфигурации, то он не играет никакой роли в *LR*(1)-конфигурации вида $[A \to \alpha \bullet \beta, a]$, где $\beta \neq \varepsilon$, а важен для *LR*(1)-конфигурации $[A \to \alpha \bullet, a]$ (эта конфигурация соответствует окончанию продукции $A \to \alpha$), когда выполняется приведение для продукции $A \to \alpha$ только тогда, когда очередным входным символом является терминал *a*. Другими словами, контекст – это символ-следователь, который при разборе строки языка может оказаться предварительно просматриваемым символом в тех случаях, когда выполняется приведение в соответствии с порождающей продукцией.

LR-анализатор использует два стека (стек символов и стек состояний) и специальную *LR*-таблицу разбора. При практической реализации стек символов не требуется, но мы будем его рассматривать для облегчения понимания принципов работы синтаксического анализатора.

LR-таблица разбора представляет собой прямоугольную матрицу, состоящую из столбцов для каждого символа грамматики (терминалов и нетерминалов), включая и маркер конца ввода, и строк, соответствующих каждому состоянию, в котором может находиться анализатор. Состояние обобщает информацию, содержащуюся в стеке ниже него. Комбинация состояния в вершине стека и текущего входного символа используется в качестве индекса таблицы разбора и по ее элементу определяется дальнейшее действие. Таблица разбора включает элементы четырех типов:

1. *Элементы сдвига*. Обычно записываются в виде *Si*, что означает: поместить в стек символов входной символ, поместить в стек состояний состояние *i* и перейти в состояние *i*. Если входной символ является терминалом, принять его (перейти к следующему символу разбираемой входной строки).

2. *Элементы приведения*. Обычно записываются в виде *Rj*, что означает: выполнить приведение для продукции номером *j*, т. е. допустив, что *n* есть число символов в правой части продукции с номером *j*, удалить *n* элементов из стека символов и *n* элементов из стека состояний и перейти к состоянию, находящемуся в вершине стека состояний. Нетерминал в левой части продукции с номером *j* считать входным символом на следующем шаге разбора.

3. *Элементы ошибок*. Это пустые ячейки в таблице разбора и соответствуют синтаксическим ошибкам.

4. *Элементы остановки* (stop), которые указывают на успешное завершение разбора входной строки.

Следует заметить, что все *LR*-анализаторы (при $k \leq 1$) работают одинаково (независимо от того, построен анализатор для *LR*(1)-грамматики или какого-либо ее подкласса), разница между ними заключается только в построении таблиц разбора.

Чтобы найти все состояния, вначале по грамматике строится детерминированный конечный автомат (*характеристический автомат*). Для этого все возможные для заданной грамматики конфигурации по определенным правилам группируются в множества, соответствующие состояниям анализатора. Конфигурации, которые неразличимы для анализатора, объединяются в одно состояние. Правила объединения конфигураций зависят от используемого подкласса *LR*(1)-грамматик и будут изложены при рассмотрении соответствующих подклассов.

Будем считать, что все используемые далее грамматики являются пополненными, т. е. подразумевается, что всегда имеется продукция вида $S' \to S\perp$, где нетерминалы S' и S – начальные символы соответственно пополненной и исходной грамматик, терминал $\perp$ – маркер конца ввода.

LR(0)-грамматики

Рассмотрим самый легкий в реализации, но наименее мощный *LR*(0)-метод построения *LR*-таблицы разбора, основанный на применении *LR*(0)-грамматик. Для таких грамматик основа распознается без исследования предварительно просматриваемого символа, а конфигурация не содержит контекста, т. е. LR(0)-конфигурация $[A \to \alpha\bullet\beta]$ представляет собой только продукцию с маркерной точкой.

Процесс построения характеристического автомата начинается с исходной конфигурации $[S' \to \bullet S\perp]$, где S' – начальный символ пополненной грамматики, которая включается в исходное состояние автомата. Затем выполняется операция замыкания, которая заключается в следующем. Если конфигурация вида $[A \to \alpha\bullet B\beta]$, $B \in V_N$, входит в некоторое состояние, то все конфигурации вида $[B \to \bullet\gamma]$, $\gamma \in (V_T \cup V_N)^*$, включаются в это же

состояние (если их еще там нет). Этот процесс продолжается рекурсивно до тех пор, пока в состояние не будут включены все возможные конфигурации. Таким образом, если в конфигурации маркерная точка стоит перед нетерминалом B, то в текущее состояние включаются все конфигурации, в которых маркерная точка стоит перед первым символом правой части продукции с нетерминалом B в левой части. Если в какой-то из добавленных конфигураций маркерная точка стоит перед нетерминалом, процесс повторяется. Такое выполнение операции замыкания объясняется следующим образом. Наличие $[A \to \alpha \bullet B\beta]$ в некотором состоянии указывает, что в данный момент в процессе синтаксического анализа предполагается, что во входном потоке может встретиться подстрока, выводимая из $B\beta$. Но если имеется продукция $B \to \gamma$, то, естественно, в этот момент может встретиться и строка, выводимая из γ. Поэтому $[B \to \bullet\gamma]$ должна быть включена в это же состояние (конфигурации $[A \to \alpha \bullet B\beta]$ и $[B \to \bullet\gamma]$ являются неразличимыми для анализатора).

Из заданного состояния I, содержащего хотя бы одну конфигурацию, не соответствующую окончанию продукции, строится переход в другое состояние I' по символу грамматики, перед которым стоит маркерная точка. При этом точка перемещается на одну позицию вправо, т. е. будет стоять после этого символа. Состояние I' называется *преемником* состояния I. Новая конфигурация, которая получается при операции образования преемника, называется *базовой* для состояния I'. Таким образом, если конфигурация $[A \to \alpha \bullet X\beta]$, $X \in V_T \cup V_N$, принадлежит состоянию I, то базовой конфигурацией для состояния I' будет конфигурация $[A \to \alpha X \bullet \beta]$ (переход по символу X). Затем для базовой конфигурации выполняется операция замыкания, чтобы включить в новое состояние все остальные неразличимые конфигурации. Следует заметить, что если состояние I содержит несколько конфигураций, в которых точка стоит перед одним и тем же символом X, то в состоянии I' будет множество базовых конфигураций, в которых точка будет стоять после X. Это связано с тем, что характеристический автомат должен быть детерминированным, а в таких автоматах не допускается, чтобы из одного состояния существовал переход по одному и тому же символу в разные состояния.

Процесс последовательного выполнения операций замыкания и образования преемника продолжается до тех пор, пока все возможные конфигурации грамматики не окажутся включенными в какие-либо состояния. Следует иметь в виду, что автомат не должен иметь несколько состояний с одинаковыми множествами конфигураций.

Рассмотрим процесс построения характеристического автомата для грамматики G_3 со следующими продукциями (перед продукцией указан ее порядковый номер):

1) $S \to SbA$,

2) $S \to A$,

3) $A \to a$.

Поскольку мы рассматриваем пополненные грамматики, то имеется еще продукция $S' \to S\bot$. Характеристический автомат для данной грамматики представлен на рис. 6. Для удобства записи в конфигурациях квадратные скобки опущены.

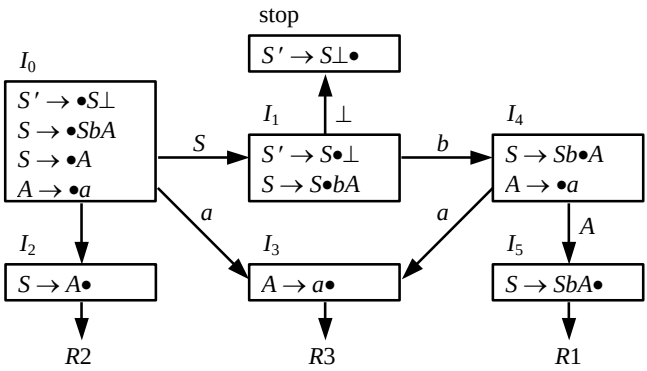

Рис. 6. Характеристический автомат для грамматики G_3

Образуем исходное состояние I_0. Базовой конфигурацией является начальная конфигурация $[S' \to \bullet S\bot]$. Выполним операцию замыкания. Поскольку маркерная точка стоит перед нетерминалом S, в состояние I_0 необходимо включить конфигурации $[S \to \bullet SbA]$ и $[S \to \bullet A]$ для продукций с нетерминалом S в левой части. Среди добавленных конфигураций появился новый нетерминал A, перед которым стоит точка, следовательно, необхо-

димо включить в I_0 конфигурацию $[A \to \bullet a]$. Операция замыкания выполнена и определены все конфигурации состояния I_0.

В состоянии I_0 перед тремя разными символами стоит маркерная точка, т. е. будет три состояния-преемника: I_1 по символу S (базовые конфигурации $[S' \to S\bullet\perp]$ и $[S \to S\bullet bA]$), I_2 по символу A (базовая конфигурация $[S \to A\bullet]$) и I_3 по символу a (базовая конфигурация $[A \to a\bullet]$). Следует заметить, что состояния I_1, I_2 и I_3 исчерпываются базовыми конфигурациями, так как в них нет конфигураций с маркерными точками перед нетерминалами. Конфигурации в состояниях I_2 и I_3 соответствуют окончаниям продукций, поэтому они не имеют преемников. Это означает, что в этих состояниях должно выполняться приведение для соответствующих продукций. На рис. 6 действия приведения обозначены так же, как и элементы приведения в таблице разбора. Например, в состоянии I_2 выполняется приведение для продукции $S \to A$ с номером два (обозначено как $R2$).

У состояния I_1 имеются два состояния-преемника: stop по символу $\perp$ (базовая конфигурация $[S' \to S\perp\bullet]$) и I_4 по символу b (базовая конфигурация $[S \to Sb\bullet A]$). Состояние stop означает, что процесс разбора завершен. Операция замыкания базовой конфигурации в состоянии I_4 добавляет в него конфигурацию $[A \to \bullet a]$. Преемниками I_4 являются состояния I_3 по символу a и I_5 по символу A (базовая конфигурация $[S \to SbA\bullet]$). В состоянии I_5 выполняется приведение $R1$ для продукции $S \to SbA$.

На этом процесс построения характеристического автомата завершен, поскольку все возможные конфигурации заданной грамматики включены в соответствующие состояния. Следует заметить, что одна конфигурация может входить в более чем одно состояние. Например, конфигурация $[S \to \bullet a]$ содержится в состояниях I_0 и I_4.

В полученном автомате никаких конфликтов нет – это особенность $LR(0)$-грамматик. Конфликт приведение/приведение возникает, если в одном состоянии возможно несколько приведений (выходят более одной стрелки приведения). Конфликт сдвиг/приведение возникает, если в состоянии есть приведение и возможен переход в другое состояние (выходят как стрелки приведения, так и стрелки перехода).

Характеристический автомат удобно представлять в табличной форме. Пример такого представления полученного автомата показан в табл. 5. Очевидно, что для состояния stop нет смысла выделять специальную строку в таблице, поскольку это состояние не предполагает никаких действий, а только сигнализирует об успешном завершении разбора.

Таблица 5

Табличное представление характеристического автомата

Состояние	Конфигурации	Символ перехода	Состояние-преемник	Приведение
I_0	$S' \to \bullet S\perp$ $S \to \bullet SbA$	S	I_1	
	$S \to \bullet A$	A	I_2	
	$A \to \bullet a$	a	I_3	
I_1	$S' \to S\bullet\perp$	$\perp$	stop	
	$S \to S\bullet bA$	b	I_4	
I_2	$S \to A\bullet$			$R2$
I_3	$A \to a\bullet$			$R3$
I_4	$S \to Sb\bullet A$	A	I_5	
	$A \to \bullet a$	a	I_3	
I_5	$S \to SbA\bullet$			$R1$

По характеристическому автомату легко строится таблица разбора. Сначала заносятся элементы сдвига. Переход из одного состояния в другое соответствует действию сдвига. Поэтому элементы сдвига вносятся в таблицу разбора на основании информации о состояниях и переходах автомата. Если из состояния I_i существует переход в состояние I_j по символу X, то на пересечении строки i и столбца X записывается элемент сдвига Sj. Если осуществляется переход в состояние stop, то в таблицу разбора вносится элемент останова stop. Следует заметить, что внесение элементов сдвига не зависит от рассматриваемой $LR(1)$-грамматики или ее подкласса. Затем в таблицу разбора вносятся элементы приведения. Поскольку для $LR(0)$-грамматик не учитывается предварительно просматриваемый символ, а в характеристическом автомате нет конфликтов, элементы приведения могут помещаться в каждый столбец в любом состоянии, соответствующем окончанию продукции (табл. 6).

Таблица 6

***LR*(0)-таблица разбора для грамматики G_3**

Номер состояния	S'	S	A	a	b	$\perp$
0		$S1$	$S2$	$S3$		
1					$S4$	stop
2	$R2$	$R2$	$R2$	$R2$	$R2$	$R2$
3	$R3$	$R3$	$R3$	$R3$	$R3$	$R3$
4			$S5$	$S3$		
5	$R1$	$R1$	$R1$	$R1$	$R1$	$R1$

Построенная рассмотренным выше методом таблица разбора называется *LR*(0)-таблицей разбора, а соответствующая грамматика – *LR*(0)-грамматикой. Таким образом, чтобы определить, обладает ли грамматика признаком *LR*(0), необходимо попытаться построить *LR*(0)-таблицу разбора. Если это можно осуществить без возникновения каких-либо конфликтов (элементы приведения можно включить в каждый столбец в любом состоянии, соответствующем окончанию продукции), то грамматика является *LR*(0)-грамматикой. Если же конфликты возникают, грамматика не относится к классу *LR*(0) и следует применить более универсальные методы.

SLR(1)-грамматики

Более мощный подкласс *LR*(1)-грамматик – так называемые *SLR*(1)-грамматики – простые (Simple) *LR*(1)-грамматики. Во многом *SLR*(1)-метод построения *SLR*(1)-таблицы разбора совпадает с *LR*(0)-методом. Как и в *LR*(0)-методе, конфигурации не содержат контекста, т. е. используются *LR*(0)-конфигурации. Отличие заключается в том, что *SLR*(1)-таблица разбора может содержать конфликты определенного вида, которые могут быть разрешены путем просмотра очередного входного символа.

SLR(1)-метод требует вычисления функции

$$FOLLOW(X) = \{a \mid S \overset{*}{\Rightarrow} \alpha X a \beta\},\ a \in V_T,\ \alpha \in V_T^*,\ \beta \in (V_T \cup V_N)^*,$$

для каждого нетерминала X грамматики. Эта функция определяет множество терминалов, которые могут следовать непосредственно за нетерминалом X в какой-либо сентенциальной форме, выводимой из начального нетерминала S. Формальный процесс

вычисления функции *FOLLOW* подробно изложен в [6] при рассмотрении *LL*(1)-грамматик. Значения данной функции используются только для реализации приведения. Если имеется конфигурация $[A \to \alpha\bullet]$, приведение должно выполняться для всех терминалов $a \in FOLLOW(A)$, т. е. в *SLR*(1)-таблицу разбора элементы приведения вносятся только в столбцы, соответствующие возможным символам-следователям.

Рассмотрим процесс построения *SLR*(1)-таблицы разбора для грамматики G_4 с продукциями

1) $S \to AB$,
2) $A \to aA$,
3) $A \to a$,
4) $B \to bB$,
5) $B \to \varepsilon$.

Функция *FOLLOW* для нетерминалов грамматики имеет следующие значения:

$FOLLOW(S) = \{\perp\}$,
$FOLLOW(A) = \{b, \perp\}$,
$FOLLOW(B) = \{\perp\}$.

Характеристический автомат представлен в табл. 7. Значения функции *FOLLOW* включены в столбец «символ перехода» для строк, соответствующих приведению. Это удобно для визуальной проверки состояний на наличие конфликтов.

Следует обратить внимание на то, что для продукции $S' \to S\perp$ построены не все конфигурации, а только $[S' \to \bullet S\perp]$ в состоянии I_0 и $[S' \to S\bullet\perp]$ в состоянии stop (в таблице соответствующая строка не показана). Это связано с тем, что в грамматике не существует более ни одной продукции, в правой части которой содержался бы нетерминал *S*. Поэтому конфигурация $[S' \to S\bullet\perp]$ однозначно устанавливает успешное завершение разбора, и нет смысла строить дополнительное состояние для конфигурации $[S' \to S\perp\bullet]$.

Из характеристического автомата видно, что данная грамматика не является *LR*(0)-грамматикой, поскольку состояния I_1, I_3 и I_5 имеют конфликты типа сдвиг/приведение. Такие состояния называются *неадекватными*. В состояниях I_1 и I_3 необходимо выполнить два сдвига (перехода) по символам *B* и *b* и одно приведение для продукции $B \to \varepsilon$. Чтобы разрешить этот конфликт,

очередной входной символ сравнивается с возможным символом-следователем для нетерминала B, т. е. для нетерминала из левой части продукции, для которой выполняется приведение. Поскольку $FOLLOW(B) = \{\perp\}$, то, если входным символом является символ $\perp$, должно выполняться приведение. Таким образом, в состояниях I_1 и I_3 по символам B и b выполняется сдвиг, а по символу $\perp$ – приведение. Эти символы не совпадают, конфликта нет, неадекватность снимается. В состоянии I_5 сдвиг выполняется по символам A и a, а приведение – по символам b и $\perp$, поскольку $FOLLOW(A) = \{b, \perp\}$, т. е. если очередным входным символом является b или $\perp$, выполняется приведение для продукции $A \to a$. Неадекватность снимается.

Таблица 7

Характеристический автомат для грамматики G_4

Состояние	Конфигурации	Символ перехода	Состояние-преемник	Приведение
I_0	$S' \to \bullet S\perp$	S	stop	
	$S \to \bullet AB$	A	I_1	
	$A \to \bullet aA$	a	I_5	
	$A \to \bullet a$			
I_1	$S \to A\bullet B$	B	I_2	
	$B \to \bullet bB$	b	I_3	
	$B \to \bullet$	$\perp$		$R5$
I_2	$S \to AB\bullet$	$\perp$		$R1$
I_3	$B \to b\bullet B$	B	I_4	
	$B \to \bullet bB$	b	I_3	
	$B \to \bullet$	$\perp$		$R5$
I_4	$B \to bB\bullet$	$\perp$		$R4$
I_5	$A \to a\bullet A$	A	I_6	
	$A \to a\bullet$	$b, \perp$		$R3$
	$A \to \bullet aA$	a	I_5	
	$A \to \bullet a$			
I_6	$A \to aA\bullet$	$b, \perp$		$R2$

В состояниях I_2, I_4 и I_6 выполняются только приведения (конфликтов нет), поэтому при построении таблицы разбора можно было бы включить соответствующие элементы приведения во все столбцы этих состояний, как это делалось в $LR(0)$-методе. Однако $SLR(1)$-метод, рассматривая только воз-

можные символы-следователи, исключает из таблицы несколько элементов приведения. В состоянии I_2 выполняется приведение для продукции $S \to AB$, а $FOLLOW(S) = \{\perp\}$, т. е. за S может следовать только символ $\perp$ и никакой другой. Поэтому для состояния I_2 элемент приведения $R1$ включается только в столбец $\perp$. Аналогично в состоянии I_4 элемент $R4$ (приведение для $B \to bB$) включается в столбец $\perp$, поскольку $FOLLOW(B) = \{\perp\}$, а в состоянии I_6 элемент $R2$ (приведение для $A \to aA$) записывается в столбцы b и $\perp$, так как $FOLLOW(A) = \{b, \perp\}$. Если этого не сделать, то некоторые синтаксические ошибки будут обнаружены на более поздних шагах анализа (но не позднее в смысле считанного текста) (табл. 8).

Таблица 8

$SLR(1)$-таблица разбора для грамматики G_4

Номер состояния	S	A	B	a	b	$\perp$
0	stop	$S1$		$S5$		
1			$S2$		$S3$	$R5$
2						$R1$
3			$S4$		$S3$	$R5$
4						$R4$
5		$S6$		$S5$	$R3$	$R3$
6					$R2$	$R2$

Таким образом, если все конфликты можно разрешить рассмотренным выше способом, т. е. достаточно проанализировать значения функции *FOLLOW* при реализации приведения, грамматика относится к подклассу $SLR(1)$-грамматик. В противном случае необходимо использовать более сложные и мощные методы построения таблиц разбора.

$SLR(1)$-метод часто используется на практике, так как большинство стандартных синтаксических конструкций языков программирования легко выражаются с помощью $SLR(1)$-грамматик. Однако возможны конструкции, с которыми $SLR(1)$-метод не может справиться.

LALR(1)-грамматики

Рассмотренный *SLR*(1)-метод недостаточно мощный, поскольку он не учитывает левый контекст, необходимый в общем случае для принятия решения о действиях синтаксического анализатора для разрешения конфликтов. В ряде случаев левый контекст играет важную роль при решении вопроса о том, можно ли считать заданный символ действительно символом-следователем, хотя в смысле *SLR*(1)-метода он и является возможным символом-следователем (входит в множество, определяемое функцией *FOLLOW*).

Левый контекст учитывается при построении таблицы разбора так называемым LR(1)-методом с предпросмотром (Look Ahead), или *LALR*(1)-методом. Грамматика, для которой все конфликты в таблице разбора разрешаются при использовании данного метода, относится к подклассу *LALR*(1)-грамматик.

В *LALR*(1)-методе, чтобы учитывать левый контекст, состояние характеристического автомата должно содержать дополнительную информацию. Это достигается с помощью контекста конфигурации. Поэтому в этом методе используется *LR*(1)-конфигурация вида $[A \to \alpha\bullet\beta, a]$, где терминал a является контекстом (предпросмотром) конфигурации. Множество таких терминалов всегда является подмножеством множества, определяемого функцией *FOLLOW*. То есть *LR*(1)-конфигурация состоит из *LR*(0)-конфигурации, дополненной контекстом.

Метод построения *LALR*(1)-таблицы разбора, по сути, тот же, что и *SLR*(1)-метод. Отличия связаны с определением контекстов конфигураций и их использованием для разрешения конфликтов. Предпросмотр не играет никакой роли в конфигурации вида $[A \to \alpha\bullet\beta, a]$, где $\beta \neq \varepsilon$, а важен для конфигурации вида $[A \to \alpha\bullet, a]$, когда выполняется приведение для продукции $A \to \alpha$, если очередным входным символом является терминал a.

Предпросмотры определяются при выполнении операции замыкания. Пусть имеется конфигурация $[A \to \alpha\bullet B\beta, a]$, $B \in V_N$. Тогда должна существовать правосторонняя схема вывода

$$S \overset{*}{\Rightarrow} \delta Aax \Rightarrow \delta\alpha B\beta ax, \text{ где } \delta \in (V_T \cup V_N)^*, x \in V_T^*.$$

Предположим, что βax генерирует терминальную строку by, $b \in V_T$, $y \in V_T^*$. Тогда для любой продукции вида $B \to \gamma$, $\gamma \in (V_T \cup V_N)^*$, можно получить схему вывода

$$S \overset{*}{\Rightarrow} \delta\alpha Bby \Rightarrow \delta\alpha\gamma by,$$

т. е. конфигурация $[B \to \bullet\gamma, b]$ должна войти в это же состояние. Необходимо обратить внимание на то, что символ b может быть первым терминалом, выводимым из β; возможно также, что β может генерировать пустую строку ε, в этом случае $b = a$. Следовательно, символ b может быть любым терминалом из множества *FIRST*(βa), где *FIRST* – функция, подробно описанная в [6] при рассмотрении *LL*(1)-грамматик.

Таким образом, операция замыкания заключается в следующем. Если конфигурация вида $[A \to \alpha\bullet B\beta, a]$ входит в некоторое состояние, то все конфигурации вида $[B \to \bullet\gamma, b]$ для всех терминалов b из *FIRST*(βa) включаются в это же состояние. Процесс продолжается рекурсивно до тех пор, пока в состояние не будут включены все возможные конфигурации.

Множество *LR*(0)-конфигураций (т. е. множество первых элементов *LR*(1)-конфигураций), входящих в одно состояние, называется *ядром* состояния. *LALR*(1)-метод не допускает разных состояний с одинаковыми ядрами, даже если контексты конфигураций отличаются. Поэтому если грамматика дает более двух состояний с одним и тем же ядром, то эти состояния должны быть объединены в одно, где различные контексты объединяются в единое множество. В остальных случаях выполнение перехода из состояния в состояние-преемник не влияет на контекст конфигурации.

Рассмотрим *LALR*(1)-метод на примере грамматики G_5 с продукциями

1) $S \to AaBb$,
2) $A \to C$,
3) $A \to db$,
4) $B \to C$,
5) $C \to Ccd$,
6) $C \to d$.

Для сравнения с *SLR*(1)-методом для всех нетерминалов определим значения функции *FOLLOW*:

$FOLLOW(S) = \{\perp\}$,
$FOLLOW(A) = \{a\}$,
$FOLLOW(B) = \{b\}$,
$FOLLOW(C) = \{a, b, c\}$.

Базовой конфигурацией для исходного состояния I_0 характеристического автомата является начальная конфигурация $[S' \to \bullet S\perp, \varepsilon]$, в которой контекстом является пустая строка ε, так как за S' не может следовать никакой символ. Поскольку маркерная точка стоит перед нетерминалом S, а $FIRST(\perp) = \{\perp\}$, необходимо добавить конфигурацию $[S \to \bullet AaBb, \perp]$. Продолжим вычисление замыкания. Поскольку $FIRST(aBb\perp) = \{a\}$, добавляются конфигурации $[A \to \bullet C, a]$ и $[A \to \bullet db, a]$. Маркерная точка стоит перед нетерминалом C, $FIRST(a) = \{a\}$, поэтому добавляются $[C \to \bullet Ccd, a]$ и $[C \to \bullet d, a]$. Опять маркерная точка перед нетерминалом C, $FIRST(cda) = \{c\}$, добавляются конфигурации $[C \to \bullet Ccd, c]$ и $[C \to \bullet d, c]$. Больше никакие новые конфигурации добавить нельзя, операция замыкания для состояния I_0 завершена. Заметим, что в одном состоянии могут быть $LR(1)$-конфигурации с одинаковыми $LR(0)$-конфигурациями, но с разными контекстами, например, $[C \to \bullet d, a]$ и $[C \to \bullet d, c]$. Для удобства записи контексты конфигураций будем представлять как множества, а конфигурации с одинаковыми первыми элементами – как одну конфигурацию (хотя формально это разные конфигурации), контекстом которой является объединение множеств их контекстов, например, конфигурации $[C \to \bullet d, a]$ и $[C \to \bullet d, c]$ будут записываться как $[C \to \bullet d, \{a, c\}]$.

Состояния-преемники определяются так же, как и в ранее описанных методах, т. е. по $LR(0)$-конфигурациям. Характеристический автомат для рассматриваемой грамматики представлен в табл. 9 (символы приведения указаны в столбце «символ перехода»), а $LALR(1)$-таблица разбора – в табл. 10.

Следует обратить внимание на то, что для состояния I_2 в соответствии с конфигурацией $[C \to C\bullet cd, \{a, c\}]$ по символу c образуется преемник с базовой конфигурацией $[C \to Cc\bullet d, \{a, c\}]$, а для состояния I_8 в соответствии с $[C \to C\bullet cd, \{b, c\}]$ – преемник с базовой конфигурацией $[C \to Cc\bullet d, \{b, c\}]$. Ядра этих состояний-преемников совпадают, поэтому, согласно требованию $LALR(1)$-метода, они слиты в одно состояние I_5, а контексты их

конфигураций объединены в единое множество, т. е. получилось состояние с конфигурацией $[C \to Cc\bullet d, \{a, b, c\}]$.

Таблица 9

Характеристический автомат для грамматики G_5

Состояние	Конфигурации	Символ перехода	Состояние-преемник	Приведение
I_0	$S' \to \bullet S\perp, \{\varepsilon\}$	S	stop	
	$S \to \bullet AaBb, \{\perp\}$	A	I_1	
	$A \to \bullet C, \{a\}$ $C \to \bullet Ccd, \{a, c\}$	C	I_2	
	$A \to \bullet db, \{a\}$ $C \to \bullet d, \{a, c\}$	d	I_3	
I_1	$S \to A\bullet aBb, \{\perp\}$	a	I_4	
I_2	$A \to C\bullet, \{a\}$	a		$R2$
	$C \to C\bullet cd, \{a, c\}$	c	I_5	
I_3	$A \to d\bullet b, \{a\}$	b	I_6	
	$C \to d\bullet, \{a, c\}$	a, c		$R6$
I_4	$S \to Aa\bullet Bb, \{\perp\}$	B	I_7	
	$B \to \bullet C, \{b\}$ $C \to \bullet Ccd, \{b, c\}$	C	I_8	
	$C \to \bullet d, \{b, c\}$	d	I_9	
I_5	$C \to Cc\bullet d, \{a, b, c\}$	d	I_{10}	
I_6	$A \to db\bullet, \{a\}$	a		$R3$
I_7	$S \to AaB\bullet b, \{\perp\}$	b	I_{11}	
I_8	$B \to C\bullet, \{b\}$	b		$R4$
	$C \to C\bullet cd, \{b, c\}$	c	I_5	
I_9	$C \to d\bullet, \{b, c\}$	b, c		$R6$
I_{10}	$C \to Ccd\bullet, \{a, b, c\}$	a, b, c		$R5$
I_{11}	$S \to AaBb\bullet, \{\perp\}$	$\perp$		$R1$

Все конфликты для рассматриваемой грамматики успешно разрешены, неадекватных состояний нет, следовательно, она является *LALR*(1)-грамматикой. Эта грамматика не обладает признаком *LR*(0), так как нельзя, например, поместить все элементы приведения $R2$ в каждый столбец для состояния I_2. Не обладает она также и признаком *SLR*(1), так как в противном случае в состоянии I_3, где выполняется приведение для продукции $C \to d$, можно было бы занести элемент приведения $R6$ в столбец, соответствующий терминалу b, поскольку $FOLLOW(C) = \{a, b, c\}$. Однако это привело бы к конфликту с уже имеющимся в этой

позиции элементом сдвига *S*6, т. е. *LALR*(1)-метод устанавливает, что в состоянии I_3 действительными символами-следователями являются только терминалы *a* и *c*, а символ *b* не может быть символом-следователем. По аналогичной причине в состоянии I_9 в столбце, соответствующем терминалу *a*, нет элемента приведения *R*6, поскольку в этом состоянии действительными символами-следователями могут быть только терминалы *b* и *c*.

Таблица 10

***LALR*(1)-таблица разбора для грамматики G_5**

Номер состояния	*S*	*A*	*B*	*C*	*a*	*b*	*c*	*d*	⊥
0	stop	*S*1		*S*2				*S*3	
1					*S*4				
2					*R*2		*S*5		
3					*R*6	*S*6	*R*6		
4			*S*7	*S*8				*S*9	
5								*S*10	
6					*R*3				
7						*S*11			
8						*R*4	*S*5		
9						*R*6	*R*6		
10					*R*5	*R*5	*R*5		
11									*R*1

LALR(1)-метод важен с практической точки зрения, поскольку для стандартных синтаксических конструкций языков программирования там, где не приводит к успеху *SLR*(1)-метод, обычно справляется *LALR*(1)-метод, и достаточно редко приходится применять наиболее общий LR(1)-метод.

LR(1)-грамматики

Наиболее общей и мощной технологией построения таблиц разбора является *LR*(1)-метод. Он, по сути, тот же, что и рассмотренный выше *LALR*(1)-метод. Отличие заключается в том, что в *LR*(1)-методе состояния с идентичными ядрами, но с несовпадающими контекстами конфигураций считаются различными, т. е. их нельзя объединять, как это делалось в *LALR*(1)-методе. Это позволяет разрешать ряд конфликтов, с которыми не справляется *LALR*(1)-метод.

Рассмотрим грамматику G_6 с продукциями

1) $S \to aAd$,
2) $S \to bBd$,
3) $S \to aBe$,
4) $S \to bAe$,
5) $A \to c$,
6) $B \to c$.

Характеристический автомат представлен в табл. 11, а соответствующая $LR(1)$-таблица разбора – в табл. 12.

Таблица 11

Характеристический автомат для грамматики G_6

Состояние	Конфигурации	Символ перехода	Состояние-преемник	Приведение
I_0	$S' \to \bullet S\perp, \{\varepsilon\}$	S	stop	
	$S \to \bullet aAd, \{\perp\}$	a	I_1	
	$S \to \bullet aBe, \{\perp\}$			
	$S \to \bullet bBd, \{\perp\}$	b	I_2	
	$S \to \bullet bAe, \{\perp\}$			
I_1	$S \to a\bullet Ad, \{\perp\}$	A	I_3	
	$S \to a\bullet Be, \{\perp\}$	B	I_4	
	$A \to \bullet c, \{d\}$	c	I_5	
	$B \to \bullet c, \{e\}$			
I_2	$S \to b\bullet Bd, \{\perp\}$	B	I_6	
	$S \to b\bullet Ae, \{\perp\}$	A	I_7	
	$A \to \bullet c, \{e\}$	c	I_8	
	$B \to \bullet c, \{d\}$			
I_3	$S \to aA\bullet d, \{\perp\}$	d	I_9	
I_4	$S \to aB\bullet e, \{\perp\}$	e	I_{10}	
I_5	$A \to c\bullet, \{d\}$	d		$R5$
	$B \to c\bullet, \{e\}$	e		$R6$
I_6	$S \to bB\bullet d, \{\perp\}$	d	I_{11}	
I_7	$S \to bA\bullet e, \{\perp\}$	e	I_{12}	
I_8	$A \to c\bullet, \{e\}$	e		$R5$
	$B \to c\bullet, \{d\}$	d		$R6$
I_9	$S \to aAd\bullet, \{\perp\}$	$\perp$		$R1$
I_{10}	$S \to aBe\bullet, \{\perp\}$	$\perp$		$R3$
I_{11}	$S \to bBd\bullet, \{\perp\}$	$\perp$		$R2$
I_{12}	$S \to bAe\bullet, \{\perp\}$	$\perp$		$R4$

Таблица 12

***LR*(1)-таблица разбора для грамматики G_6**

Номер состояния	*S*	*A*	*B*	*a*	*b*	*c*	*d*	*e*	⊥
0	stop			*S*1	*S*2				
1		*S*3	*S*4			*S*5			
2		*S*7	*S*6			*S*8			
3							*S*9		
4								*S*10	
5							*R*5	*R*6	
6							*S*11		
7								*S*12	
8							*R*6	*R*5	
9									*R*1
10									*R*3
11									*R*2
12									*R*4

Грамматика является *LR*(1)-грамматикой, поскольку все конфликты успешно разрешаются. Но она не относится к подклассу *LALR*(1), так как в соответствии с *LALR*(1)-методом состояния I_5 и I_8, имеющие одно и то же ядро $\{[A \to c\bullet], [B \to c\bullet]\}$, должны объединяться в одно состояние с конфигурациями $[A \to c\bullet, \{d, e\}]$ и $[B \to c\bullet, \{d, e\}]$, что вызывает конфликт приведение/приведение.

Рассмотренная выше грамматика специально усложнена для иллюстрации *LR*(1)-метода. Генерируемый ею язык включает в себя четыре строки и может быть представлен эквивалентной *LR*(0)-грамматикой с продукциями $S \to acd \mid bcd \mid ace \mid bce$.

Если для некоторой грамматики применить все рассмотренные методы построения таблиц разбора, то *LR*(0)-, *SLR*(1)- и *LALR*(1)-таблицы разбора будут иметь одинаковое число состояний, а *LR*(1)-таблица разбора – значительно больше. Таким образом, методы *LR*(0), *SLR*(1) и *LALR*(1) проще и экономичнее общего *LR*(1)-метода. Поскольку большинство конструкций языков программирования легко представляются с помощью *SLR*(1)- или *LALR*(1)-грамматик, лучше сначала опробовать *SLR*(1)-метод. При успешной попытке грамматика будет *SLR*(1)-грамматикой. В противном случае пробуется *LALR*(1)-метод, и если это разрешает все конфликты, то данная грамматика обладает признаком *LALR*(1). Если конфликты остаются, то исполь-

зуется наиболее общий *LR*(1)-метод, и если он не приводит к успеху, то либо необходимо преобразовать грамматику, либо язык не относится к классу *LR*(1)-языков и, следовательно, для него не может существовать *LR*(1)-грамматики.

Очевидно, что классификация *LR*(1)-грамматик включающая, т. е. все грамматики с признаками *LR*(0), *SLR*(1) и *LALR*(1) являются *LR*(1)-грамматиками, все грамматики с признаками *LR*(0) и *SLR*(1) являются *LALR*(1)-грамматиками и т. д.

Рассмотренные выше таблицы разбора обеспечивают быструю выборку и широкие диагностические возможности. Главный их недостаток – для хранения требуется большой объем памяти. Можно использовать известные методы хранения неплотных матриц, но обычно это достигается за счет увеличения времени разбора и более позднего обнаружения синтаксических ошибок.

Синтаксический анализ

Работа синтаксического анализатора типа сдвиг-приведение совершенно не зависит от *LR*-метода, использованного для построения таблицы разбора, и от разбираемого языка (грамматика языка определяет содержимое таблицы, но не влияет на сам алгоритм анализа). Главное требование – *LR*-таблица разбора не должна содержать конфликтов. *LR*-анализатор считывает по одному символы входной строки слева направо и в процессе анализа переходит из одного состояния в другое. При этом используется два стека: стек символов (при практической реализации в нем нет необходимости) и стек состояний. Анализатор находится в состоянии, хранящемся в вершине стека состояний. Следующий шаг синтаксического анализатора зависит от элемента таблицы разбора, позиция которого определяется текущим состоянием (находится в вершине стека состояний) и входным символом. В качестве входного символа может быть текущий символ (терминал) входной строки или нетерминал из левой части продукции, для которой на предыдущем шаге выполнялось приведение. Изначально в стеке состояний содержится исходное состояние I_0 анализатора. Процесс анализа продолжается до тех пор, пока не будет достигнуто успешное завершение (элемент stop в таблице разбора) или не обнаружится синтаксическая

ошибка (пустая ячейка в таблице разбора, позиция которой может дать информацию о характере ошибки).

Рассмотрим работу анализатора для *LALR*(1)-грамматики G_5, иллюстрирующей работу *LALR*(1)-метода (см. с. 34), и соответствующей *LALR*(1)-таблицы разбора (см. табл. 10). Процесс разбора строки *dbadcdb*⊥, которая выводится в соответствии с правосторонней схемой

$S\perp \Rightarrow AaBb\perp \Rightarrow AaCb\perp \Rightarrow AaCcdb\perp \Rightarrow Aadcdb\perp \Rightarrow dbadcdb\perp$,

показан в табл. 13. Содержимое каждого стека представляется строкой, в которой самый правый символ находится в вершине стека, символ # показывает дно стека. В стеке состояний указаны номера состояний.

Таблица 13

Процесс разбора строки *dbadcdb*⊥

Номер шага	Вводимая строка	Стек символов	Стек состояний	Выполняемые действия
1	*dbadcdb*⊥	#	#0	Сдвиг *S*3
2	*badcdb*⊥	#*d*	#0,3	Сдвиг *S*6
3	*adcdb*⊥	#*db*	#0,3,6	Приведение *R*3 для $A \to db$
4	*adcdb*⊥	#	#0	Входной символ *A*. Сдвиг *S*1
5	*adcdb*⊥	#*A*	#0,1	Сдвиг *S*4
6	*dcdb*⊥	#*Aa*	#0,1,4	Сдвиг *S*9
7	*cdb*⊥	#*Aad*	#0,1,4,9	Приведение *R*6 для $C \to d$
8	*cdb*⊥	#*Aa*	#0,1,4	Входной символ *C*. Сдвиг *S*8
9	*cdb*⊥	#*AaC*	#0,1,4,8	Сдвиг *S*5
10	*db*⊥	#*AaCc*	#0,1,4,8,5	Сдвиг *S*10
11	*b*⊥	#*AaCcd*	#0,1,4,8,5,10	Приведение *R*5 для $C \to Ccd$
12	*b*⊥	#*Aa*	#0,1,4	Входной символ *C*. Сдвиг *S*8
13	*b*⊥	#*AaC*	#0,1,4,8	Приведение *R*4 для $B \to C$
14	*b*⊥	#*Aa*	#0,1,4	Входной символ *B*. Сдвиг *S*7
15	*b*⊥	#*AaB*	#0,1,4,7	Сдвиг *S*11
16	⊥	#*AaBb*	#0,1,4,7,11	Приведение *R*1 для $S \to AaBb$
17	⊥	#	#0	Входной символ *S*. stop. Разбор успешно завершен

На шаге 1 входным является символ *d*, анализатор находится в состоянии 0. Поскольку в столбце *d* таблицы разбора для данного состояния содержится элемент *S*3, выполняется сдвиг: в стек символов помещается символ *d*, в стек состояний – состояние 3, и анализатор переходит в состояние 3. На шаге 2 входной

символ – *b*, элемент таблицы – *S*6, следовательно, выполняется сдвиг: в стек символов заносится *b*, в стек состояний – 6, переход в состояние 6. На шаге 3 для состояния 6 и входного символа *a* в таблице указан элемент приведения *R*3, т. е. выполняется приведение по продукции $A \to db$. Поскольку в правой части этой продукции два символа, из стеков удаляются по два верхних элемента, т. е. из стека символов исключается основа. В результате в вершине стека состояний будет 0, т. е. анализатор переходит в состояние 0. Входным символом для следующего шага (шаг 4) будет нетерминал *A*. Последовательность выполнения действий приведения и сдвига продолжается до тех пор, пока на шаге 17 элементом таблицы разбора не станет элемент stop. Это говорит о том, что разбор успешно завершен, входная строка принадлежит языку, т. е. анализатор принимает эту строку.

Пример обнаружения синтаксической ошибки при анализе строки *dbaab*⊥, не принадлежащей языку, показан в табл. 14.

Таблица 14

Процесс разбора строки *dbaab*⊥, не принадлежащей языку

Номер шага	Вводимая строка	Стек символов	Стек состояний	Выполняемые действия
1	*dbaab*⊥	#	#0	Сдвиг *S*3
2	*baab*⊥	#*d*	#0,3	Сдвиг *S*6
3	*aab*⊥	#*db*	#0,3,6	Приведение *R*3 для $A \to db$
4	*aab*⊥	#	#0	Входной символ *A*. Сдвиг *S*1
5	*aab*⊥	#*A*	#0,1	Сдвиг *S*4
6	*ab*⊥	#*Aa*	#0,1,4	Синтаксическая ошибка

На шаге 6 в позиции таблицы разбора, соответствующей состоянию 4 и входному символу *a*, указан элемент ошибки (пустая ячейка), что говорит об обнаружении синтаксической ошибки. К этому моменту принята подстрока *dba*, свернутая в подстроку *Aa*. Характер этой ошибки можно определить, проанализировав строку таблицы разбора, соответствующую состоянию 4. В этой строке в столбцах *B*, *C* и *d* указаны элементы сдвига, а во всех остальных – элементы ошибок. Это говорит о том, что после принятой на настоящий момент подстроки *dba* (свернутой в процессе разбора в подстроку *Aa*) может следовать только подстрока, начинающаяся с *d*, либо подстрока, выводимая из нетерминалов *B* или *C*, т. е. с того же терминала *d*. Таким

образом, для определения характера ошибки достаточно проанализировать только столбцы таблицы разбора, соответствующие терминалам, для состояния, в котором обнаружилась ошибка. Возможными очередными символами входной строки могут быть только терминалы, в столбцах которых не содержатся элементы ошибок.

Сравнение с *LL*-методом разбора

Основными достоинствами как *LL*-, так и *LR*-методов разбора являются их детерминированность и возможность обнаружения синтаксических ошибок, как правило, на самых ранних этапах анализа. Кроме того, оба метода позволяют включать в синтаксис действия для выполнения некоторых аспектов процесса компиляции.

Преимуществом *LR*-метода является то, что он применим к более широкому классу грамматик и языков и обычно не требует преобразований грамматик, которые практически всегда необходимы для *LL*-метода. Однако при наличии хорошего преобразователя грамматик сам факт необходимости преобразований грамматик в действительности не вызывает затруднений. В тех случаях, когда преобразователь грамматик не справляется с какой-либо конструкцией языка программирования, то часто эта конструкция не обладает и признаком *LR*(1). Поэтому для обоих методов возникает необходимость в ручном преобразовании грамматик. Это снижает значимость преимущества *LR*-метода перед *LL*-методом в части преобразования грамматик.

В отношении размеров таблиц разбора *LL*-метод существенно превосходит *LR*-метод. Однако использование методов оптимизации *LR*-таблиц разбора делает их размер примерно того же порядка, что и *LL*-таблицы разбора, но обычно это достигается за счет некоторого увеличения времени разбора.

Сравнение максимального, минимального и среднего времени разбора показывает, что *LL*-метод более эффективен, чем LR-метод.

Список рекомендуемой литературы

1. Ахо А., Сети Р., Ульман Д. Компиляторы: принципы, технологии и инструменты. М.: Вильямс, 2001. 768 с.

2. Ахо А., Ульман Д. Теория синтаксического анализа, перевода и компиляции: В 2 т. М.: Мир, 1978. Т1. 612 с.; Т2. 487 с.

3. Гордеев А.В., Молчанов А.Ю. Системное программное обеспечение. СПб.: Питер, 2002. 736 с.

4. Компаниец Р.И. Системное программирование. Основы построения трансляторов. М.: КОРОНАпринт, 2000. 286 с.

5. Льюис Ф., Розенкранц Д., Стирнз Р. Теоретические основы проектирования компиляторов. М.: Мир, 1979. 654 с.

6. Павлов Л.А. Нисходящий синтаксический анализ: Конспект лекций / Чуваш. ун-т. Чебоксары, 2003. 48 с.

7. Рейуорд-Смит В. Теория формальных языков. Вводный курс. М.: Радио и связь, 1988. 128 с.

8. Хантер Р. Проектирование и конструирование компиляторов. М.: Финансы и статистика, 1984. 232 с.

ОГЛАВЛЕНИЕ