В. Дж. Рейуорд-Смит

ТЕОРИЯ ФОРМАЛЬНЫХ ЯЗЫКОВ

ВВОДНЫЙ КУРС

ТЕОРИЯ ФОРМАЛЬНЫХ ЯЗЫКОВ

ВВОДНЫЙ КУРС

COMPUTER SCIENCE TEXTS

A First Course in Formal Language Theory

V. J. RAYWARD-SMITH
MA, PhD
Senior Lecturer in Computing
University of East Anglia
Norwich NR4 7TJ, UK

BLACKWELL SCIENTIFIC PUBLICATIONS
OXFORD · LONDON · EDINBURGH
BOSTON · PALO ALTO · MELBOURNE

В. Дж. Рейуорд-Смит

ТЕОРИЯ ФОРМАЛЬНЫХ ЯЗЫКОВ

ВВОДНЫЙ КУРС

Перевод с английского
Б. А. Кузьмина
под редакцией
И. Г. Шестакова

Москва «Радио и связь» 1988

ББК 32.97
Р 96
УДК 519.768.2

Рейуорд-Смит В. Дж.

Р 96 Теория формальных языков. Вводный курс. Пер. с англ. – М.: Радио и связь, 1988. – 128 с.: ил.

ISBN 5-256-00159-0.

В книге автора из Великобритании изложены основы теории формальных языков. Использован математический аппарат теории множеств, теории графов и математической логики. Все сведения, необходимые для понимания рассмотренных в книге вопросов, приведены в соответствующих главах. Удачно подобранные упражнения в конце каждой главы не только поясняют, но и дополняют основной материал книги.

Для разработчиков программного обеспечения ЭВМ.

$\text{Р}\frac{2405000000\text{-}027}{046\,(01)\text{-}88}$ 148-88 ББК 32.97

Редакция переводной литературы

Производственное издание

В. ДЖ. РЕЙУОРД-СМИТ

ТЕОРИЯ ФОРМАЛЬНЫХ ЯЗЫКОВ ВВОДНЫЙ КУРС

Заведующая редакцией О. В. Толкачева
Редактор М. Г. Коробочкина
Обложка художника А. Б. Николаевского
Художественный редактор Т. В. Бусарова
Технический редактор И. Л. Ткаченко
Корректор Т. Л. Кускова

ИБ № 1618

Подписано в печать 8.12.87 Формат 60x88/16 Бумага офс. № 2 Гарнитура "Пресс-роман" Печать офсетная Усл. печ. л. 7,84 Усл. кр.-отт. 8,21 Уч.-изд. л. 8,23
Тираж 30 000 экз. Изд. № 22110 Зак. № 507 Цена 55 к.
Издательство "Радио и связь". 101000 Москва, Почтамт, а/я 693

Московская типография № 4 Союзполиграфпрома при Государственном комитете СССР по делам издательств, полиграфии и книжной торговли. 129041 Москва, Б. Переяславская ул., д. 46

ISBN 5-256-00159-0 (рус.)
ISBN 0-632-01176-9 (англ.)

Давно уже прошли те времена, когда прежде чем написать программу надо было понять и запомнить не один десяток машинных инструкций. Современный программист формулирует свои задачи на языках программирования высокого уровня и использует язык ассемблера лишь в исключительных случаях. Причин для этого много и некоторые из них очевидны: резко повышается эффективность программирования и, в основном, становится безразличным, на какой конкретно ЭВМ будет выполняться данная программа. В 80-е годы резко усилилась роль мини-, микро- и персональных компьютеров.

В настоящее время рядом с суперЭВМ в самых различных областях человеческой деятельности успешно используются персональные компьютеры, при необходимости легко связываемые средствами вычислительных сетей. Контингент и подготовка специалистов, пользующихся программными средствами, поставляемыми с персональными компьютерами, становятся все разнообразнее. Разнообразнее становятся и решаемые с помощью этих средств задачи, что приводит к появлению различных специализированных, как правило диалоговых, программных средств и, следовательно, новых языков программирования высокого и сверхвысокого уровней, таких, как Форт, Пролог и Шелл. Наряду с этим не следует забывать и о таких языках программирования, как Си, Ада и Паскаль, популярность которых продолжает год от года расти. Таким образом, основополагающее значение алгоритмических языков в программировании сохраняется.

Как известно, алгоритмические языки становятся доступными программисту лишь после создания трансляторов с этих языков, и с решением этой проблемы самым тесным образом связана данная книга.

Дело в том, что теоретические основы методов проектирования алгоритмических языков, а также конструирования трансляторов, особенно в части грамматического и семантического анализа текста программы, укладываются в рамки теории формальных языков.

Основными достоинствами книги Рейуорда-Смита являются строгость изложения и использование математического аппарата теории множеств, теории графов и математической логики. Изложенный в книге материал представляет собой учебный курс, и это оправдывает принятую автором методологию изложения, общую для математических дисциплин. Это обстоятельство немаловажно для студентов, а также профессиональных программистов, решивших посвятить себя проектированию языков программирования и трансляторов с них. Успешному восприятию материала способствует также изобилие примеров, иллюстрирующих каждую теорему и сопровождающих каждое сколько-нибудь сложное логическое построение. Упражнения, приведенные после каждой главы книги, дают читателю возможность не просто проследить практическое использование изложенных в главе положений, но и развить их наиболее наглядным и естественным способом.

Перевод книги Рейуорда-Смита на русский язык является очень своевременным и актуальным, так как появление персональных компьютеров, оснащенных графическими средствами, безусловно приведет к появлению большого числа разработок в области автоматизации проектирования, информационно-справочных систем и локалвных вычислительных сетей. Однако грамотное и эффективное решение этих проблем невозможно без понимания основ теории формальных языков.

Посвящается моей семье

Книга, которая не рождает мыслей, *по моему убеждению* – просто забавная безделушка.

Т. Л. Пикок "Замок чудес"

ПРЕДИСЛОВИЕ

Для большинства новичков теория формальных языков является весьма трудным препятствием в основном из-за ее громоздких и тяжеловесных формулировок. Обойти это препятствие невозможно – каждый уважающий себя программист обязан хорошо понимать процесс компиляции программ, что, в свою очередь, приводит к необходимости знакомства с теорией формальных языков. Предлагаемая книга адресована студентам первых двух курсов, и поэтому помимо основных материалов по теории формальных языков содержит достаточно много дополнительных сведений из высшей математики и теории множеств, чтобы обеспечить необходимую подготовку к любому курсу по компиляции программ. По этой же причине основное внимание сконцентрировано на регулярных и контекстно-независимых грамматиках.

Большинство учебников по этой теме, в отличие от данной книги, адресовано студентам третьего курса и аспирантам. Они содержат более полное и подробное изложение принципов теории формальных языков, однако весьма трудны для понимания. Я же постарался изложить материал как можно проще, не нарушая, тем не менее, строгости формулировок. Так, все приведенные в первых главах теоремы доказаны очень подробно и аккуратно, чтобы дать возможность читателю поскорее привыкнуть к стилю изложения и понять схему большинства доказательств. С каждой новой главой, по мере того как читатель приобретает навыки в построении доказательств, последние становятся все более лаконичными и формальными. Заключительные главы книги содержат, как правило, лишь наброски доказательств, а главное внимание уделяется практическому использованию результатов. Взаимная зависимость содержания глав проиллюстрирована на рис. П.1.

Материал, изложенный в гл. 1–3 и 5, соответствует, в основном, первому году трехгодичного курса по теории вычислений, а вся книга – двум годам указанного курса. Хотя, если речь идет об университете, то план такого курса может быть несколько изменен или же весь материал книги может быть положен в основу 21-часового спецкурса по теории формальных языков.

Основным мотивом, побудившим меня взяться за написание этой книги,

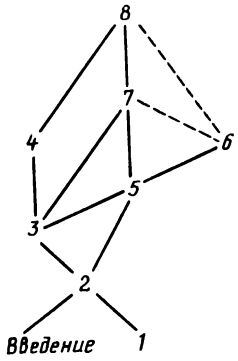


Рис. П.1

стали многочисленные просьбы группы студентов младших курсов Калифорнийского университета в Санта-Барбаре, где я читал курс по теории формальных языков группе студентов, состоявшей, в основном, из новичков. Я благодарен им за их энтузиазм, а кроме того, хочу выразить признательность сотрудникам университета за помощь и поддержку при подготовке книги. Особо хочу поблагодарить доктора Дж. П. Макьюина за полезные советы и Т. Поттер за великолепную перепечатку рукописи книги.

В. Дж. Рейуорд-Смит

ВВЕДЕНИЕ

Приступая к изучению какого-либо языка, мы обнаруживаем, что неплохо подготовлены к этому. Действительно, все мы знаем хотя бы один язык — язык, на котором мы общаемся друг с другом. Каким бы языком мы ни пользовались в жизни (английским, французским или испанским), мы продолжаем совершенствоваться в нем до конца своих дней. Кроме того, большинство из нас, владея одним или несколькими естественными языками, знакомятся и даже становятся знатоками одного или нескольких языков программирования, например таких, как Алгол 68, Паскаль, Фортран или Бейсик. Перечисленные языки используются для написания программ, т. е. являются средством общения с ЭВМ. Никто, по-видимому, не станет спорить с тем, что ЭВМ "не догадывается" о смысле и назначении написанных человеком программ, следовательно, роль языка при "общении" с ЭВМ особенно высока. Если при разговоре друг с другом мы способны однозначно понять сказанное, когда в нем содержатся неполные или неправильно построенные фразы, то при "общении" с ЭВМ подобные вольности недопустимы. Текст программ должен отвечать набору жестких правил, а малейшее отклонение от этих правил делает программы непонятными с точки зрения ЭВМ.

В идеальном случае любое предложение на любом языке должно быть корректно одновременно семантически (т. е. иметь разумный смысл) и синтаксически (т. е. иметь верную грамматическую структуру). В разговорном языке фразы иногда могут быть неверны грамматически, но всегда осмысленны. В программировании каждое предложение должно быть, в первую очередь, корректно синтаксически, так как только в этом случае оно может стать осмысленным.

Для того чтобы понять семантику предложения, необходимо понять его смысл. Например, следующие предложения семантически эквивалентны:

The man hits the dog.
The dog is hit by the man.
L'homme frappe le chien.

Для того чтобы понять синтаксис предложения, необходимо рассмотреть его грамматическую структуру. Проделаем синтаксический разбор, например, предложения The man hits the dog (человек бьет собаку):

The man	hits	the dog
⟨ИМЯ СУЩЕСТВИТЕЛЬНОЕ⟩	⟨ГЛАГОЛ⟩	⟨ИМЯ СУЩЕСТВИТЕЛЬНОЕ⟩

Можно утверждать, что любое предложение с такой грамматической структурой будет в английском языке синтаксически корректным[1].

Можно описать семейство подобных простых предложений с помощью форм Бэкуса–Наура:

```
⟨ПРОСТОЕ ПРЕДЛОЖЕНИЕ⟩ : := ⟨ПОДФРАЗА СУЩЕСТВИТЕЛЬНОГО⟩
                            ⟨ГЛАГОЛ⟩ ⟨ПОДФРАЗА
                            СУЩЕСТВИТЕЛЬНОГО⟩
⟨ПОДФРАЗА СУЩЕСТВИТЕЛЬНОГО⟩ : := ⟨АРТИКЛЬ⟩ ⟨ИМЯ СУЩЕСТВИТЕЛЬНОЕ⟩
⟨ИМЯ СУЩЕСТВИТЕЛЬНОЕ⟩ : := CAR
⟨ИМЯ СУЩЕСТВИТЕЛЬНОЕ⟩ : := MAN
⟨ИМЯ СУЩЕСТВИТЕЛЬНОЕ⟩ : := DOG
⟨АРТИКЛЬ⟩ : := THE
⟨АРТИКЛЬ⟩ : := A
⟨ГЛАГОЛ⟩ : := HITS
⟨ГЛАГОЛ⟩ : := EATS
```

Правила, определяющие грамматическую структуру простого предложения, записаны здесь с помощью форм Бэкуса–Наура (БНФ); подобная нотация широко используется для описания синтаксиса языков программирования. Подробно формы Бэкуса–Наура будут обсуждены в гл.2.

В рассмотренном примере ⟨ПРОСТОЕ ПРЕДЛОЖЕНИЕ⟩ было определено как упорядоченная последовательность:
⟨ПОДФРАЗА СУЩЕСТВИТЕЛЬНОГО⟩ ⟨ГЛАГОЛ⟩ ⟨ПОДФРАЗА СУЩЕСТВИТЕЛЬНОГО⟩.

В свою очередь, ⟨ПОДФРАЗА СУЩЕСТВИТЕЛЬНОГО⟩ была определена как упорядоченная последовательность:

⟨АРТИКЛЬ⟩ ⟨ИМЯ СУЩЕСТВИТЕЛЬНОЕ⟩.

Выбирая в качестве
первого ⟨АРТИКЛЯ⟩ "THE",
первого ⟨ИМЕНИ СУЩЕСТВИТЕЛЬНОГО⟩ "MAN",
⟨ГЛАГОЛА⟩ "HITS",
второго ⟨АРТИКЛЯ⟩ "THE",
второго ⟨ИМЕНИ СУЩЕСТВИТЕЛЬНОГО⟩ "DOG",
получим, согласно определению, простое предложение вида

THE MAN HITS THE DOG.

Подобная взаимосвязь упомянутых конструкций может быть описана с помощью так называемого дерева вывода (рис. В.1).

---

[1] До конца этой главы читателю будет предложено слегка углубиться в грамматику английского языка. – *Прим. ред.*

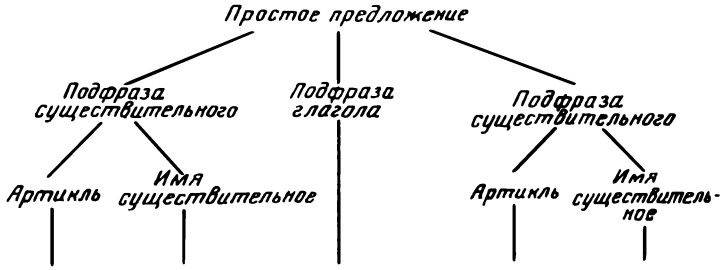


Рис. В.1

Используя определение простого предложения, можно построить множество различных простых предложений и, хотя все они будут синтаксически корректны, осмысленными могут оказаться далеко не все. Для иллюстрации сказанного рассмотрим простое предложение вида

THE CAR EATS THE MAN[1]

Оно вполне корректно синтаксически, но абсолютно лишено смысла.

Рассмотрим теперь предложение, не являющееся производным от простого предложения:

THEY ARE FLYING PLANES.

Первый вариант грамматического разбора этого предложения выглядит следующим образом:

THEY	ARE	FLYING	PLANES
МЕСТОИМЕНИЕ	ГЛАГОЛ	ИМЯ ПРИЛАГАТЕЛЬНОЕ	ИМЯ СУЩЕСТВИТЕЛЬНОЕ
		ПОДФРАЗА СУЩЕСТВИТЕЛЬНОГО	

При таком грамматическом разборе предложения смысл его заключается в следующем: "они" – это самолеты, летающие в небе в данный момент.

Во втором варианте грамматического разбора этого предложения:

THEY	ARE FLYING	PLANES
МЕСТОИМЕНИЕ	ГЛАГОЛ	ИМЯ СУЩЕСТВИТЕЛЬНОЕ

смысл его заключается в том, что "они" – это люди, управляющие в данный момент полетами самолетов.

Как вы, наверное, убедились, докопаться до смысла рассмотренного предложения можно только с помощью подробного грамматического разбора. И хотя рассмотренная в примере ситуация весьма надуманна и не претендует на какие-либо обобщения в отношении естественного языка, пример этот иллюстрирует весьма важное в отношении формальных языков обстоятельство: грамматический разбор исходного текста программы, написанной

[1] "Автомобиль ест человека". – *Прим. перев.*

на одном из языков программирования как основной шаг на пути ее семантической интерпретации, является решающим этапом компиляции программы. По этой причине программирование без знания синтаксиса используемого языка программирования невозможно.

Значительное место в книге занимает исследование свойств различных грамматик, а также их использование при создании языков программирования. Хорошее понимание теории грамматик необходимо всякому системному программисту, так как природа исследуемых им объектов непременно потребует от него математических усилий. Формально данная книга адресована начинающим системным программистам, однако теория формальных языков часто рассматривается как ветвь математики, а исследования в этой области невозможны без привлечения мощного математического аппарата. Желающим более подробно познакомиться с исследованиями на эту тему, я настоятельно рекомендую обратиться к книге Hopcroft J. E., Ullman J. D. Introduction to Automata Theory, Languages and Computation.— Addison-Wesley, Reading, Mass.

ГЛАВА 1

МАТЕМАТИЧЕСКОЕ ПРЕДИСЛОВИЕ

"Математика – это не только строгая истина, – это еще и высшая красота, красота строгая и холодная как статуя".

Лорд Бернард Рассел
"Урок математики"

Первая глава книги представляет собой краткий обзор основных понятий и положений высшей математики, без знания которых невозможно понять остальной материал. Если читатель не знаком с этими положениями, то рекомендуем тщательно изучить эту главу и убедиться в том, что все понятно. Наоборот, если изложенный здесь материал уже известен и понятен читателю, можно пролистать эту главу с целью познакомиться с используемой в этой книге нотацией.

МНОЖЕСТВА

Множество – это набор нескольких несовпадающих объектов. Каждый из этих объектов называется *элементом множества.* Если множество содержит конечное число элементов, то оно может быть определено перечислением элементов. Например, если D – множество дней недели, то

$D =$ {Понедельник, Вторник, Среда, Четверг, Пятница, Суббота, Воскресенье}

– его определение.

В данном случае элементы множества D разделены запятыми, а весь набор элементов заключен в фигурные скобки. Вообще говоря, порядок перечисления элементов множества D не важен, т. е. запись

$D =$ {Среда, Четверг, Понедельник, Воскресенье, Суббота, Вторник, Пятница}

эквивалентна предыдущему определению множества D.

Если x – элемент множества A, то это записывают следующим образом $x \in A$; при этом говорят, что элемент x принадлежит множеству A, или просто, что x принадлежит A. Если x не является элементом множества A, то это записывают следующим образом: $x \notin A$; при этом говорят, что

элемент x не принадлежит множеству A, или просто, что x не принадлежит A Например,

Понедельник $\in D$,
Рыба $\notin D$.

Если множество A содержит бесконечное число элементов, то говорят, что множество A бесконечно. Определить это множество простым перечислением его элементов невозможно. Для определения бесконечного множества используется характеристическое свойство, которым должен обладать объект, чтобы стать элементом множества. В случае рассмотренного выше множества D определить его элементы удобно с помощью следующего свойства:

$\{x$ – день недели$\}$.

Тогда определение множества D можно записать следующим образом:

$D = \{x | x$ – день недели$\}$,

и в этом случае говорят, что "D – множество элементов x, таких, что x – день недели".

Другой пример:

$P = \{x | x$ – простое число$\}$,

где P – бесконечное множество, содержащее целые числа.

Если множество определено подобным образом, то исследуя, является ли некоторый объект элементом этого множества, необходимо быть чрезвычайно внимательным. Например, если множество X определено как

$X = \{x | x > 2\}$,

то природа множества может быть описана лишь после рассмотрения всех возможных значений x, так как если рассмотреть только случай "x – целое положительное число", то очевидно, что число $2.1 \notin X$, в случае же, когда "x – любое число", число $2.1 \in X$. В книге будут рассматриваться множества, все элементы которых будут принадлежать одному из аналогичных классов чисел, например: целые числа, действительные числа, целые положительные числа или действительные положительные числа.

Особую роль в теории множеств играет пустое множество. *Пустое множество* – это множество, которое не содержит ни одного элемента, оно обозначается $\emptyset$ или $\{\ \}$.

Говорят, что множество A называется подмножеством множества B, если каждый элемент множества A является элементом множества B, и это записывается следующим образом: $A \subset B$. Если хотя бы один элемент множества A не является элементом множества B, то множество A не является подмножеством множества B и это записывается следующим образом: $A \not\subset B$. Например,

$\{1, 2, 4\} \subset \{1, 2, 3, 4, 5\}$,

но

$\{2, 4, 6\} \not\subset \{1, 2, 3, 4, 5\}$.

Заметим, что $\emptyset \subset A$ для любого множества A.

Говорят, что множество A совпадает с множеством B, если каждый элемент множества A является элементом множества B и, наоборот, каждый элемент множества B является элементом множества A, и это записывается следующим образом: $A = B$, что эквивалентно одновременному выполнению двух условий $A \subset B$ и $B \subset A$. Например,

$$\{1,2,3,4\} = \{2,1,4,3\},$$

но

$$\{1,2,3,4\} \neq \{2,1,3,5\}.$$

Таким образом, множества A и B совпадают тогда и только тогда[1], когда они содержат одни и те же элементы.

Основными операциями над множествами являются унарная операция[2] – дополнение $(')$, бинарные операции[3] – объединение $(\cup)$, пересечение $(\cap)$ и вычитание $(\setminus)$. Предположим, что A и B – множества элементов из некоторого класса объектов U; тогда эти операции определяются следующим образом:

$A' = \{x | x \notin A\}$ содержит все те элементы U, которые не являются элементами множества A;

$A \cup B = \{x | x \in A$ или $x \in B\}$ содержит все те элементы U, каждый из которых является либо элементом множества A, либо элементом множества B;

$A \cap B = \{x | x \in A$ и $x \in B\}$ содержит все те элементы U, каждый из которых является одновременно элементом множества A и элементом множества B;

$A \setminus B = \{x | x \in A$, но $x \notin B\}$ содержит все те элементы U, каждый из которых является элементом множества A, но не является элементом множества B.

Например, пусть $U = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, $A = \{0, 1, 3, 5\}$ и $B = \{2, 3, 5\}$; тогда

$$A' = \{2,4,6,7,8,9\},$$
$$A \cup B = \{0,1,2,3,5\},$$
$$A \cap B = \{3,5\},$$
$$A \setminus B = \{0,1\},$$
$$B \setminus A = \{2\}.$$

Отметим, что операции объединения и пересечения множеств ассоциативны, например,

$$(A \cup B) \cup C = A \cup (B \cup C);$$
$$(A \cap B) \cap C = A \cap (B \cap C).$$

Операция вычитания множеств неассоциативна.

---

[1] В качестве стандартного обозначения формулировки "тогда и только тогда, когда" используется знак $\Leftrightarrow$.

[2] Унарная операция – операция, совершаемая над одним операндом.

[3] Бинарная операция – операция, совершаемая над двумя операндами.

Аналогично операции объединения и пересечения множеств коммутативны, а операция вычитания множеств некоммутативна:

$A \cup B = B \cup A;$
$A \cap B = B \cap A.$

ТЕОРЕМА 1.1. СВОЙСТВА МНОЖЕСТВ

Для любых множеств A, B и C, рассматриваемых над классом объектов U, выполняются следующие законы:

1. Ассоциативный закон	$(A \cup B) \cup C = A \cup (B \cup C),$ $(A \cap B) \cap C = A \cap (B \cap C).$
2. Коммутативный закон	$A \cup B = B \cup A,$ $A \cap B = B \cap A.$
3. Закон о дополнении	$A \cup A' = U,$ $A \cap A' = \emptyset.$
4. Закон идемпотенции	$A \cup A = A,$ $A \cap A = A.$
5. Закон эквивалентности	$A \cup \emptyset = A,$ $A \cap U = A.$
6. Закон существования пустого множества	$A \cup U = U,$ $A \cap \emptyset = \emptyset.$
7. Закон инволюции	$(A')' = A.$
8. Закон Моргана	$(A \cup B)' = A' \cap B'$ $(A \cap B)' = A' \cup B'$
9. Дистрибутивный закон	$A \cup (B \cap C) = (A \cup B) \cap (A \cup C),$ $A \cap (B \cup C) = (A \cap B) \cup (A \cap C).$

Поскольку операция объединения множеств ассоциативна, а объединение двух любых множеств над классом объектов U есть множество над классом объектов U, для любого n и для любой конечной последовательности множеств $A_1, A_2, \ldots, A_n$ над классом объектов U можно записать

$A_1 \cup A_2 \cup \ldots \cup A_n$, или $\bigcup_{i=1}^{n} A_i$.

Аналогично для операции пересечения множеств можно записать

$A_1 \cap A_2 \cap \ldots \cap A_n$ или $\bigcap_{i=1}^{n} A_i$.

Два множества не пересекаются, если они не имеют общих элементов, т. е. $A \cap B = \emptyset$.

МОЩНОСТЬ И СЧЕТНОСТЬ МНОЖЕСТВА

Если число элементов множества A конечно, такое множество называют *конечным множеством*, в противном случае его называют *бесконечным множеством*.

Мощностью конечного множества A называют число элементов этого множества и обозначают $\#(A)$. Например, если D – множество дней неде-

ли, то $\#(D) = 7$. Множество, имеющее мощность равную 1, называют *синглетоном.*

Рассмотрим теперь бесконечные множества и остановимся на способе перечисления элементов такого множества. Пусть N – множество положительных целых чисел; тогда способ перечисления его элементов очевиден: 1, 2, 3, 4, . . . , и можно с уверенностью говорить об "i-м положительном целом числе". Однако не очевидно (и к тому же неверно), что аналогично можно перечислить элементы любого бесконечного множества.

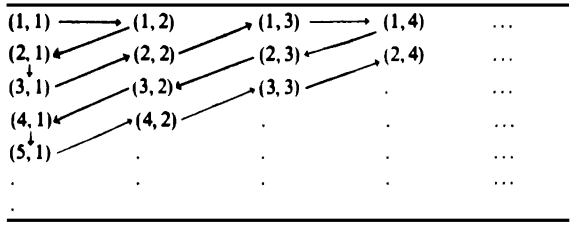
Пусть Z – множество целых чисел. Можно ли перечислить элементы множества Z так, как это было сделано для элементов множества положительных целых чисел? Да, можно – достаточно перечислить пары целых чисел, имеющих противоположные знаки, но одинаковые модули, таким образом:

$$0, +1, -1, +2, -2, +3, -3, \ldots$$

Любой элемент $z \in Z$ обязательно попадет в эту последовательность.

Пары положительных целых чисел (n_1, n_2), где $n_1 \in N$ и $n_2 \in N$, могут быть перечислены с помощью алгоритма, приведенного в табл. 1.1.

Таблица 1.1



Множество, элементы которого могут быть перечислены, т. е. упорядочены с помощью некоторого алгоритма таким образом, что любому элементу может быть поставлен в соответствие его номер, называют *счетным множеством*[1]. Очевидно, что всякое конечное множество счетно. В качестве несчетного множества рассмотрим множество действительных чисел из полуинтервала $[0,1)$, т. е. множество действительных чисел, не меньших нуля, но меньших единицы. Доказательство несчетности этого множества основано на методе диагонализации или методе Кантора. Предположим, что рассматриваемое множество счетно; тогда очевидно, что его элементы могут быть перечислены. Запишем элементы полученной упорядоченной последовательности в десятичной форме:

1-е действительное число $0.a_{11}a_{12}a_{13}\ldots$

2-е действительное число $0.a_{21}a_{22}a_{23}\ldots$

3-е действительное число $0.a_{31}a_{32}a_{33}\ldots$

. . . . . .

. . . . . ,

[1] Или более строго: счетное множество – это множество, для которого существует взаимно однозначное соответствие его элементов элементам натурального ряда. – *Прим. ред.*

где каждое a_{ij} – цифра. Поскольку рассматриваемое множество счетно, любой его элемент должен в конце концов встретиться в этой последовательности. Рассмотрим действительные числа вида $0.a_{11}a_{22}a_{33}\ldots$ и вида $0.b_{11}b_{22}b_{33}\ldots$, где

$$b_{ii}=\begin{cases}a_{ii}+1, \text{ если } a_{ii}<9,\\ 0 \quad\quad , \text{ если } a_{ii}=9.\end{cases}$$

Действительное число вида $0.b_{11}b_{22}b_{33}\ldots$ лежит в полуинтервале $[0,1)$, но не может встретиться в приведенной последовательности действительных чисел, так как для любого целого положительного i i-м знаком числа $0.b_{11}b_{22}b_{33}\ldots$ будет цифра b_{ii}, по определению не совпадающая с соответствующей цифрой a_{ii} соответствующего действительного числа $0.a_{i_1}a_{i_2}a_{i3}$, являющегося элементом построенной нами упорядоченной последовательности. Таким образом, исходное предположение о счетности множества действительных чисел из полуинтервала $[0,1)$ неверно.

ПРОИЗВЕДЕНИЯ МНОЖЕСТВ

Пусть A_1 и A_2 – два множества, *произведением двух множеств* A_1 и A_2 называют такое множество, каждый элемент которого представляет собой совокупность двух объектов (пару), при этом один из объектов пары является элементом множества A_1, а второй – элементом множества A_2; записывается это следующим образом:

$$A_1 \times A_2 = \{(a_1, a_2) | a_1 \in A_1, a_2 \in A_2\}.$$

Рассмотрим простой пример: $A_1=\{0,1\}$, $A_2=\{x,y,z\}$, тогда $A_1 \times A_2 = \{(0,x), (0,y), (0,z), (1,x), (1,y), (1,z)\}$.

Произведение трех или более множеств можно определить аналогично как множество, элементами которого являются соответственно тройки или совокупности из n объектов:

$$A_1 \times A_2 \times A_3 = \{(a_1, a_2, a_3) |\ a_1 \in A_1, a_2 \in A_2,\ a_3 \in A_3\},$$

$$A_1 \times A_2 \times \ldots \times A_n = \{(a_1, a_2, \ldots, a_n) |\ a_1 \in A_1,\ a_2 \in A_2, \ldots,\ a_n \in A_n\}.$$

Для конечных множеств легко доказать следующие утверждения: 1) $\#(A_1 \times A_2) = \#(A_1) \times \#(A_2)$; 2) произведение двух счетных множеств есть множество счетное. Методом индукции по n легко показать, что для любого $n>1$ справедливо утверждение: $\#(A_1 \times A_2 \times \ldots \times A_n) = \#A_1 \times \#A_2 \times \ldots \times \#A_n$ и множество $A_1 \times A_2 \times \ldots \times A_n$ – счетно $\Leftrightarrow$ каждое из множеств $A_1, A_2, \ldots, A_n$ – счетно.

Любое подмножество множества $A_1 \times A_2$ есть *отношение* (обозначим его R), при этом множество A_1 называют *областью определения*, а множество A_2 – *областью значений*[1]. Чаще всего мы будем рассматривать отношения, у

[1] Подобно тому, как словом "произведение" принято обозначать одновременно и операцию умножения, и ее результат, справедливо ожидать, что словом "отношение" будет обозначено как само отношение (например $<$, $>$, $=$, $\neq$, . . .), так и множество удовлетворяющих ему пар. – *Прим. ред.*

которых эти области представляют собой одно и то же множество, скажем, множество A. В таком случае говорят, что отношение $R \subset A \times A$ определено на множестве A. Например, если $A = \{0, 1, 2, 3\}$, тогда множество упорядоченных пар

$$L = \{(0,1),(0,2),(0,3),(1,2),(1,3),(2,3)\}$$

представляет собой отношение, отвечающее общепринятому понятию "строго больше чем", а множество

$$E = \{(0,0),(1,1),(2,2),(3,3)\}$$

представляет собой отношение, отвечающее понятию "равно". Воспользуемся обычной нотацией отношений; тогда

aRb для $(a, b) \in R$,
$a \not R b$ для $(a, b) \notin R$.

Отношение R, определенное на множестве A, называют *рефлексивным*, если aRa для всех $a \in A$. Таким образом, отношение L, определенное на множестве A, не является рефлексивным, поскольку $(0, 0) \notin L$, а отношение E, определенное на множестве A, является рефлексивным.

Отношение R, определенное на множестве A, называют *симметричным*, если

из aRb следует, что bRa.

Таким образом, отношение L не является симметричным, поскольку $(0,1) \in L$, но $(1,0) \notin L$, а отношение E является симметричным.

Отношение R, определенное на множестве A, называют *транзитивным*, если

из aRb и bRc следует, что aRc.

Например, отношения L и E, определенные на множестве A, являются транзитивными.

Если отношение R, определенное на множестве A, является рефлексивным, симметричным и транзитивным, то его называют *отношением эквивалентности*.

Так, отношение E, определенное на множестве A, является отношением эквивалентности.

Если R – отношение эквивалентности, определенное на множестве A, и элемент $a \in A$, то ассоциированное с элементом a множество $\bar{a}$ содержит все те элементы множества A, для которых справедливо отношение эквивалентности R, т. е.

$$\bar{a} = \{b \in A \mid aRb\}.$$

Такое множество $\bar{a}$ называют *классом эквивалентности*.

Таким образом, для отношения эквивалентности E существуют четыре класса эквивалентности: $\bar{0} = \{0\}$, $\bar{1} = \{1\}$, $\bar{2} = \{2\}$, $\bar{3} = \{3\}$, а для отношения эквивалентности V, определенного на множестве A, следующим образом:

$$V = \{(0,0),(0,2),(1,1),(1,3),(2,2),(2,0),(3,1),(3,3)\},$$

существует всего два класса эквивалентности: $\bar{0} = \bar{2} = \{0, 2\}$ и $\bar{1} = \bar{3} = \{1, 3\}$.

ТЕОРЕМА 1.2

Классы эквивалентности, соответствующие отношению эквивалентности R, определенному на множестве A, разбивают множество A на конечное число непустых непересекающихся множеств.

Доказательство

Каждое множество $\overline{a}$ – не пустое. Действительно, $a \in \overline{a}$, поскольку отношение эквивалентности R – рефлексивно, и следовательно, утверждение aRa справедливо. Пусть $\overline{a}$ и $\overline{b}$ – два класса эквивалентности; тогда либо $\overline{a} = \overline{b}$, либо $\overline{a} \cap \overline{b} = \emptyset$. Предположим обратное – существует элемент $c \in \overline{a} \cap \overline{b}$; тогда $c \in \overline{a}$ и $c \in \overline{b}$, т. е. утверждения aRc и bRc справедливы. Поскольку R – отношение эквивалентности, оно симметрично, и из справедливости утверждения bRc следует справедливость утверждения cRb. Так как отношение R – транзитивно, то из справедливости утверждения aRc и cRb следует справедливость утверждения aRb. Следовательно, элемент $b \in \overline{a}$. Очевидно, что для любого элемента $x \in \overline{b}$ справедливо bRx, и поскольку справедливость aRb только что доказана, то из транзитивности R следует справедливость aRx, значит $x \in \overline{a}$. Последнее означает, что $\overline{b} \subset \overline{a}$. Аналогично можно показать, что $\overline{a} \subset \overline{b}$, следовательно $\overline{a} = \overline{b}$.

Продолжим рассмотрение классов эквивалентности. Определим отношение R на множестве натуральных чисел N следующим образом: aRb справедливо $\Leftrightarrow |a-b|$ делится на 5 без остатка. В таком случае множество N разбивается на 5 бесконечных классов эквивалентности:

$$\overline{1} = \overline{6} = \overline{11} = \cdots,$$

$$\overline{2} = \overline{7} = \overline{12} = \cdots,$$

$$\overline{3} = \overline{8} = \overline{13} = \cdots,$$

$$\overline{4} = \overline{9} = \overline{14} = \cdots,$$

$$\overline{5} = \overline{10} = \overline{15} = \cdots.$$

Если R – произвольное отношение, определенное на множестве A, то его *рефлексивным замыканием* называется наименьшее рефлексивное отношение, определенное на множестве A, для которого отношение R является подмножеством. Например, если

$$R = \{(0,1),(1,1),(1,2)\}$$

– отношение, определенное на множестве $A = \{0,1,2\}$, то его рефлексивным замыканием является множество

$$\{(0,0),(0,1),(1,1),(1,2),(2,2)\},$$

симметричным замыканием является множество

$$\{(0,1),(1,0),(1,1),(1,2),(2,2)\}$$

а *транзитивным замыканием* является множество

$$\{(0,1),(0,2),(1,1),(1,2)\}.$$

Заметим, что рефлексивным замыканием отношения $<$, определенного на множестве целых чисел, является отношение $\leqslant$, его симметричным замыка-

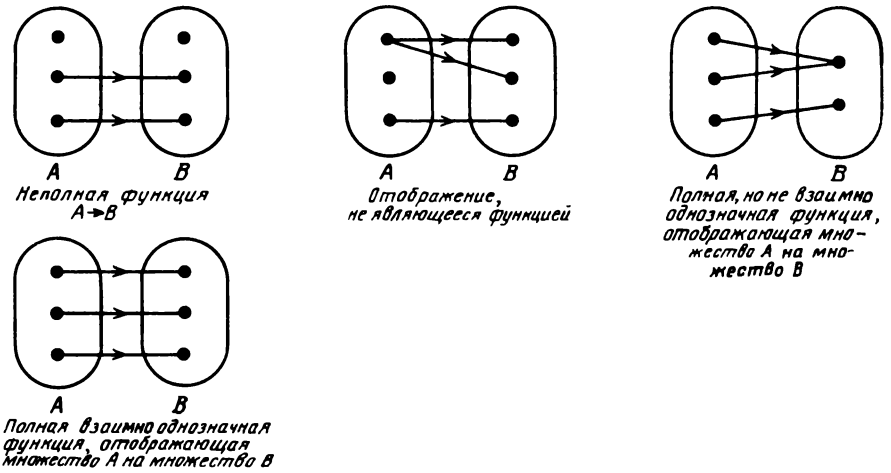


Рис. 1.1

нием является отношение $\neq$, а транзитивным замыканием является само отношение $<$.

Функция $f : A \to B$ есть множество пар элементов (a, b), где $a \in A$, $b \in B$ и $b = f(a)$. Таким образом, функция $f : A \to B$ может быть определена как отношение, определенное на множестве $A \times B$, обладающее следующим свойством: если $(a, b) \in f$ и $(a, c) \in f$, то $b = c$. Такие функции f называют *однозначными.* Если функция f определена не на всех элементах множества A, то ее называют *неполной функцией* (в противном случае ее называют *полной функцией*). Полная функция f называется *функцией "на"*, если для любого элемента $b \in B$ существует элемент $a \in A$ такой, что $f(a) = b$; полная функция f называется *взаимно однозначной*, если из того, что $f(a_1) = f(a_2)$, где $a_1 \in A$, $a_2 \in A$ следует, что $a_1 = a_2$. Рисунок 1.1 иллюстрирует свойства различных функций (направленные отрезки прямых на этом рисунке иллюстрируют соответствие аргумента функции и ее значения).

Если полная функция $f : A \to B$ взаимно однозначно отображает множество A на множество B, то говорят, что между множествами A и B может быть установлено *взаимно однозначное соответствие.*

Счетность множества можно определить теперь следующим образом: множество A является счетным, если оно конечно или существует взаимно однозначное соответствие между множествами A и N, где N — натуральный ряд.

ГРАФЫ И ДЕРЕВЬЯ

Направленным графом $G = (V, E)$ называется совокупность конечного множества узлов и определенного на этом множестве отношения. Направленный граф можно представить графически следующим образом: нарисуем на листе бумаги столько узлов (обозначим их v), сколько элементов

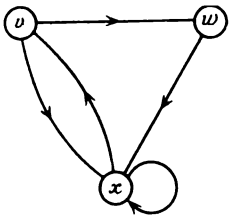


Рис. 1.2

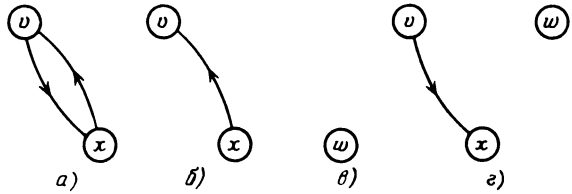


Рис. 1.3

имеет множество V, затем соединим два изображения узлов v и w отрезком прямой, направленным от узла v к узлу w, для любой пары элементов v, w, для которой $(v, w) \in E$. На рис. 1.2 изображен направленный граф $G_1 = (V_1, E_1)$, где $V_1 = \{v, w, x\}$, а $E = \{(v, w), (v, x), (w, x), (x, v), (x, x)\}$.

Если для некоторой пары узлов направленного графа $G = (V, E)$ $(v, w) \in E$, то говорят, что существует *дуга на графе G от узла v до узла w*. Если имеется некоторая последовательность узлов $v = v_0, v_1, \ldots, v_n = w$, где $n \geqslant 0$, направленного графа G такая, что $(v_i, v_{i+1}) \in E$ для $i = 0, 1, \ldots, n-1$, то говорят, что существует *направленный путь по графу G от узла v до узла w* длиной n. Это означает, что либо $v = w$, либо упомянутый путь по графу есть последовательность направленных дуг.

Пусть $G = (V, E)$ – некоторый направленный граф; тогда $G' = (V', E')$ – *направленный подграф* графа G, если $V' \subset V$, $E' \subset E$. На рис. 1.3 представлены четыре подграфа направленного графа G_1, изображенного на рис. 1.2. Два подграфа, не имеющие общих узлов, называют *раздельными подграфами*. На рис. 1.3 пары подграфов *а* и *б* и *в* и *г* – примеры раздельных подграфов.

Важной разновидностью направленного графа является *дерево*, определяемое рекурсивно следующим образом: деревом $T = (V, E)$ называют направленный граф такой, что он либо имеет один-единственный узел, называемый корнем $T = (\{r\}, \varnothing)$, либо представляет собой совокупность корня нескольких раздельных подграфов $T_1, T_2, \ldots, T_k$ $(k \geqslant 1)$ (каждый из которых является, в свою очередь, деревом) и дуг от корня дерева T до узлов, являющихся корнями деревьев $T_1, T_2, \ldots, T_k$. На рис. 1.4 представлена графическая интерпретация дерева; убедитесь в том, что она соответствует приведенному выше определению дерева.

Обычно графическое изображение дерева имеет вид расположенного вверху корня и направленных сверху вниз дуг. Если мы условимся впредь пользоваться только таким графическим представлением дерева, то направленность дуг можно не указывать.

Узел дерева, из которого не исходит ни одной дуги, называют либо *внешним узлом*, либо *граничным узлом*, либо просто *листом*

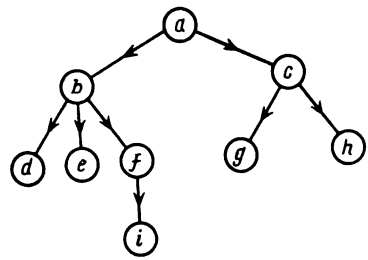


Рис. 1.4

дерева. На рисунке узлы дерева, помеченные буквами d, e, g, h и i, – внешние узлы дерева. Если узел дерева не является внешним узлом, его называют *внутренним узлом* дерева. Вообще говоря, терминология, принятая для описания деревьев, напоминает генеалогическую, например, если два узла v и w соединены дугой, направленной от узла v к узлу w, то узел v принято называть *родительским узлом* узла w. Все узлы дерева, за исключением корневого узла, имеют родительские узлы. Если узел v – родительский узел узла w, то узел w называют *дочерним узлом* узла v. Любой внутренний узел дерева имеет дочерний узел. Если направленный путь по дереву от узла v к узлу w состоит из нескольких направленных дуг, тогда узел v называют *узлом-предком* узла w, а узел w называют *узлом-потомком* узла v.

СТРОКИ

Используя термин *строка*, мы всегда подразумеваем конечную последовательность символов $a_1, a_2, \ldots, a_n$, каждый из которых принадлежит некоторому конечному алфавиту Σ; при этом символы в строке могут повторяться. Например, строка символов из алфавита $\Sigma = \{0, 1\}$ выглядит следующим образом: 0001110. Если строка содержит m символов (подсчитываются все символы строки независимо от их повторений), то о такой строке говорят, что она имеет длину m. Например, строка 001110 имеет длину 6. Пустой строкой называют строку длины 0, т.е. строку, не содержащую ни одного символа; пустую строку принято обозначать буквой ε. Длина строки обозначается $|x|$. В приведенном выше примере $|001110| = 6$, а $|\varepsilon| = 0$.

Математическое описание строки чрезвычайно просто. Действительно, строка – это конечное множество символов, каждый из которых является элементом некоторого заранее определенного произвольного непустого конечного множества символов, называемого *алфавитом*. Например, если строка включает символ "пробел", то это означает, что этот символ является элементом определенного алфавита. Необходимо отметить, что во избежание недоразумений в тексте символ "пробел" часто заменяют символом Λ; таким образом, строка вида $00\Lambda11\Lambda01$ является фактически условной записью строки вида 00 11 01.

Пусть Σ – некоторый алфавит; обозначим через Σ^* множество определенных над этим алфавитом строк. Пусть $\Sigma = \{0, 1\}$, тогда

$$\Sigma^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, 001, 010, \ldots\}.$$

Множество Σ^* представляет собой, очевидно, бесконечное счетное множество элементов. Используя обычную нотацию, можно записать, что $x \in \Sigma^*$, если x – элемент множества Σ^*, и $x \notin \Sigma^*$ в противном случае. Заметим, что по определению, для любого алфавита Σ пустая строка $\varepsilon \in \Sigma^*$.

Если $x \in \Sigma^*$ – строка длины m, то $x = a_1 a_2 \ldots a_m$, где $a_i \in \Sigma$, $1 \leqslant i \leqslant m$. В таком случае, если $x \in \Sigma^*$ – строка длины m, а $y \in \Sigma^*$ – строка длины n, то их *объединение*, обозначаемое xy, может быть определено как строка длины $m + n$, причем первые m символов составляют строку, совпадающую со строкой x, а последние n символов составляют строку, совпадающую со

строкой y. Например, если $x = a_1a_2 \ldots a_m$, а $y = b_1b_2 \ldots b_n$, тогда $xy = a_1a_2 \ldots a_mb_1b_2 \ldots b_n$. Отметим, что объединение является ассоциативной операцией, т.е. $(xy)z = x(yz)$, но не является коммутативной операцией, поскольку, вообще говоря, $xy \neq yx$. Для пустой строки, очевидно, справедливо следующее утверждение:

$\varepsilon x = x\varepsilon = x$ для любого $x \in \Sigma^*$

Если строка $z \in \Sigma^*$ имеет вид xy, где $x, y \in \Sigma^*$, то строку x называют *префиксом строки* z, а строку y – *суффиксом строки* z. Например, если $z = 00110$, то по определению префиксом строки z является строка ε, или строка 0, или строка 00, или строка 001, или строка 0011, или, наконец, сама строка z. Аналогично суффиксом строки может быть любая из следующих строк:

$\varepsilon, 0, 10, 110, 0110, z$.

Если строки $x, z \in \Sigma^*$ таковы, что для некоторых $w, y \in \Sigma^*$ справедливо соотношение $z = wxy$, то строка x называется *подстрокой* строки z. Например, подстроками строки $z = 00110$ являются строки ε, 0, 1, 00, 01, 10, 11, 001, 011, 110, 0011, 0110 и, наконец, сама строка z.

Формальным языком L над алфавитом Σ называется произвольное подмножество множества Σ^*. Если L_1, L_2 – два формальных языка, то их объединение $L_1L_2 = \{xy | x \in L_1, y \in L_2\}$ также является формальным языком. Например, если $L_1 = \{01, 0\}$ и $L_2 = \{\varepsilon, 0, 10\}$, то $L_1L_2 = \{01, 0, 010, 00, 0110\}$. Как и в случае объединения строк, операция объединения формальных языков ассоциативна, но не коммутативна. Для произвольного формального языка L справедливо следующее утверждение:

$L\{\varepsilon\} = \{\varepsilon\}L = L$.

Заметим, однако, что синглетон, содержащий пустую строку, сильно отличается от пустого множества. Действительно, если L – произвольный формальный язык, то $L\varnothing = \varnothing L = \varnothing$.

Объединение формального языка L с самим собой записывается как L^2 или в общем случае как $L^0 = \{\varepsilon\}, L^1 = L, L^i = LL^{i-1} = L^{i-1}L$ для $i \geqslant 2$. Замыкание Клини формального языка L, обозначаемое L^*, может быть определено как $\bigcup_{i=0}^{\infty} L^i$. Если определить L^+ как $\bigcup_{i=1}^{\infty} L^i$, то определение замыкания Клини формального языка L можно записать как $L^* = L^+ \cup \{\varepsilon\}$. Обычно совокупность строк, принадлежащих множеству Σ^* и имеющих длину 2, обозначают Σ^2, а имеющих длину 3 – соответственно Σ^3, и т.д. Совокупность строк, принадлежащих множеству Σ^* и имеющих длину, большую или равную 1, обозначают Σ^+. Воспользуемся принятыми обозначениями и запишем следующее очевидное утверждение:

$$\Sigma^+ = \bigcup_{i=1}^{\infty} \Sigma^i \text{ и } \Sigma^* = \Sigma^+ \cup \{\varepsilon\} = \bigcup_{i=0}^{\infty} \Sigma^i.$$

УПРАЖНЕНИЯ

1.1. Если $A = \{ab, c\}$ и $B = \{c, ca\}$ – два формальных языка над алфавитом $\{a, b, c\}^*$, вычислите:

$A \cup B$; $A \setminus B$; $A^r \setminus B^r$; AB; BA; $A^2 \cup B^2$.

1.2. Докажите, что множество всех рациональных чисел – бесконечно и счетно.

1.3. Докажите, что множество Σ – бесконечно и счетно, если Σ^* произвольный конечный алфавит.

1.4. *Уровень узла* на дереве $T = (V, E)$ определяется следующим образом: если узел v – корень дерева, то его уровень на дереве равен 0; в противном случае узел $v \in T_i$ является также узлом одного из поддеревьев дерева T, корень которого соединен с корнем дерева T одной дугой, и тогда уровень этого узла на дереве T на единицу больше его уровня на указанном поддереве.

а) Выпишите все уровни всех узлов дерева, показанного на рис. 1.4. *Уровнем дерева* называется максимальный из всех уровней всех его узлов.

б) Пусть T – дерево уровня k, и каждый его внутренний узел имеет по два дочерних узла. Докажите, что число узлов этого дерева n удовлетворяет соотношению $2k+1 \leqslant n \leqslant 2^{k+1}-1$.

1.5. Пусть T – дерево. Покажите, что путь по дереву от корня до любого узла единственен (воспользуйтесь методом индукции).

1.6. Пусть $x \in \Sigma^*$, тогда *элемент, обратный* x, может быть рекурсивно определен следующим образом: если $x = \varepsilon$, то $x^r = \varepsilon$; в противном случае $x = ay$, где $a \in \Sigma$, а $y \in \Sigma^*$, тогда $x^r = y^r a$. Запишите рекурсивное определение $|x|$ – длины строки, и, используя метод индукции, докажите, что $|x| = |x^r|$ для любого $x \in \Sigma^*$ (возможно, понадобится доказать, что $|xy| = |x| + |y|$ для любых $x, y \in \Sigma^*$).

1.7. Докажите, что если R_1, R_2 – два определенных на множестве A отношения, то

а) если R_1 – рефлексивное отношение, то $R_1 \cup R_2$ – также рефлексивное отношение;

б) если R_1 и R_2 – рефлексивные отношения, то $R_1 \cap R_2$ – также рефлексивное отношение;

в) докажите то же самое для свойств симметричности и транзитивности.

1.8. Пусть R – отношение, определенное на множестве $A \times B$, а S – отношение, определенное на множестве $B \times C$. Тогда на множестве $A \times C$ может быть определено отношение RS такое, что RS $\{(a, c) |$ существует $b \in B |$ $(a, b) \in R$ и $(b, c) \in S\}$. Это отношение называют *композицией отношений* R и S.

а) Докажите ассоциативность операции композиции.

б) Докажите, что если R и S – полные функции "на", то функция RS – полная и "на".

1.9. Пусть $R \subset A \times B$ – отношение, определенное на множестве $A \times B$, тогда обратное ему отношение R^{-1} определено на множестве $B \times A$ следующим образом:

$$R^{-1} = \{(b, a) | (a, b) \in R\}.$$

а) Докажите, что $(R^{-1})^{-1} = R$.

б) Каким условиям должна удовлетворять функция R, чтобы функция R^{-1} была полной функцией?

Пусть $A = B$, т. е. отношение R определено на множестве A. Покажите, что

в) R – рефлексивное отношение $\iff$ R^{-1} – рефлексивное отношение;

г) R – симметричное отношение $\iff$ R^{-1} – симметричное отношение;

д) R – транзитивное отношение $\iff$ R^{-1} – транзитивное отношение.

1.10. Множество всех подмножеств множества A обозначают 2^A и называют *степенью множества* A.

а) Пусть $A = \{a, b, c\}$. Выпишите 8 элементов множества 2^A.

б) Покажите, что если $\#(A) = n$, то $\#(2^A) = 2^n$.

1.11. Функция $f : 2^A \times 2^A \to 2^A$ *монотонно возрастает,* если из условия $A_1 \subset B_1$ и $A_2 \subset B_2$ следует, что $f(A_1, A_2) \subset f(B_1, B_2)$. Докажите, что операция конкатенации множеств – монотонно возрастающая функция.

1.12. Пусть $G = (V, E)$ – направленный граф и пусть E^* – рефлексивное замыкание транзитивного замыкания множества E. Докажите, что направленный путь по графу от узла v до узла $w (v, w \in V)$ существует $\iff (v, w) \in E^*$.

ГЛАВА 2

ВВЕДЕНИЕ В ГРАММАТИКИ

"Бесконечность состоит из конечного".

Теодор Ротке "Дальнее поле"

Большинство современных работ, посвященных языкам программирования высокого уровня, обязательно содержит раздел, где формально определяется синтаксис языка программирования. Например, синтаксис языка программирования Паскаль обычно описывается набором *синтаксических диаграмм.* Рассмотрим одну из таких диаграмм, приведенную на рис. 2.1 и иллюстрирующую понятие "множитель".

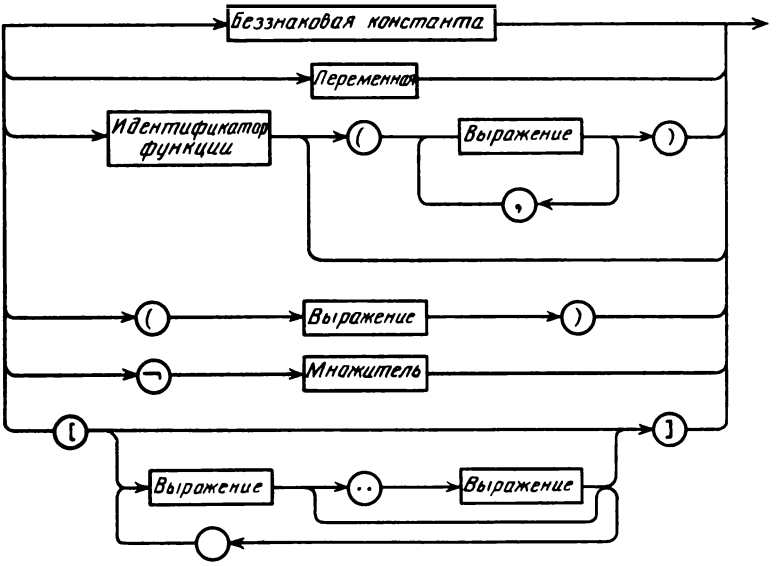
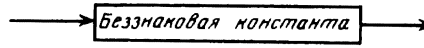
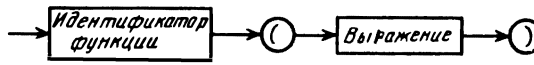


Рис. 2.1

Любой путь по синтаксической диаграмме обязательно пройдет через несколько ее узлов. Например, самый простой путь



проходит всего через один узел. Более сложный путь



проходит через четыре узла диаграммы. Путь

еще более сложен, и т. д.

На самом деле, несложно показать, что число различных путей по рассматриваемой синтаксической диаграмме бесконечно и каждый из этих путей соответствует одному из возможных определений понятия "множитель". Синтаксические диаграммы – один из простых и эффективных способов описания синтаксических определений. Если читатель не пожалеет сил и выпишет словесное определение множителя, то убедится в его чрезвычайной громоздкости.

Каждому узлу приведенной синтаксической диаграммы, изображенному в виде прямоугольника, например, переменная, соответствует, в свою очередь, синтаксическая диаграмма, определяющая понятие внутри прямоугольника. Подобный узел называют *нетерминальным узлом,* а понятие, ему соответствующее, – *нетерминальным символом* языка. Если же узел синтаксической диаграммы изображен в виде круга, например ©, то его называют *терминальным узлом,* а понятие внутри круга – *терминальным символом* языка.

Существует другой путь определения синтаксиса языка, альтернативный синтаксическим диаграммам, – это формы Бэкуса–Наура (БНФ). При использовании форм Бэкуса–Наура нетерминальные символы языка заключаются в угловые скобки вида ⟨. . .⟩ ; запись вида : : = означает "определяется как", а символ | означает "или". Перепишем определение множителя, используя формы Бэкуса–Наура:

⟨множитель⟩ : : = ⟨беззнаковая постоянная⟩
| ⟨переменная⟩
| ⟨идентификатор функции⟩
| ⟨идентификатор функции⟩ (⟨список выражений⟩)
| (⟨выражение⟩)
| ¬⟨множитель⟩
| [ ]
| [⟨список пар выражений⟩]

Здесь ⟨список выражений⟩ и ⟨список пар выражений⟩ – новые нетерминальные символы, которые определяются следующим образом:

⟨список выражений⟩ : : = ⟨выражение⟩
| ⟨выражение⟩, ⟨список выражений⟩

и

⟨список пар выражений⟩ : : = ⟨пара выражений⟩
| ⟨пара выражений⟩, ⟨список пар выражений⟩,

где

⟨пара выражений⟩ : : = ⟨выражение⟩
| ⟨выражение⟩ . . . ⟨выражение⟩.

Нотация БНФ эквивалентна синтаксическим диаграммам. Любая синтаксически корректная программа, написанная на некотором языке программирования, может быть определена в нотации БНФ как нетерминальный символ некоторого языка. Однако необходимо заметить, что указанные синтаксические определения не составляют полного описания языка программирования. Так, в языке программирования Паскаль существует ряд дополнительных требований к исходному тексту программы, не упоминаемых среди синтаксических определений языка программирования, например, требование описания типа всех используемых в программе переменных. Сформулировать это требование с помощью нотации БНФ или эквивалентной ей нотации невозможно. При изучении теории формальных языков важно понимать все возможности и ограничения БНФ и эквивалентных ей нотаций.

КОНТЕКСТНАЯ ГРАММАТИКА

Приведенное ниже определение формальной модели грамматики очень напоминает нотацию БНФ. Будем придерживаться следующих соглашений:

> прописными буквами обозначаются нетерминальные символы языка;
> строчными буквами обозначаются терминальные символы языка;
> символ → используется для обозначения отношения "определяется как".

Таким образом, простой пример грамматики может иметь следующий набор правил вывода:

$S \to A \mid B$

$A \to aA \mid a$

$B \to bB \mid b$

Из первого правила следует, что S либо совпадает с A, либо совпадает с B. Из второго правила следует, что A представляет собой либо строку вида a, либо строку вида aa, либо строку вида $aa \ldots a$. Аналогичные рассуждения справедливы для B.

Если строка β может быть получена из строки α с помощью одного или нескольких правил вывода, то будем записывать это следующим образом: $\alpha \Rightarrow \beta$. Если $\alpha_1 \Rightarrow \alpha_2$, $\alpha_2 \Rightarrow \alpha_3, \ldots, \alpha_{n-1} \Rightarrow \alpha_n (n \geqslant 1)$, то эту последовательность выводов будем кратко записывать так: $\alpha_1 \Rightarrow \alpha_2 \Rightarrow \alpha_3 \Rightarrow \ldots \Rightarrow \alpha_n$, или,

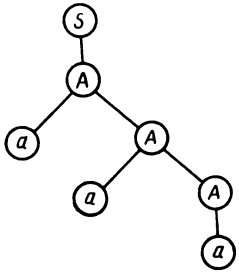


Рис. 2.2

еще более кратко, $\alpha_1 \overset{*}{\Rightarrow} \alpha_n$. Воспользуемся теперь определенной выше грамматикой

$$S \Rightarrow A \Rightarrow aA \Rightarrow aaA \Rightarrow aaaA \Rightarrow aaa$$

или $S \Rightarrow a^3$.

Проиллюстрировать эту последовательность выводов можно и с помощью дерева, изображенного на рис. 2.2.

Теперь необходимо формализовать введенные понятия. До сих пор слева от символа "→" в правилах вывода был лишь один нетерминальный символ языка. Приведенное ниже определение обобщает правила вывода, которые соответствуют формальным понятиям, на строки терминальных и нетерминальных символов языка.

Контекстная грамматика может быть представлена совокупностью четырех объектов (N, T, P, S), где

N – конечное множество нетерминальных символов языка. Согласно принятому ранее соглашению элементы множества N обозначаются прописными буквами (иногда с нижними индексами);

T – конечное множество терминальных символов (поэтому $N \cap T = \emptyset$). Согласно ранее принятому соглашению элементы множества T обозначаются строчными буквами (иногда с нижними индексами);

P – конечное множество *продукций* вида $\alpha \to \beta$, где α – строка в левой части продукции, такая, что $\alpha \in (N \cup T)^+$, а β – строка в правой части продукции, такая, что $\beta \in (N \cup T)^*$; множество $S \in N$ – *начальный символ грамматики.*

Например, пусть $G_1 = (\{S, A, B\}, \{a, b\}, P, S)$, где P представляет собой совокупность продукций вида

$S \to A$, $S \to B$,

$A \to aA$, $A \to a$,

$B \to bB$, $B \to b$.

Перепишем эти продукции, используя обычную нотацию:

$S \to A|B$,

$A \to aA|a$,

$B \to bB|b$.

Заметим, что при соблюдении соглашений об обозначениях терминальных и нетерминальных символов языка, необходимым остается лишь задание набора продукций и начального символа грамматики. Если в дальнейшем начальный символ грамматики будет обозначаться S, то можно опустить и это требование.

Рассмотрим теперь контекстную грамматику G_2, продукции которой имеют вид

$S \to aSBC|aBC$,

$CB \to BC$,

$aB \to ab$,
$bB \to bb$,
$bC \to bc$,
$cC \to cc$.

Это – первый пример, где в левой части продукции указана строка. Можно показать, что из S можно вывести любую строку вида $a^n b^n c^n$, $n \geqslant 1$. Например,

$S \Rightarrow aSBC$,
$\Rightarrow aaBCBC$ (так как $S \to aBC$),
$\Rightarrow aabCBC$ (так как $aB \to ab$),
$\Rightarrow aabBCC$ (так как $CB \to BC$),
$\Rightarrow aabbCC$ (так как $bB \to bb$),
$\Rightarrow aabbcC$ (так как $bC \to bc$),
$\Rightarrow aabbcc$ (так как $cC \to cc$),

т. е. из S выводится $a^2 b^2 c^2$.

До сих пор мы опирались на интуитивное понимание понятия "вывод". Воспользуемся теперь формальным определением грамматики для строгого определения выводимой строки. Пусть $G = (N, T, P, S)$ – контекстная грамматика и пусть $\gamma_1 \alpha \gamma_2 \in (N \cup T)^+$ – строка терминальных и нетерминальных символов длиной $\geqslant 1$. Если $\alpha \to \beta$ – продукция из P, то строка α в строке $\gamma_1 \alpha \gamma_2$ может быть заменена строкой β, и в результате получим $\gamma_1 \beta \gamma_2$.

Это записывают следующим образом:

$$\gamma_1 \alpha \gamma_2 \underset{G}{\Rightarrow} \gamma_1 \beta \gamma_2,$$

при этом говорят, что строка $\gamma_1 \alpha \gamma_2$ генерирует строку $\gamma_1 \beta \gamma_2$, или что строка $\gamma_1 \beta \gamma_2$ выводится из строки $\gamma_1 \alpha \gamma_2$.

Если $\alpha_1, \alpha_2, \ldots, \alpha_n \in (N \cup T)^*$ и $\alpha_1 \underset{G}{\Rightarrow} \alpha_2, \alpha_2 \underset{G}{\Rightarrow} \alpha_3, \ldots, \alpha_{n-1} \underset{G}{\Rightarrow} \alpha_n$ $(n \geqslant 1)$, то обычно пишут сокращенно $\alpha_1 \underset{G}{\Rightarrow} \alpha_2 \underset{G}{\Rightarrow} \alpha_3 \cdots \underset{G}{\Rightarrow} \alpha_{n-1} \underset{G}{\Rightarrow} \alpha_n$, или еще короче, $\alpha_1 \overset{+}{\underset{G}{\Rightarrow}} \alpha_n$, и при этом говорят, что строка α_n выводится из строки α_1 за один или более шагов. Далее, $\overset{+}{\underset{G}{\Rightarrow}}$ обозначает транзитивное замыкание отношения $\underset{G}{\Rightarrow}$, а $\overset{*}{\underset{G}{\Rightarrow}}$ обозначает рефлексивное замыкание отношения $\overset{+}{\underset{G}{\Rightarrow}}$. В таком случае

$$\alpha_1 \overset{*}{\underset{G}{\Rightarrow}} \alpha_n \iff \alpha_1 = \alpha_n \text{ или } \alpha_1 \overset{+}{\underset{G}{\Rightarrow}} \alpha_n.$$

Если строка $\alpha \in (N \cup T)^*$ такая, что $S \overset{*}{\underset{G}{\Rightarrow}} \alpha$, то строку α называют *сентенциальной формой* контекстной грамматики G. *Сентенцией* грамматики G называют произвольную сентенциальную форму из A^*, т. е. произвольную строку терминальных символов, которая может быть выведена из начального символа S. Тогда множество всех сентенций грамматики G называют *языком, порождаемым грамматикой* G, и обозначают $L(G)$.

Таким образом,

$$L(G) = \{x \in T^* \mid S \overset{*}{\underset{G}{\Rightarrow}} x\}.$$

В большинстве примеров рассматривается грамматика G, поэтому символ G можно опустить в формулировках вида $\underset{G}{\Rightarrow}$, $\overset{+}{\underset{G}{\Rightarrow}}$, $\overset{*}{\underset{G}{\Rightarrow}}$ и просто написать $\Rightarrow$, $\overset{+}{\Rightarrow}$, $\overset{*}{\Rightarrow}$.

Предполагаем читателю вернуться к примерам, в которых рассматривались грамматики G_1 и G_2, чтобы убедиться в том, что

$$L(G_1) = \{a^n \mid n \geqslant 1\} \cup \{b^n \mid n \geqslant 1\},$$

$$L(G_2) = \{a^n b^n c^n \mid n \geqslant 1\}.$$

Иногда оказывается возможным, чтобы две различные грамматики G и G', порождали один и тот же язык, т. е. $L(G) = L(G')$. В этом случае говорят, что *грамматики* G и G' *эквивалентны.*

Например, грамматикой, эквивалентной грамматике G_1, является грамматика G_3, продукции которой имеют вид

$$S \to aA \mid bB \mid a \mid b,$$

$$A \to aA \mid a,$$

$$B \to bB \mid b.$$

КОНТЕКСТНО СВОБОДНЫЕ ГРАММАТИКИ

Формальное определение основных синтаксических понятий большинства языков программирования может быть дано с помощью нотации БНФ, при этом вид левой части каждой продукции может быть ограничен лишь единственным нетерминальным символом. Контекстные грамматики, продукции которых удовлетворяют такому ограничению, в теории формальных языков носят название *контекстно свободных грамматик.* Строгое определение контекстно свободной грамматики может быть таким:

> контекстно свободной грамматикой называют такую контекстную грамматику $G = (N, T, P, S)$, каждая продукция которой имеет вид $A \to \beta$, где $A \in N, \beta \in (N \cup T)^*$.

Любой язык, порожденный контекстно свободной грамматикой, называют *контекстно свободным языком.* Поводом к возникновению термина "контекстно свободный" послужил тот факт, что любой символ $A \in N$ в сентенциальной форме грамматики G может быть раскрыт согласно продукции $A \to \beta$ независимо от того, какими строками он окружен внутри самой сентенциальной формы.

Если G_1 и G_3 – две совпадающие контекстно свободные грамматики, то очевидно, что $L = L(G_1) = L(G_3)$ – контекстно свободный язык. Однако мы не можем утверждать, что $L_2 = L(G_2)$ – контекстно свободный язык, так как грамматика G_2 не является контекстно свободной, но, быть может, существует контекстно свободная грамматика, которая порождает тот же язык L_2. Можно даже попытаться определить такую контекстно свободную грамматику (позже будет ясно, что это – пустая трата времени). Однако

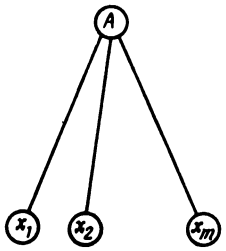


Рис. 2.3

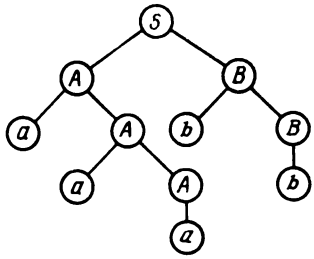


Рис. 2.4

породить язык $\{a^n b^n | n \geqslant 1\}$ с помощью контекстно свободной грамматики возможно, например, с помощью грамматики, имеющей продукцию

$S \to aSb | ab.$

Напомним, что если $G = (N, T, P, S)$ – контекстно свободная грамматика, то любой нетерминальный символ $x \in L(G)$ может быть выведен из начального символа S. Общепринятым методом представления вывода некоторой непустой строки считается *дерево вывода* (иногда его называют *деревом грамматического разбора*). Корнем этого дерева будет начальный символ S, остальные его узлы пометим слева направо следующим образом: $a_1, a_2, \ldots, a_n$, где $a_i \in T$, $1 \leqslant i \leqslant n$ и $x = a_1 a_2 \ldots a_n$; при этом внутренние узлы дерева будут соответствовать нетерминальным символам языка. Пусть A – нетерминальный символ, который может быть расширен с помощью продукции вида $A \to x_1 x_2 \ldots x_m (x_i \in N \cup T, i = 1, \ldots, m)$. В таком случае узел дерева, соответствующий нетерминальному символу A, является родительским узлом для m узлов дерева, соответствующих m нетерминальным символам $x_1, x_2, \ldots, x_m$ (рис. 2.3).

Рассмотрим в качестве примера дерево вывода (рис. 2.4), иллюстрирующее вывод сентенции $a^3 b^3$ контекстно свободной грамматики G_4, имеющей продукции вида

$S \to AB,$

$A \to aA | a,$

$B \to bB | b.$

Это дерево вывода в точности соответствует следующей последовательности выводов:

$S \Rightarrow AB,$

$\Rightarrow aAB,$

$\Rightarrow aaAB,$

$\Rightarrow aaaAB,$

$\Rightarrow aaabB,$

$\Rightarrow aaabb.$

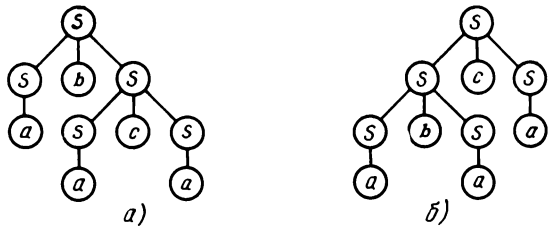


Рис. 2.5

Если $x \in L(G)$, то, как правило, можно найти несколько возможных вариантов вывода. Например, сентенцию a^3b^2 можно вывести в G_4 следующим образом:

$S \Rightarrow AB \Rightarrow AbB \Rightarrow aAbB \Rightarrow aaAbB \Rightarrow aaAbb \Rightarrow aaabb,$

$S \Rightarrow AB \Rightarrow aAB \Rightarrow aAbB \Rightarrow aAbb \Rightarrow aaAbb \Rightarrow aaabb,$

$S \Rightarrow AB \Rightarrow aAB \Rightarrow aaAB \Rightarrow aaAbB \Rightarrow aaabB \Rightarrow aaabb.$

Понятно, что все эти схемы вывода соответствуют одному и тому же дереву вывода.

Если для любого элемента $x \in L(G)$ все возможные схемы его вывода соответствуют одному и тому же дереву вывода, то такую контекстно свободную грамматику называют *однозначной грамматикой*. Если же некоторому элементу $x \in L(G)$ соответствует несколько несовпадающих деревьев вывода, то контекстно свободная грамматика G называется *неоднозначной*. Например, грамматика G_4 является однозначной контекстно свободной грамматикой.

Схема вывода сентенции $x \in L(G)$ называется *левосторонней*, если раскрывается всегда самый левый нетерминальный символ сентенциальной формы. Поскольку теперь каждому дереву вывода соответствует единственная левосторонняя схема вывода, то можно переопределить неоднозначную грамматику следующим образом: контекстно свободная грамматика G называется неоднозначной, если существует такая сентенция $x \in L(G)$, которой можно поставить в соответствие две различные левосторонние схемы вывода. Рассмотрим неоднозначную грамматику G_5, имеющую продукцию

$S \to SbS \mid ScS \mid a.$

Пусть сентенции $abaca \in L(G_5)$ соответствуют две различные левосторонние схемы вывода

$S \Rightarrow SbS \Rightarrow abS \Rightarrow abScS \Rightarrow abacS \Rightarrow abaca,$

$S \Rightarrow ScS \Rightarrow SbScS \Rightarrow abScS \Rightarrow abacS \Rightarrow abaca.$

Соответствующие этим схемам вывода деревья вывода представлены на рис. 2.5.

ГРАММАТИЧЕСКИЙ РАЗБОР АРИФМЕТИЧЕСКИХ ВЫРАЖЕНИЙ

Рассмотрим грамматику, предназначенную для описания арифметических выражений, содержащих переменные a, b, c, и имеющую следующие продукции:

⟨выражение⟩ : :=⟨терм⟩ | ⟨выражение⟩ + ⟨терм⟩ | ⟨выражение⟩ − ⟨терм⟩,
⟨терм⟩ : :=⟨множитель⟩ | ⟨терм⟩ × ⟨множитель⟩ | ⟨терм⟩ / ⟨множитель⟩,
⟨множитель⟩ : :=$a|b|c|$ (⟨выражение⟩).

Используя формальную нотацию контекстно свободных грамматик и обозначая начальный элемент грамматики E, запишем эти продукции по-другому:

$E \to T|E + T|E - T,$

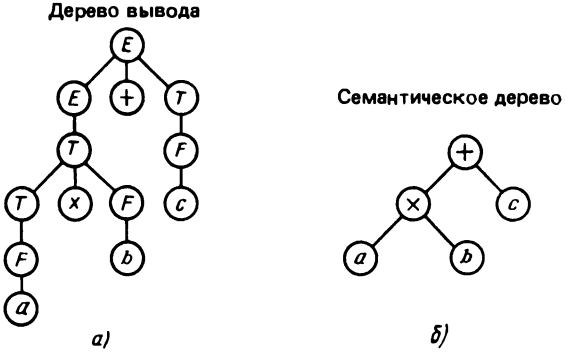
$T \to F|T \times F|T / F,$

$F \to a|b|c|(E).$

Рассмотренная грамматика является однозначной грамматикой, поскольку каждой сентенции, порождаемой этой грамматикой, соответствует единственное дерево вывода. Одно из таких деревьев вывода представлено на рис: 2.6, *а*. После грамматического разбора такого дерева компилятор построит другое дерево, называемое *семантическим деревом*. Семантическое дерево, соответствующее строке $a \times b + c$, представлено на рис. 2.6,*б*, и читатель легко поймет, каким образом оно было получено из дерева вывода. После соответствующей обработки полученного семантического дерева компилятор сгенерирует последовательность машинных инструкций, которые составят программу для вычисления указанного арифметического выражения. Обычно инструкции имеют следующий вид:

LOAD a
MULT b
ADD c

Грамматика, рассмотренная выше в качестве примера, представляет упрощенный вариант грамматики, которая используется для синтаксичес-



а) *б)*

Рис. 2.6

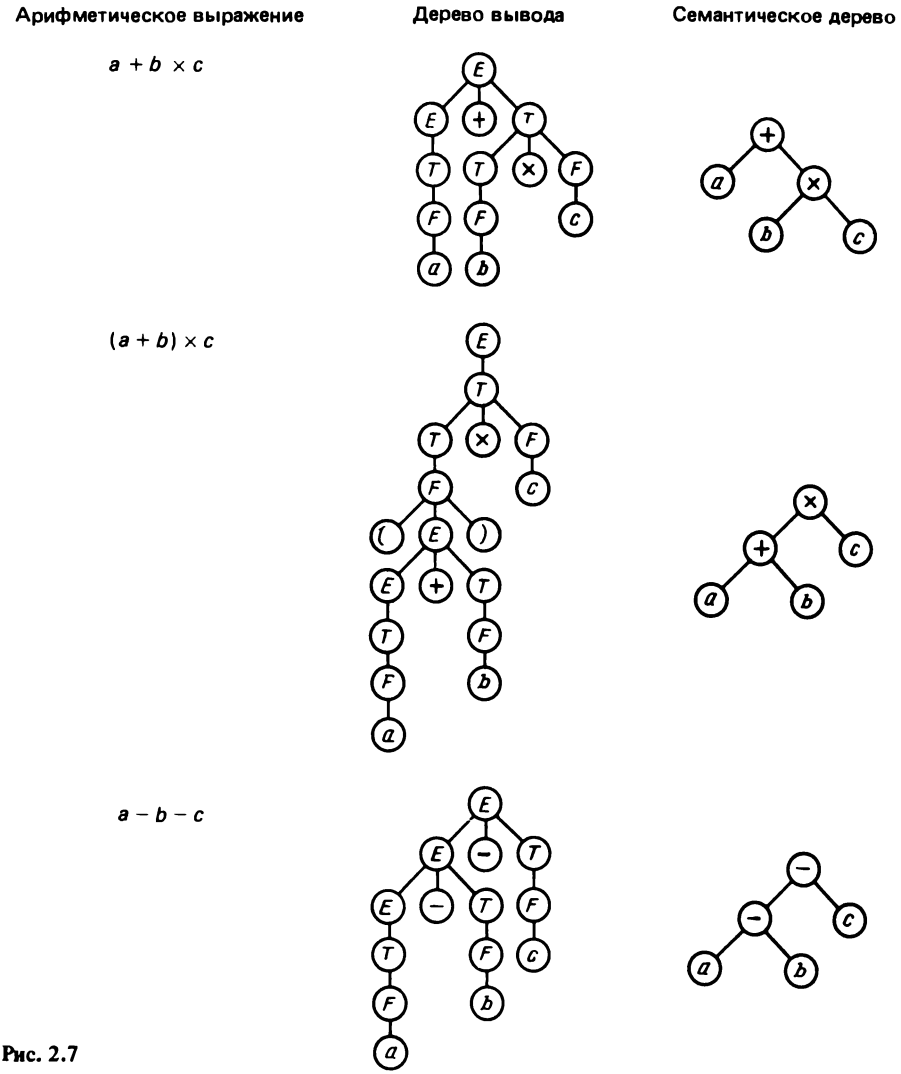


Рис. 2.7

кого определения арифметического выражения в языке программирования Паскаль. Вообще, если мы хотим, чтобы арифметическое выражение имело привычную для нас семантическую интерпретацию (например, приоритет арифметических операций, /, +, −), мы должны позаботиться не только об однозначности используемой грамматики, но и о подходящем выборе набора ее продукций. На рис. 2.7 проиллюстрированы приведенные выше соображения.

Было бы идеально определять синтаксис любого языка программирования с помощью контекстно свободных грамматик. Человечество накопило боль-

шой опыт использования таких грамматик, а техника построения деревьев вывода хорошо изучена. Если контекстно свободная грамматика обладает некоторыми дополнительными свойствами (мы обсудим их в следующих главах), то в ряде случаев оказывается возможным и достаточным построить эффективный алгоритм грамматического разбора. К сожалению, хотя большая часть синтаксических правил языка и может быть описана таким способом, обязательно найдутся правила, которые не могут быть выражены в терминах контекстно свободных грамматик. Поэтому, несмотря на то, что значительная часть кода компилятора и реализует различные алгоритмы грамматического разбора, ведущего к построению семантических деревьев, тем не менее компилятор содержит и различные дополнительные проверки и таблицы параметров и т. д.

ПУСТАЯ СТРОКА В ОПРЕДЕЛЕНИИ КОНТЕКСТНО СВОБОДНОЙ ГРАММАТИКИ

В приведенном ранее определении контекстно свободной грамматики ее продукции обязательно должны были иметь вид $A \to \beta$, где A – нетерминальный символ языка, а β – некоторая строка из терминальных и нетерминальных символов. Это означает, что, строго говоря, корректна следующая запись: $A \to \varepsilon$. Будем называть такую продукцию ε-продукцией. Наличие у контекстно свободной грамматики ε-продукции осложняет как процедуру грамматического разбора, так и использование грамматики для описания формального языка. Если вы были внимательны, то теперь непременно удивитесь тому, что до сих пор ε-продукции не появлялись в рассматриваемых нами деревьях вывода. Грамматика, не имеющая ε-продукции, называется *ε-свободной грамматикой.* Конечно, было бы идеально иметь дело лишь с ε-свободными грамматиками, однако если $\varepsilon \in L(G)$ для некоторой контекстно свободной грамматики, то избавиться от этой пустой строки и не изменить порождаемого формального языка невозможно. Самое большее, на что можно рассчитывать, это преобразовать грамматику G таким образом, чтобы полученная контекстно свободная грамматика G' была ε-свободной грамматикой и выполнялось бы условие $L(G') = L(G) \setminus \{\varepsilon\}$. К счастью, найти и проделать такие преобразования несложно.

Пусть $G = (N, T, P, S)$ – контекстно свободная грамматика, имеющая ε-продукцию. Тогда построить контекстно свободную грамматику $G' = (N, T, P, S)$, обладающую описанными выше свойствами, можно следующим образом:

1. Объединим все имеющиеся в P ε-продукции в множество P'

2. Рассмотрим все нетерминальные символы $A \in N$, такие, что $A \overset{*}{\underset{G}{\Rightarrow}} \varepsilon$, назовем такие нетерминальные символы *ε-порождающими* нетерминальными символами. Затем каждой продукции $p \in P'$, содержащей в правой части ε-порождающий нетерминальный символ, поставим в соответствие такую продукцию, что в ее правой части опущены (по сравнению с продукцией p) один или более нетерминальных символов, являющихся для грамматики G ε-порождающими символами, и присоединим ее к множеству P'

Докажем теперь, что $L(G') = L(G) \setminus \{\varepsilon\}$. Пусть контекстно свободная грамматика G имеет продукции вида

$S \to [E] | E,$

$E \to T | E + T | E - T,$

$T \to F | T \times F | T / F,$

$F \to a | b | c | \varepsilon,$

тогда $S \overset{*}{\Rightarrow} \varepsilon$, $E \overset{*}{\Rightarrow} \varepsilon$, $T \overset{*}{\Rightarrow} \varepsilon$ и $F \overset{*}{\Rightarrow} \varepsilon$.

Таким образом, построенная контекстно свободная грамматика имеет следующие продукции:

$S \to [E] | [\ ] | E,$

$E \to T | E + T | E - T | E + | E - | + T | - T | + | -,$

$T \to F | T \times F | T / F | T \times | T / | \times F | / F | \times | /,$

$F \to a | b | c.$

Для доказательства теоремы необходимо сначала доказать следующую лемму.

Лемма. Для любых $A \in N$ и $x \in T^*$ справедливо следующее утверждение:

$A \underset{G}{\overset{*}{\Rightarrow}} x \iff A \underset{G'}{\overset{*}{\Rightarrow}} x.$

Доказательство.

Прежде всего покажем, что если $A \underset{G'}{\overset{*}{\Rightarrow}} x$, то $A \underset{G}{\overset{*}{\Rightarrow}} x$. Это непосредственно следует из справедливости следующего утверждения: если $X \to \beta$ – продукция G', то $X \underset{G}{\overset{*}{\Rightarrow}} \beta$. Итак, любая схема вывода, корректная в грамматике G' корректна и в грамматике G. Для доказательства утверждения, что из $A \underset{G'}{\overset{*}{\Rightarrow}} x$ следует, что $A \underset{G}{\overset{*}{\Rightarrow}} x$, для любых $A \in N$ и $x \in T^+$, воспользуемся индукцией по n – числу шагов вывода.

При $n = 0$ – тривиальный случай. Если $n = 1$, то очевидно, P содержит одну продукцию (пусть она имеет вид $A \to x$) и поскольку $x \neq \varepsilon$, эта продукция также принадлежит и множеству продукций P' грамматики G', т. е. $A \underset{G'}{\Rightarrow} x$. Предположим, что утверждение справедливо для всех $n \leqslant k$ шагов вывода непустой строки терминальных символов из нетерминального символа. Пусть непустая строка терминальных символов x выводится из нетерминального символа A на $(k + 1)$-м шаге. Тогда, очевидно, P содержит продукцию вида $A \to X_1 X_2 \ldots X_m$, $X_i \in N \cup T$, где $A \underset{G}{\Rightarrow} X_1 X_2 \ldots X_m \underset{G}{\overset{*}{\Rightarrow}} x$. Разобьем строку x на подстроки $x = x_1 x_2 \ldots x_m$ таким образом, что если $X_i \in N$, то $X_i \underset{G}{\overset{*}{\Rightarrow}} x_i$, а если $X_i \in T$, то $X_i = x_i$ (и, следовательно, по определению $X_i \underset{G}{\overset{*}{\Rightarrow}} x_i$). Некоторые подстроки x_i могут оказаться пустыми, так как грамматика G имеет ε-продукции. Пусть индексы $1 \leqslant i1 < i2 < \ldots < ir \leqslant m$, такие,

что $x_{i1}, x_{i2}, \ldots, x_{ir}$ — непустые подстроки из последовательности подстрок $x_1 \ldots x_m$, полученной из непустой строки терминальных символов. Тогда, очевидно, $x = x_{i1}x_{i2} \ldots x_{ir}$ и $X_{ij} \overset{*}{\underset{G}{\Rightarrow}} x_{ij}$, где $j = 1, \ldots, r$. Но каждый из таких выводов состоит из числа шагов вывода, не превышающих k, и для него справедливо предположение, что $X_{ij} \overset{*}{\underset{G'}{\Rightarrow}} x_{ij}$, для $j = 1, 2, \ldots, r$

Таким образом, из утверждения, что $A \to X_{i1}X_{i2} \ldots X_{ir}$ — продукция грамматики G', следует, что $A \overset{*}{\underset{G'}{\Rightarrow}} x$ для $k + 1$ шагов вывода. Из этого по индукции получаем, что утверждение верно для любого k — числа шагов вывода, что и требовалось доказать.

Из только что доказанной леммы можно сделать вывод, что $S \overset{*}{\underset{G}{\Rightarrow}} x \Leftrightarrow S \overset{*}{\underset{G'}{\Rightarrow}} x$ для любого $x \in T^+$ и, следовательно, $x \in L(G) \setminus \{\varepsilon\} \Leftrightarrow x \in L(G')$. Сформулируем теперь теорему, доказательство которой мы только что получили.

ТЕОРЕМА 2.1

Для любого контекстно свободного языка L существует ε-свободная контекстно свободная грамматика G, такая, что $L(G) = L \setminus \{\varepsilon\}$.

Позже мы покажем, что если $\varepsilon \in L$, где L — некоторый контекстно свободный язык, то контекстно свободная грамматика G, порождающая язык L, обязательно имеет только одну продукцию вида $S \to \varepsilon$, при этом не существует более ни одной продукции грамматики G, в правой части которой содержался бы нетерминальный символ S. Объяснить этот факт можно следующим образом: пусть $G = (N, T, P, S)$ — ε-свободная контекстно свободная грамматика, порождающая язык $L \setminus \{\varepsilon\}$; S' — нетерминальный символ и $G' = (N \cup \{S'\}, T, P \cup \{S' \to \varepsilon \mid S\}, S')$, тогда контекстно свободная грамматика G' порождает язык $L(G') = L$.

УПРАЖНЕНИЯ

2.1. Определите контекстно свободные грамматики, которые порождали бы следующие языки:

а) все строки — элементы множества $\{0, 1\}^*$, такие, что в каждой из них непосредственно справа от каждого символа 0 стоит символ 1;

б) все строки — элементы множества $\{0, 1\}^*$, такие, что результаты чтения этих строк символов слева направо и справа налево совпадают;

в) все строки — элементы множества $\{0, 1\}^*$, которые содержат символов 0 вдвое больше, чем символов 1.

2.2. Пусть некоторая контекстно свободная грамматика имеет продукции вида

$S \to AB,$
$A \to SA \mid BB \mid bB,$
$B \to b \mid aA \mid \varepsilon.$

Определите новую контекстно свободную грамматику, которая была бы эквивалентна данной, но имела бы единственную ε-продукцию вида $S \to \varepsilon$.

2.3. Покажите, что грамматика, имеющая продукции вида

$S \to bA \mid aB,$
$A \to a \mid aS \mid bAA,$
$B \to b \mid bS \mid aBB,$

порождает язык $L \subset \{a, b\}^*$, составленный из строк, содержащих равное число символов a и символов b. (Воспользуйтесь методом индукции и докажите, что для любой сентенциальной формы общее число элементов a и A равно общему числу элементов b и B.)

2.4. Докажите, что язык $L(G_2) = \{a^n b^n c^n \mid n \geqslant 1\}$, где G_2 – контекстная грамматика, рассмотренная в этой главе.

2.5. Покажите, что грамматика, определенная в упражнении 2.3, неоднозначна. Покажите, что грамматика, имеющая продукции вида

$S \to aB \mid aB \mid bAS \mid bA,$
$A \to bAA \mid a,$
$B \to aBB \mid b,$

есть однозначная грамматика, порождающая тот же язык.

2.6. Если каждая продукция грамматики G, не являющаяся ε-продукцией, имеет форму $A \to aB$ или $A \to a$, где $A, B \in N$, $a \in T$, то грамматику G называют *регулярной грамматикой*.

Покажите, что если G – регулярная грамматика, то ε-свободная грамматика G', полученная из грамматики G при доказательстве теоремы 2.1, также является регулярной.

2.7. Проверьте, являются ли следующие исходные тексты синтаксически корректными программами на языке программирования Паскаль:

а)
```
program ex (output);
  begin
      if 1+1 =2 then write ('hooray')
  end
```

б)
```
program ex2;
  begin
  write (1 + 1)
  end.
```

в)
```
program copy (f1, f2);
  var f1, f2; file of char;
  begin reset(f1); rewrite(f2);
        while not eof(f1) do
              begin f2↑:=f1; put(f2); get(f1);
              end
  end
```

2.8. Напишите на языке программирования Паскаль программу, корректную с точки зрения синтаксических определений, но некорректную с точки зрения компилятора этого языка.

ГЛАВА 3

РЕГУЛЯРНЫЕ ЯЗЫКИ, I

"Вся трудность – в выборе".

Джордж Моор "Ветви гнутся"

Приведем пример использования нотации БНФ для определения целого беззнакового числа:

⟨последовательность цифр⟩ : : = 0|1|2|3|4|5|6|7|8|9
|0⟨последовательность цифр⟩ | 1 ⟨последовательность цифр⟩
|2⟨последовательность цифр⟩ | 3 ⟨последовательность цифр⟩
|4⟨последовательность цифр⟩ | 5 ⟨последовательность цифр⟩
|6⟨последовательность цифр⟩ | 7 ⟨последовательность цифр⟩
|8⟨последовательность цифр⟩ | 9 ⟨последовательность цифр⟩;

⟨беззнаковое целое число⟩ : : = 0 1|2|3|4|5|6|7|8|9
|1⟨последовательность цифр⟩ | 2 ⟨последовательность цифр⟩
|3⟨последовательность цифр⟩ | 4 ⟨последовательность цифр⟩
|5⟨последовательность цифр⟩ | 6 ⟨последовательность цифр⟩
|7⟨последовательность цифр⟩ | 8 ⟨последовательность цифр⟩
|9⟨последовательность цифр⟩.

Правая часть каждой формы представляет собой либо один терминальный символ, либо последовательность терминальных и нетерминальных символов. Все символы любого языка программирования (целые числа, идентификаторы переменных, операторы, зарезервированные слова, знаки пунктуации) могут быть определены таким образом. Поскольку значительную часть времени компиляции программирования составляет распознавание символов языка программирования в исходном тексте программы, то есть смысл изучить грамматики, имеющие продукции такого же вида. Подобные грамматики называют регулярными грамматиками, а порожденные ими языки программирования – регулярными языками. В этой главе будет показано, что тексты, написанные на регулярном языке, поддаются эффективному распознаванию, а сами регулярные грамматики могут обладать широким спектром необходимых нам свойств. К сожалению, возможности регулярных языков сильно ограничены, и даже самые простые программные конструкции не могут быть описаны с использованием одних лишь регулярных грамматик. Таким образом, роль регулярных грамматик в процессе компиляции программы ограничивается лишь распознаванием символов языка, использованных в программе – этот этап компиляции носит название *лексического анализа.* После того, как лексический анализ закончен, требуется более изощренная техника для осуществления полного грамматического разбора текста программы. В следующих главах мы обсудим этот вопрос.

РЕГУЛЯРНЫЕ ГРАММАТИКИ

Контекстная грамматика $G=(N,T,P,S)$ называется *регулярной грамматикой*, если:

1) она имеет ε-продукцию вида $S\to\varepsilon$, и ни одна из оставшихся продукций G не содержит своей правой части нетерминального символа S;

2) все остальные продукции грамматики G имеют вид $A\to a$, где $A\in N$, $a\in T$ или вид $A\to aB$, где $A,B\in N, a\in T$.

Язык называется *регулярным языком*, если он порожден регулярной грамматикой. Из этого определения и рассуждений в теореме 2.1, можно сделать вывод, что язык L является регулярным $\Leftrightarrow L\setminus\{\varepsilon\}$ порожден ε-свободной грамматикой.

Регулярная грамматика G_1, имеющая продукции вида

$$S\to aS|aB,$$

$$B\to bB|b$$

порождает регулярный язык $L(G_1)=\{a^mb^n|m,n\geqslant 1\}$.

Регулярный язык $L=L(G_1)\cup\{\varepsilon\}$ порожден регулярной грамматикой, имеющей продукции вида

$$S\to\varepsilon|aS_1|aB,$$

$$S_1\to aS_1|aB,$$

$$B\to bB|b.$$

Поскольку каждая продукция регулярной грамматики специфицирует замену нетерминального символа строкой, содержащей не более одного нетерминального символа, то каждая сентенциальная форма такой грамматики либо является элементом множества T^* либо содержит ровно один элемент множества N, который, кроме того, обязательно будет самым правым ее символом.

Если $G=(N, T, P, S)$ – ε-свободная регулярная грамматика, порождающая язык L, можно легко построить ε-свободную грамматику $G^+=(N, T, P^+, S)$, порождающую язык L^+, поставив в соответствие каждому элементу множества P, имеющему вид $A\to a$, продукцию вида $A\to aS$, которую присоединим к множеству P. Полученное в результате множество продукций и есть множество P^+. Например, грамматика G_1^+ имеет продукции вида

$$S\to aS|aB,$$

$$B\to bB|b|bS.$$

Вернемся к общему случаю и докажем, что $L(G^+)=L^+$. Для этого достаточно показать, что $L(G^+)\subset L^+$ и $L^+\subset L(G^+)$. Докажем, что если $x\in L(G^+)$, то $x\in L^+$, воспользовавшись методом индукции по n, где n – число "новых" продукций (тех продукций, которые принадлежат множеству $P^+\setminus P$), используемых для вывода строки x. Пусть $n=0$, тогда схема вывода строки x содержит только те продукции, которые принадлежат P, поэтому $x\in L$ и, следовательно, $x\in L^+$.

Предположим теперь, что для всех $n<k$ строка $x\in L^+$. Рассмотрим схе-

му вывода $x \in L(G^+)$, которая включает k "новых" продукций, а продукция $A \to aS$ – последняя из них. Тогда $S \overset{+}{\underset{G^+}{\Rightarrow}} yA \underset{G^+}{\Rightarrow} yaS \overset{+}{\underset{G^+}{\Rightarrow}} yaz = x$, где $y, z \in T^+$, $a \in T$. Очевидно, что вывод $yaS \overset{+}{\underset{G^+}{\Rightarrow}} yaz$ не включает новых продукций и, следовательно, $yaS \overset{+}{\underset{G}{\Rightarrow}} yaz$, значит, $S \overset{+}{\underset{G}{\Rightarrow}} z$, т. е. $z \in L$. Поскольку $A \to aS$ – "новая" продукция, то в множестве P должна существовать продукция вида $A \to a$. Тогда $S \overset{+}{\underset{G^+}{\Rightarrow}} yA \Rightarrow ya$ – вывод элемента $ya \in L(G^+)$ включает меньше k "новых" продукций, поэтому $ya \in L^+$ согласно предположению. Но из того, что $ya \in L^+$, а $z \in L$ следует, что $x = yaz \in L^+$, и утверждение доказано.

Если $x \in L^+$, то $x \in L^n$ для некоторого $n \geqslant 1$. Докажем методом индукции по n, что если $x \in L^n$, то $x \in L(G^+)$ для любого n. Пусть $n = 1$, тогда $x \in L$, и x может быть выведено из S с помощью одних лишь продукций грамматики G. Поскольку все эти продукции являются также элементами множества P^+, этот вывод справедлив и для грамматики G^+ и, следовательно, $x \in L(G^+)$. Предположим, что это утверждение справедливо для всех $n < k$ и рассмотрим $x \in L^k$. Тогда, очевидно, строка x может быть представлена в виде $x = yz$, где $y \in L$, $z \in L^{k-1}$. Пусть $S \overset{+}{\underset{G}{\Rightarrow}} y$ – вывод строки y в грамматике G. Последняя входящая в состав этого вывода продукция, очевидно, будет иметь вид $A \to a$, где $y = y'a$, $y' \in T^*$. Таким образом, $A \to aS$ – продукция из множества P^+ и, следовательно, yS – сентенциальная форма контекстной грамматики G^+. Согласно индуктивному предположению, существует схема вывода строки z из S в грамматике G^+ и, следовательно, $yz \in L(G^+)$, т. е. $x \in L(G^+)$ для любого k, что и требовалось доказать.

Пусть теперь L – некоторый регулярный язык, тогда $L^* = (L \setminus \{\varepsilon\}^+) \cup \{\varepsilon\}$. Как мы только что показали, $(L \setminus \{\varepsilon\})^+$ – регулярный язык, и, следовательно, L^* – тоже регулярный язык. Сформулируем теперь теорему, первую часть которой мы только что доказали.

ТЕОРЕМА 3.1

а) Если L – регулярный язык, то его так называемое замыкание Клини L^* – тоже регулярный язык.

б) Если L_1 и L_2 – регулярные языки, то $L_1 \cup L_2$ и $L_1 L_2$ – регулярные языки.

Доказательство второй части теоремы также основывается на построении новой грамматики. Основная схема доказательства будет изложена, а детали доказательства оставлены читателю. Прежде всего, предположим, что L_1 и L_2 не содержат элемента ε (если $\varepsilon \in L_1$ или $\varepsilon \in L_2$, тогда $\varepsilon \in L_1 \cup L_2$ и грамматика, порождающая язык $L_1 \cup L_2$ может быть построена из грамматики, порождающей язык $(L_1 \setminus \{\varepsilon\}) \cup (L_2 \setminus \{\varepsilon\})$). Прежде, чем приступить к построению грамматики, рассмотрим один пример. Пусть $L_1 = \{a^m b^n | m, n \geqslant 1\}$ – язык, порожденный грамматикой G_1, описанной выше, и пусть L_2 – язык, порожденный грамматикой G_2, имеющей продукцию $S \to cS | c$, тогда $L_2 = \{c^n | n \geqslant 1\}$.

Пусть набор продукций грамматики, которую мы собираемся построить

и которая порождает язык $L_1 \cup L_2$, представляет собой просто переписанные вместе продукции грамматик G_1 и G_2. Считать такую грамматику искомой нельзя, поскольку в результате "спутывания" продукций двух грамматик могут возникнуть такие схемы вывода, что порожденные ими строки не будут входить в язык $L_1 \cup L_2$.

Если это рассуждение использовать в рассмотренном выше примере, то одна из таких схем вывода может выглядеть следующим образом: $S \Rightarrow aS \Rightarrow \Rightarrow ac$.

Мы должны быть уверены, что любая схема вывода состоит из продукций грамматики G_1 или продукций грамматики G_2. Чтобы добиться этого, необходимо различать нетерминальные символы грамматики G_1 и нетерминальные символы грамматики G_2. Пусть $G_1 = (\{S_1, B_1\}, \{a, b\}, P_1, S_1)$ имеет продукции

$$S_1 \to aS_1 | aB_1,$$

$$B_1 \to bB_1 | b$$

и $G_2 = (\{S_2\}, \{c\}, P_2, S_2)$ имеет продукции

$$S_2 \to cS_2 | c,$$

где S_1, B_1 и S_2 – несовпадающие нетерминальные символы.

Объединим теперь эти продукции и введем новый начальный символ S. Однако пополнить объединенный набор продукций еще одной продукцией вида $S \to S_1 | S_2$ нельзя, так как регулярная грамматика, по определению, не может иметь продукции такого вида. Поэтому запишем продукции вида $S \to \alpha$ для всех таких α, что $S_1 \to \alpha$ – продукция грамматики G_1 или $S_2 \to \alpha$ – продукция грамматики G_2. Применим это рассуждение к вышеприведенному примеру

$$S \to aS_1 | aB_1 | cS_2 | c,$$

$$S_1 \to aS_1 | aB_1,$$

$$B_1 \to bB_1 | b,$$

$$S_2 \to cS_2 | c.$$

Нетрудно видеть, что данная грамматика порождает язык $\{a^m b^n | m, n \geq 1\} \cup \cup \{c^n | n \geq 1\}$.

В результате всех этих рассуждений, можно формально записать искомую грамматику. Пусть $G_1 = (N_1, T_1, P_1, S_1)$ и $G_2 = (N_2, T_2, P_2, S_2)$ – две ε-свободные регулярные грамматики, порождающие языки L_1 и L_2 соответственно. Без потери общности можно предположить, что $N_1 \cap N_2 = \emptyset$. Пусть $S \notin N_1 \cup N_2$ – новый начальный символ, и рассмотрим грамматику $G = (N_1 \cup \cup N_2 \cup \{S\}, T_1 \cup T_2, P_1 \cup P_2 \cup P_3, S)$, где P_3 – множество всех продукций вида $S \to \alpha$, где либо $S_1 \to \alpha \in P_1$, либо $S_2 \to \alpha \in P_2$. Предлагаем читателю самостоятельно доказать, что $L(G) = L(G_1) \cup L(G_2)$.

С помощью грамматик G_1 и G_2 построим еще одну грамматику G', такую, что $L(G') = L(G_1) L(G_2)$. При этом $N_1 \cap N_2 = \emptyset$, $G' = (N_1 \cup N_2, T_1 \cup T_2, P', S_1)$, где P' – множество всех продукций грамматик G_1 и G_2, в котором

продукции грамматики G_1 вида $A \to a$ заменены на продукции вида $A \to aS_2$, и снова предложим читателю самостоятельно доказать, что $L_1 L_2$ – регулярный язык. В вышеприведенном примере продукции вновь построенной грамматики имеют вид

$S_1 \to aS_1 | aB_1,$

$B_1 \to bB_1 | bS_2,$

$S_2 \to cS_2 | c.$

КОНЕЧНЫЙ АВТОМАТ

Любая регулярная грамматика $G = (N, T, P, S)$ может быть представлена направленным графом с помеченными узлами и дугами. Каждый узел такого графа может быть помечен символом, являющимся элементом множества N, кроме одного специального – "конечного" узла, помеченного символом #. Согласно принятым соглашениям, узел, соответствующий начальному символу S, выделяется среди всех прочих узлов графа с помощью символа ↘, а конечный узел, соответствующий символу #, изображается в виде прямоугольника. Если грамматика G имеет продукцию $A \to aB$, то узел графа, помеченный символом A, должен быть соединен непосредственно с узлом, помеченным символом B, с помощью дуги, которую принято в таких случаях помечать символом a. Если грамматика G имеет продукцию вида $A \to a$, то соответствующий узел графа, помеченный символом A, соединен с узлом, помеченным символом #, дугой, помеченной символом a. Таким образом, каждая дуга графа соответствует одной и только одной продукции грамматики G.

Пусть регулярная грамматика G_3 имеет следующие продукции:

$S \to aA | bB,$

$A \to aA | a,$

$B \to bB | b,$

тогда она, очевидно, может быть представлена с помощью направленного графа, изображенного на рис. 3.1.

Рассмотрим теперь некоторый путь по графу, соединяющий узел, помеченный символом S, с узлом, помеченным символом #. Метки, ассоциированные с дугами, составляющими этот путь, составят некоторую строку. Множество таких строк совпадает с языком $L(G_3)$, а каждый такой путь будет соответствовать одному из реальных выводов. Таким образом, путь, составленный из дуг вида: "от Ⓢ к Ⓐ", "от Ⓐ к Ⓐ", "от Ⓐ к Ⓐ", от Ⓐ к ⊕" в точности соответствует схеме вывода вида $S \Rightarrow aA \Rightarrow aaA \Rightarrow aaaA \Rightarrow aaaa.$

Говорят, что конечный направленный граф, имеющий один начальный узел и один или более конечных узлов, есть *конечный автомат*. Мы ограничимся рассмотрением лишь *детерминированных конечных автоматов*, т. е. таких конечных автоматов, ни один узел которых не имеет одинаково помеченных дуг, соединяющих его с другими узлами автомата. В вышеприведенном примере конечный автомат, представленный на рис. 3.1, не являет-

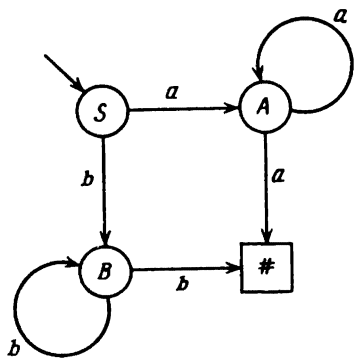


Рис. 3.1

ся детерминированным конечным автоматом, ибо узел A имеет две одинаково помеченные дуги, соединяющие его с другими узлами автомата.

Детерминированным конечным автоматом называют совокупность 5 объектов $M = (K, T, t, k_1, F)$, где

а) K – конечное множество состояний автомата;

б) T – конечный входной алфавит;

в) t – переходная функция, преобразующая совокупность текущего состояния автомата и входного алфавита в состояние автомата на следующем шаге вывода;

г) k_1 – так называемое начальное состояние конечного автомата;

д) F – множество так называемых конечных состояний автомата.

Мы будем по-прежнему представлять детерминированный конечный автомат в виде помеченного направленного графа, узлы которого соответствуют состояниям конечного автомата, и считать, что его начальное состояние соответствует узлу графа, помеченному символом ↘, а его конечное состояние соответствует узлу графа, изображенному в виде прямоугольника. В качестве примера рассмотрим детерминированный конечный автомат, изображенный на рис. 3.2. Запишем его с помощью пяти объектов: $(\{A, B, C, D, E\}, \{a, b\}, t, A, \{D, E\})$, где переходная функция t задана табл. 3.1.

Таблица 3.1

Состояние (K)	*Ввод* (T)	*Следующее состояние* (K)
A	a	B
A	b	C
B	a	D
B	b	B
C	a	D
C	b	C
D	b	E
E	a	E

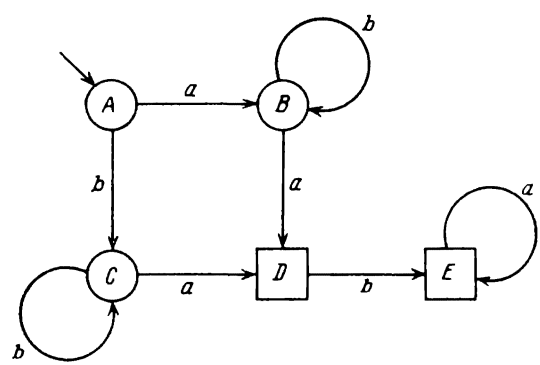


Рис. 3.2

Отметим, что в данном случае t – неполная функция, так как она не определена на элементах (D, a) и (E, b). Альтернативным представлением переходной функции является *переходный массив*. Если $M = (K, T, t, k_1, F)$ детерминированный конечный автомат, то переходный массив A_t, реализующий фактически переходную функцию t, представляет собой матрицу, элементы которой – целые числа. Действительно, пусть $k_1, k_2, \ldots, k_n$ – упоря-

доченная последовательность состояний автомата, а $a_1, a_2, \ldots, a_m$ — упорядоченный входной алфавит; тогда элементы матрицы A_t удовлетворяют условию

$$A_t(i,j) = l \Leftrightarrow t(k_i, a_j) = k_l.$$

Если, например, и множество меток состояний конечного автомата, и входной алфавит упорядочены по алфавиту, то соответствующий переходной массив в рассмотренном выше примере имеет вид

$$\begin{pmatrix} 2 & 3 \\ 4 & 2 \\ 4 & 3 \\ . & 5 \\ 5 & . \end{pmatrix}$$

Если (как в нашем примере) t — неполная функция, то не все элементы матрицы A_t определены.

Нас будут интересовать строки, составленные из символов, которыми помечены дуги, соединяющие исходный узел конечного автомата с некоторым его конечным узлом. Множество всех таких строк для данного детерминированного конечного автомата M обычно обозначают $T(M)$ и называют множеством *строк, принимаемых* M. Чтобы дать строгое определение $T(M)$, расширим определение переходной функции, определив ее как функцию $t: K \times T^* \to K$. Если $k \in K$ и $x \in T^*$, то $t(k, x)$ — такое состояние автомата, которое будет достигнуто, если начиная из узла, помеченного символом k, проследить путь, отвечающий строке x как совокупности символов, помечающих отдельные дуги автомата. Понятно, что если $x = \varepsilon$, то $t(k, \varepsilon) = k$. Если существуют $a \in T$, $y \in T^*$, такие, что $x = ay$, то переходную функцию t можно определить рекурсивно с помощью соотношения $t(k, x) = t(t(k, a), y)$.

Вернемся к примеру и запишем

$$\begin{aligned} t(A, aba) &= t(t(A, a), ba) = \\ &= t(B, ba) = \\ &= t(t(B, b), a) = \\ &= t(B, a) = D. \end{aligned}$$

Теперь множество строк $T(M) \subset T^*$ принимаемых автоматом $M = (K, T, t, k_1, F)$ может быть описано следующим образом:

$$T(M) = \{x \in T^* \mid t(k_1, x) \in F\}.$$

Если вас беспокоит, что переходная функция t детерминированного конечного автомата — неполная, представьте себе, что рассматриваемый нами автомат имеет еще одно фиктивное состояние, которому соответствует узел графа, помеченный символом Δ, и это состояние является значением переходной функции на всех тех элементах множества $(K \times T)$, на которых она считается неопределенной. Таким образом, для всех пар $(K \times a)$, на которых функция t не определена, она принимает значение Δ, т. е. $t(k, a) = \Delta$, кроме

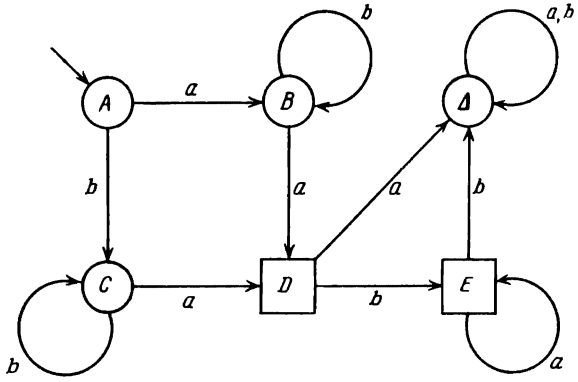


Рис. 3.3

того, очевидно, $t(\Delta, a) = \Delta$ для всех $a \in T$. Воспользуемся этим рассуждением в приведенном выше примере (см. рис. 3.3). (Дуга от узла Δ до узла Δ, помеченная двумя символами a и b – это стандартное сокращенное обозначение двух совпадающих дуг, одна из которых была помечена символом a, а другая символом b.) Если что-нибудь и может действительно внушить беспокойство, так это ограничение, определяющее конечный автомат как детерминированный конечный автомат. Избавимся от этого ограничения и позволим узлам графа иметь по нескольку исходящих из них дуг, помеченных одним символом. Тогда переходная функция t на таком элементе множества $K \times T$ будет иметь множество значений вместо одного. Если же для некоторого из них в множестве $K \times T$ не существует соответствующей пары (k, a), то $t(k, a) \in \emptyset$, а так как $\emptyset \in 2^K$, то $t : K \times T \to 2^K$ – всегда полная функция.

Недетерминированный конечный автомат – это совокупность пяти объектов: $M = (K, T, t, k_1, F)$, где

а) K – конечное множество состояний;

б) T – конечный входной алфавит;

в) t – полная функция, отображающая $K \times T \to 2^K$, называемая также переходной функцией;

г) $k_1 \in K$ – начальное состояние;

д) $F \subset K$ – множество конечных состояний.

Таблица 3.2

K	T	2^K
S	a	$\{A\}$
S	b	$\{B\}$
A	a	$\{A, \#\}$
A	b	$\emptyset$
B	a	$\emptyset$
B	b	$\{B, \#\}$
$\#$	a	$\emptyset$
$\#$	b	$\emptyset$

Конечный автомат, приведенный на рис. 3.1, является недетерминированным, переходная функция которого задана табл. 3.2.

Перед тем как определить множество строк, принимаемых данным автоматом, необходимо вновь расширить определение переходной функции. Определим переходную функцию на множестве $2^K \times T$ следующим образом:

$$t(K', a) = \bigcup_{k \in K'} t(k, a) \quad \text{для } K' \subset K,\ a \in T.$$

Таким образом, множество узлов конечного автомата, соединенных с некоторым узлом из множества K' дугами, помеченными символом a, в точности совпадает с множеством $t(K', a)$. Рассмотрим теперь определение функции t на области $2^K \times T^*$, определив ее область значений как 2^K. Воспользуемся для этого тем же приемом, что и в случае детерминированного конечного автомата. Если $K' \subset K$, то $t(K', \varepsilon) = K'$, и если $x = ay, a \in T^*$, $y \in T^*$, то $t(K', x) = t(t(K', a), y)$.

Используем для иллюстрации недетерминированный конечный автомат, приведенный на рис. 3.1, получим

$$\begin{aligned} t(\{S, A\}, ab) &= t(t(\{S, A\}, a), b) = \\ &= t(t(S, a) \cup t(A, a), b) = \\ &= t(\{A, \#\}, b) = \\ &= t(A, b) \cup t(\#, b) = \varnothing. \end{aligned}$$

Определим теперь множество строк $T(M)$, принимаемых недетерминированным конечным автоматом $M = (K, T, t, k_1, F)$ как множество таких строк, каждый элемент которых совпадает с одним из символов, помечающих дуги автоматов, а строка в целом соответствует пути по графу от начального его узла до конечного, т. е.

$$T(M) = \{x \in T^* \mid t(\{k_1\}, x) \cap F \neq \varnothing\}.$$

ТЕОРЕМА 3.2

Множество строк L может быть принято детерминированным конечным автоматом тогда и только тогда, когда множество строк L может быть принято недетерминированным конечным автоматом.

Доказательство

а) Для любого детерминированного конечного автомата существует эквивалентный ему детерминированный конечный автомат, переходная функция которого является полной (было показано, что для этого достаточно фиктивно доопределить переходную функцию t). Пусть $L \subset T^*$ такое, что $L = T(M)$ для некоторого детерминированного конечного автомата $M = (K, T, t, k_1, F)$, где t – полная функция. Рассмотрим такой недетерминированный конечный автомат $M' = (K, T, t, k_1, F)$, что $t'(k, a) = \{t(k, a)\}$ (это следует из того, что $T(M) = T(M')$).

б) Пусть $M = (K, T, t, k_1, F)$ – некоторый недетерминированный конечный автомат, тогда определим такой детерминированный конечный автомат, что множество принимаемых им строк совпадает с множеством $T(M)$. Пусть множество состояний такого детерминированного автомата совпадает с множеством 2^K, входной алфавит совпадает с множеством T, начальное состояние совпадает с множеством $\{k_1\}$, а переходная функция определена как расширение функции t с областью определения $2^K \times T$ и с областью значений 2^K. По определению $x \in T(M) \Leftrightarrow t(\{k_1\}, x) \cap F \neq \varnothing$, и если в качестве множества конечных состояний детерминированного конечного автома-

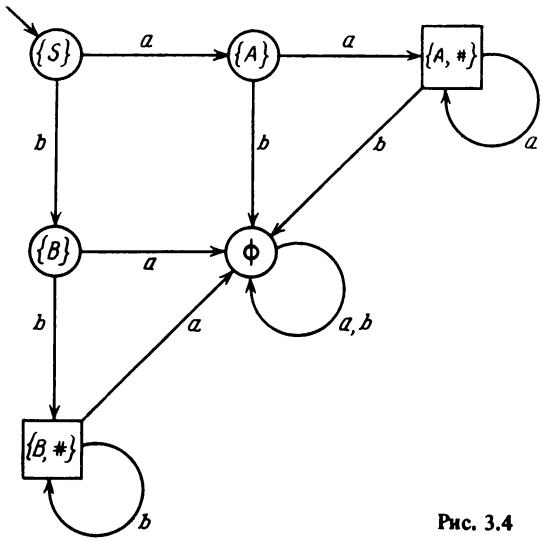


Рис. 3.4

та выбрать такое множество $K' \subset K$, что $K' \cap F \neq \emptyset$, то детерминированный конечный автомат будет определен и исходное утверждение доказано.

На практике задача построения детерминированного конечного автомата, эквивалентного некоторому недетерминированному конечному автомату, встречается довольно часто. Ограничимся рассмотрением множества тех состояний, которые могут быть получены из начального состояния $\{k_1\}$. Таким образом, вместо того, чтобы рассматривать все $2^{*(K)}$ возможных состояний автомата, ограничимся рассмотрением всех состояний $t(\{k_1\}, a)$ для всех $a \in T$. Далее рассмотрим множество состояний $t(K', a)$ для всех $a \in T$, $K' \subset K$. Этот процесс будем продолжать до тех пор, пока полученные таким образом состояния автомата не начнут совпадать с уже существующими. Поскольку общее число различных состояний автомата не может превышать $2^{*(K)}$, то множество получаемых состояний ограничено. На рис. 3.4 представлен детерминированный конечный автомат, построенный описанным выше способом из недетерминированного конечного автомата, показанного на рис. 3.1. Отметим в заключение, что пустое множество играет в этом случае роль фиктивного состояния, и в действительности все проведенные к соответствующему узлу дуги могут быть опущены.

Перейдем теперь к основной теореме данной главы.

ТЕОРЕМА 3.3

Следующие три утверждения равносильны:

1) L – может быть принято некоторым недетерминированным конечным автоматом;

2) L – может быть принято некоторым детерминированным конечным автоматом;

3) L – порождено некоторой регулярной грамматикой.

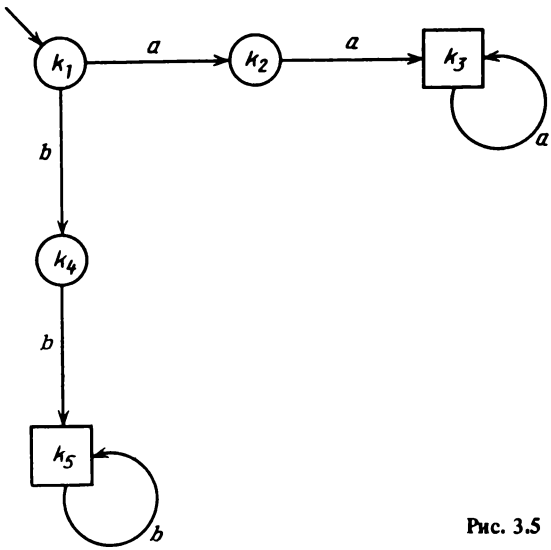


Рис. 3.5

Доказательство

Равносильность двух первых утверждений была доказана в теореме 3.2. Теперь аналогично докажем, что из справедливости утв. 2 следует справедливость утв. 3, а из справедливости утв. 3 следует справедливость утв. 1. Приведем здесь лишь схему доказательства, а все детали и формулировки предоставим читателю.

Итак, пусть L – принято детерминированным конечным автоматом $M = (K, T, t, k_1, F)$. Без потери общности можно считать, что $K \cap T \neq 0$. Определим теперь регулярную грамматику G так, чтобы $L(G) = T(M) \setminus \{\varepsilon\}$. Если $\varepsilon \in T(M)$, то $k_1 \in F$ и можно определить регулярную грамматику G' так, чтобы $L(G') = L(G) \cup \{\varepsilon\} = T(M)$. Грамматика, которую надо определить, как обычно, имеет вид $G = (N, T, P, S)$, где $N = K$, $S = k_1$, и множество P содержит все продукции вида $k_i \to ak_j$, где $t(k_i, a) = k_j$, а также все продукции вида $k_i \to a$, где $t(k_i, a) = k_j$ и $k_j \in F$. Например, переобозначим узлы детерминированного конечного автомата, показанного на рис. 3.4, не изображая фиктивное состояние детерминированного конечного автомата, как это показано на рис. 3.5; грамматика, ассоциированная с таким автоматом, может выглядеть следующим образом:

$N = \{k_1, k_2, k_3, k_4, k_5\}$, $S = k_1$,

а продукции ее имеют вид

$k_1 \to ak_2 \mid bk_4$,

$k_2 \to ak_3 \mid a$,

$k_3 \to ak_3 \mid a$,

$k_4 \to bk_5 \mid b$,

$k_5 \to bk_5 \mid b$.

Предоставляем читателю самостоятельно доказать, что $x \in L(G) \Leftrightarrow \Leftrightarrow x \in T(M) \setminus \{\varepsilon\}$.

Продолжим доказательство равносильности: из справедливости утв. 3 следует справедливость утв. 1.

Если $L = L(G)$ — язык, порожденный грамматикой $G = (N, T, P, S)$, тогда L — множество строк, принятых недетерминированным конечным автоматом вида $(N \cup \{\#\}, T, t, S\{\#\})$, где $\#$ — некоторый новый символ и $\# \notin N$, а переходная функция t определена следующим образом: если продукция $A \to a$ — элемент множества P, то множество $t(A, a)$ содержит элемент $\#$, если продукция $A \to aB$ — элемент множества P, то множество $t(A, a)$ содержит элемент B. Понятно, что при таком определении переходной функции t любой продукции из P соответствует некоторое подмножество множества $t(A, a)$.

Утверждение, что для любого регулярного языка L существует детерминированный конечный автомат M, такой, что $T(M) = L$, можно использовать как эффективный способ определения, является ли произвольная строка x элементом множества L или нет. Для этого достаточно построить диаграмму автомата M и определить по ней, существует ли такой путь от его начального узла до конечного, что упорядоченная последовательность меток узлов и дуг, полученная при построении такого пути, совпадает со строкой x. Поскольку M — детерминированный конечный автомат, то существует не более одного такого пути. Если же воспользоваться определенным выше фиктивным состоянием автомата, то для любого $x \in T^*$ всегда существует в точности один путь по графу от начального узла и остается лишь выяснить, проходит ли этот путь через конечный узел графа.

Итак, показав, что регулярные языки совпадают с множеством строк, принимаемых детерминированными и недетерминированными конечными автоматами, мы получили инструмент для доказательства ряда интересных теорем.

ТЕОРЕМА 3.4

Если L — регулярный язык, порожденный грамматикой (N, T, P, S), то дополнение множества L в множестве $T^* - \overline{L} = T^* \setminus L$ — тоже регулярный язык.

Доказательство

Пусть $M = (K, T, t, k_1, F)$ — детерминированный конечный автомат, а L — множество принятых им строк. Определим фиктивное состояние автомата так, как выше для того, чтобы функция t стала полной. Если $x \in L$, то существует путь по направленному графу, такой, что упорядоченная последовательность меток входящих в него узлов и дуг совпадает со строкой x и соединяет начальный узел графа с его конечным узлом. В то же время если $x \notin L$, то путь, построенный в соответствии со строкой x, начнется в узле k_1, а кончится не в конечном узле. Таким образом, $\overline{L}$ — множество строк, принятых автоматом $\overline{M} = (K, T, t, k_1, K \setminus F)$, и следовательно, должно быть регулярным языком.

ТЕОРЕМА 3.5

Если L_1 и L_2 – регулярные языки, то множество $L_1 \cap L_2$ – регулярный язык.

Доказательство

$\overline{L}_1$ и $\overline{L}_2$ – регулярные языки по теореме 3.4. Таким образом, по теореме 3.1, *б* множество $\overline{L}_1 \cup \overline{L}_2$ – регулярный язык, но $L_1 \cap L_2 = \overline{\overline{L}_1 \cup \overline{L}_2}$ – регулярный язык из теоремы 3.4, и утверждение доказано.

КОНЕЧНЫЙ АВТОМАТ С ε-ПЕРЕХОДАМИ

Говорят, что недетерминированный конечный автомат $M = (K, T, t, k_1, F)$ имеет ε-переходы, если функция t определена на множестве $K \times (T \cup \{\varepsilon\})$, т.е. $t : K \times (T \cup \{\varepsilon\}) \to 2^K$. Пример такого автомата приведен на рис. 3.6.

Если $M = (K, T, t, k_1, F)$ – недетерминированный конечный автомат, имеющий *ε-переходы* и $k, k' \in K$, такое, что $k' \in t(k, \varepsilon)$, то говорят, что k' – ε-достижимо из узла k и записывают это

$$k \xrightarrow[\varepsilon]{} k'.$$

В таком случае недетерминированный конечный автомат M может перейти из состояния k в состояние k' без обработки какой-либо дополнительной информации. Пусть $\overset{*}{\underset{\varepsilon}{}}$ – рефлексивное и транзитивное замыкание множества $\overline{\varepsilon}$. Тогда $k \overset{*}{\underset{\varepsilon}{\to}} k' \iff k = k'$, т.е. существует путь по графу от узла k до k', состоящий из дуг, помеченных символом ε и длиной $\geqslant 1$. Если $k \in K$, то множество узлов, достижимых из узла k без обработки дополнительной информации, обозначается $R(k)$, что формально записывается как

$$R(k) = \{k' \mid k \overset{*}{\underset{\varepsilon}{\to}} k'\}.$$

Расширим это определение следующим образом: пусть $K' \subset K$, тогда

$$R(K') = \bigcup_{k \in K'} R(k).$$

В рассмотренном выше примере $R(A) = \{A, B, C\}$, $R(B) = \{B\}$, $R(C) = \{C\}$, $R(D) = \{D\}$. $R(\{A, B\}) = R(A) \cup R(B) = \{A, B, C\}$.

Тогда, по определению, $R(\{A, B\}) = R(A) \cup R(B) = \{A, B, C\}, \ldots$ Переопределим теперь переходную функцию t, определив ее на ε-переходах как $\hat{t}: K \times (T \cup \{\varepsilon\}) \to 2^K$ таким образом, что если $k \in R$, то $\hat{t}(k, a)$ – множество состояний (узлов), которые могут быть достигнуты из узла (состояния) k после получения и обработки дополнительной информации в виде символа a. Например, если автомат находится в состоянии A, то $t(A, a) = \{B\}$.

Однако, используя лишь один раз введенный символ a, можно перевести автомат из состояния A в состояние D, из состояния A в состояние A и из состояния A в состояние C. Чтобы перевести автомат в состояние D, например, необходимо сначала достичь состояния B, используя ε-переход, а затем, введя символ a, достичь состояния D; чтобы достичь состояния A, необходимо сначала достичь состояния C, используя ε-переход, а затем, введя символ a, достичь состояния A.

Итак, мы определили переходную функцию

$$t: K \times (T \cup \{\varepsilon\}) \to 2^K$$

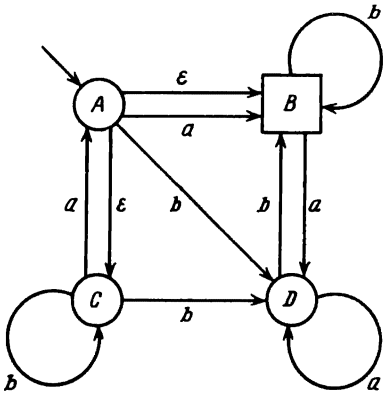


Рис. 3.6

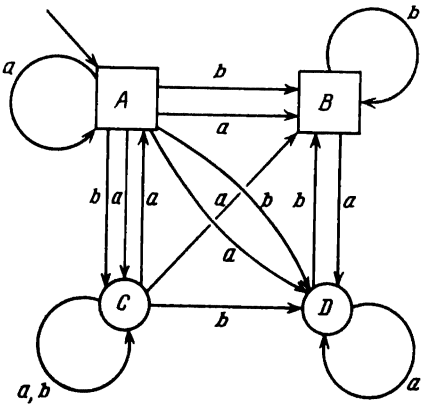


Рис. 3.7

как функцию

$$\hat{t}(k, \varepsilon) = R(k)$$

и расширили ее следующим образом:

$$\hat{t}(k, a) = \bigcup_{k' \in R(k)} R(t(k', a)) \quad \text{для всех} \quad k \in K, a \in T.$$

Для рассмотренного выше примера верны следующие утверждения:

$$\hat{t}(A, \varepsilon) = R(A) = \{A, B, C\},$$

$$\begin{aligned}\hat{t}(A, a) &= \bigcup_{k' \in \{A,B,C\}} R(t(k', a)) = \\ &= R(\{B\}) \cup R(\{D\}) \cup R(\{A\}) = \\ &= \{A, B, C, D\},\end{aligned}$$

$$\begin{aligned}\hat{t}(A, b) &= \bigcup_{k' \in \{A,B,C\}} R(t(k', b)) = \\ &= R(\{D\}) \cup R(\{B\}) \cup R(\{C, D\}) = \\ &= \{B, C, D\} \qquad \text{и т.д.}\end{aligned}$$

Расширим теперь функцию $\hat{t}$ следующим образом: $\hat{t} : 2^K \times (T \cup \{\varepsilon\}) \to 2^K$. Для этого, как было уже сделано ранее, возьмем $K' \subset K$ и $a \in T$ и запишем

$$\hat{t}(K', \varepsilon) = R(K'),$$

$$\hat{t}(K', a) = \bigcup_{k \in K'} \hat{t}(k, a).$$

Определим теперь функцию $t : 2^K \times T^* \to 2^K$ следующим образом:

$$\hat{t}(K', ax) = \hat{t}(\hat{t}(K', a), x), \text{ для всех } K' \subset K, a \in T, x \in T^*.$$

Множество строк, принятых недетерминированным конечным автоматом, имеющим ε-переходы $M = (K, T, t, k_1, F)$, формально определяется как

$$T(M) = \{x \in T^* \mid \hat{t}(\{k_1\}, x) \cap F \neq \varnothing\}.$$

Если M — такой конечный автомат, то можно определить недетерминированный конечный автомат $M' = (K, T, t', k_1, F')$, где $K'(k, a) = \hat{t}(k, a)$ и $F' = F$ при $R(k_1) \cap F \neq \Phi$ и $F' = F \cup \{k\}$, в противном случае недетерминированный конечный автомат M не имеет ε-переходов и нетрудно показать, что $T(M) = T(M')$.

ТЕОРЕМА 3.6

Если L — множество строк, принятых некоторым имеющим ε-переходы недетерминированным конечным автоматом, тогда L — регулярный язык.

На рис. 3.7 приведен недетерминированный конечный автомат, эквивалентный автомату, приведенному на рис. 3.6.

УПРАЖНЕНИЯ

3.1. Пусть множество строк L — подмножество множества $\{0,1\}^*$ таких, что справа от символа 0 всегда находится символ 1.

а) Определите детерминированный конечный автомат, которым могут быть приняты все строки из L.

б) Определите регулярную грамматику, порождающую язык L.

3.2. Пусть L — регулярное множество. Пусть $\mathrm{Pref}(L) = \{x | x$ — префикс строки из $L\}$. Докажите, что $\mathrm{Pref}(L)$ — также регулярное множество. (Воспользуйтесь тем, что $L = T(M)$ для некоторого детерминированного автомата M).

3.3. Докажите теорему 3.1, *б*.

3.4. Пусть $M = (\{k_1, k_2, k_3\}, \{a, b\}, t, k_1, \{k_3\})$ — недетерминированный конечный автомат, где $t(k_1, a) = \{k_2, k_3\}$, $t(k_2, a) = \{k_1, k_2\}$, $t(k_3, a) = \{k_1 k_3\}$, $t\{k_1, b\} = \{k_1\}$, $t\{k_2, b\} = \Phi$, $t(k_3, b) = \{k_1, k_2\}$. Определите недетерминированный конечный автомат так, чтобы все строки из множества $T(M)$ были бы им приняты.

3.5. Постройте детерминированный конечный автомат, эквивалентный недетерминированному конечному автомату, изображенному на рис. 3.6.

3.6. *Гомоморфизм* $\theta: T_1^* \to T_2^*$ есть полная функция, такая, что $\theta(\varepsilon) = \varepsilon$ и $\theta(x, y) = \theta(x)\theta(y)$ для любого $x, y \in T_1^*$. Пусть $T_1 = \{a, b\}$, $T_2 = \{b, c\}$ и $\theta(a) = bc$, $\theta(b) = bb$; определите значения $\theta(a, b)$ и $\theta(b, a)$. Покажите, что если L — регулярный язык и θ — произвольный гомоморфизм, то $\theta(L)$ — регулярный язык.

3.7. Докажите теорему 3.6.

3.8. Повторите доказательство теоремы 3.1, *а*, используя теорему 3.6.

3.9. Опишите конечный автомат, который будет принимать любое английское слово, начинающееся с "un" и кончающееся на "d". Напишите на языке программирования Паскаль программу подсчета числа таких слов в произвольном текстовом файле.

3.10. Пусть каждая продукция грамматики $G = (N, T, P, S)$ имеет вид либо $A \to Ba$, либо $A \to a$, $A, B \in N$, $a \in T$. Покажите, что существует недетерминированный конечный автомат, который способен принять все строки из множества $L(G)$. (Продукция $A \to Ba$ соответствует дуге, помеченной символом a и идущей от узла B к узлу A.)

3.11. Если каждая продукция грамматики $G = (N, T, P, S)$ имеет вид $A \to xB$ или $A \to x$, $A, B \in N$, $x \in T^*$, то такая грамматика называется *правосторонней грамматикой.* Аналогично, если каждая продукция грамматики G имеет вид $A \to Bx$ или вид $A \to x$ $A, B \in N$, $x \in T^*$, то такая грамматика называется *левосторонней грамматикой.* Покажите, что $L \in T^*$ порожден правосторонней грамматикой $\Leftrightarrow L$ — регулярный язык $\Leftrightarrow L$ порожден левосторонней грамматикой.

ГЛАВА 4

РЕГУЛЯРНЫЕ ЯЗЫКИ, II

"На рисунке не должно быть ненужных линий – у машины не должно быть ненужных частей".

Вильям Струнк "Элементы стиля"

РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ

Регулярные выражения – самый точный способ описания регулярных языков. Набор правил, определяющих регулярное выражение над множеством T, выглядит следующим образом:

I. 0 – соответствует пустому множеству;
   1 – соответствует множеству $\{\varepsilon\}$.

II. Если $L = \{x_1, x_2, \ldots, x_n\}$ – конечное множество строк, где $x_i \in T^*$, $i = 1, 2, \ldots, n$, то множеству L ставится в соответствие выражение $(x_1 + x_2 + \ldots + x_n)$.

III. Если r_1 – регулярное выражение, соответствующее регулярному языку L_1, и r_2 – регулярное выражение, соответствующее регулярному языку L_2, то выражение $(r_1 + r_2)$ соответствует множеству $L_1 \cup L_2$;
   выражение $(r_1 + r_2)$ соответствует множеству $L_1 \cup L_2$;
   выражение $(r_1 \cdot r_2)$ соответствует множеству $L_1 L_2$;
   выражение (r_1^k) соответствует множеству L_1^k $(k \geqslant 0)$;
   выражение (r_1^*) соответствует множеству L_1^*.

IV. Любое выражение, не удовлетворяющее условиям I–III, не является регулярным выражением.

Скобки в регулярном выражении можно опустить, при этом сохраняется общепринятое соглашение о старшинстве операций – самыми старшими считаются унарные операции, затем бинарная операция $\cdot$, и наконец, бинарная операция $+$.

Поскольку каждое конечное множество строк – регулярный язык, а всякий регулярный язык есть множество, замкнутое относительно операций конкатенации и замыкания Клини, то каждое регулярное выражение над множеством T соответствует некоторому регулярному языку из множества T^*. Например, регулярное выражение $(a + b)^* ba(ba)^*$ соответствует регулярному языку $\{a, b\}^* \{ba\} \{ba\}^* = \{a, b\}^* \{ba\}^+$ и, следовательно, регулярное выражение $(a + ba^*b)^* + b$ соответствует регулярному языку $(\{a\} \cup \{b\}\{a\}^*\{b\})^* \cup \{b\}$. Обычно определить недетерминированный конечный автомат, который принял бы регулярный язык, соответствующий заданному регулярному выражению, очень просто. Однако столь же просто при этом допустить ошибку. Некоторые из методов построения такого определения показаны на рис. 4.1, *а* – 4.1, *г*. В данных примерах и далее R обозначает регулярный язык, которому соответствует регулярное выражение r

Вы, должно быть, заметили, что в двух из приведенных примеров, мы

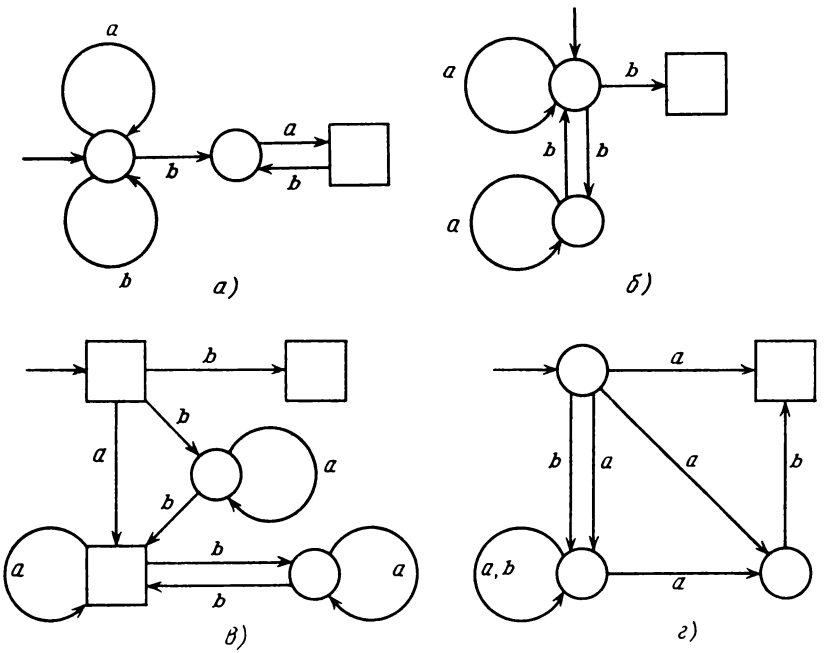


Рис. 4.1

упростили регулярные выражения, чтобы избежать определенных трудностей. Сформулированная ниже теорема описывает алгебру регулярных выражений, при этом положения (с) и (т) являются производными от предыдущих положений. Из этой теоремы, в частности, следует, что два регулярных выражения эквивалентны, если они соответствуют одному и тому же регулярному множеству, однако доказательство этого факта нетривиально. Это утверждение будет доказано ниже.

ТЕОРЕМА 4.1

Если Q, R, S – некоторые регулярные выражения над множеством T, то

а) $Q + Q = Q, Q + 0 = Q,$

б) $Q + R = R + Q,$

в) $(Q + R) + S = Q + (R + S) = Q + R + S,$

г) $(QR)S = Q(RS) = QRS,$

д) $Q1 = 1Q = Q$ и $Q0 = 0Q = 0,$

е) $(R + S)Q = RQ + SQ,$

ж) $Q(R + S) = QR + QS,$

з) $Q^*Q^* = Q^*,$

и) $(Q^*)^* = Q^*,$

к) $QQ^* = Q^*Q,$

л) $Q^* = 1 + Q + Q^2 + \cdots + Q^{k+1}Q^*$, для $k \geqslant 0$,

м) $1^* = 1$ и $0^* = 1$,

н) $(Q^* + R^*)^* = (Q^*R^*)^* = (Q + R)^*$,

о) $(RQ)^*R = R(QR)^*$,

п) $(Q^*R)^*Q^* = (Q + R)^*$,

р) $(Q^*R)^* = (Q + R)^*R + 1$ и $(QR^*)^* = Q(Q + R)^* + 1$,

с) если $Q = R^*S$, то $Q = RQ + S$;

т) если $1 \notin R$ и $Q = RQ + S$, то $Q = R^*S$.

Доказательство

Большинство из этих положений очевидны, для доказательства достаточно воспользоваться соответствием операций, операция + соответствует операции объединения множеств $\cup$, а операция $\cdot$ соответствует операции конкатенации множеств. Поэтому перейдем сразу к доказательству двух последних положений. Сначала перепишем их следующим образом:

с) $Q = R^*S = (RR^* + 1)S$ = (из б) и л))

$= RR^*S + S$ = (из д) и е))

$= RQ + S$.

т) $Q = RQ + S \Rightarrow Q = R(RQ + S) + S =$

$= R^2Q + RS + S =$

$= R^2(RQ + S) + RS + S =$

$= R^3Q + R^2S + RS + S =$

$= R^{k+1}Q + (R^k + \cdots + R + 1)S$ для любого $k \geqslant 0$.

Теперь найдется такое $k \geqslant 0$, что $|x| \leqslant k$ для любого $x \in Q$. Поскольку $1 \notin R$, то $x \notin R^{k+1}Q$ и, следовательно, $x \in (R^k + \ldots + R + 1)S$ т.е. $x \in R^*S^*$. Таким образом, если $x \in Q$, то $x \in R^*S$. Сравнив эти результаты, получим $Q = R^*S$. Итак, мы выяснили, что любое регулярное выражение соответствует некоторому регулярному языку. Обратное утверждение также верно и будет доказано в следующей теореме.

ТЕОРЕМА 4.2 (ТЕОРЕМА КЛИНИ)

Каждому регулярному языку из T^* соответствует регулярное выражение над множеством T.

Доказательство

Пусть L – регулярное выражение, тогда существует такой детерминированный конечный автомат $M = (K, T, t, k_1, F)$, что $L = T(M)$. Предположим, что $K = \{k_1, k_2, \ldots, k_n\}$. Если $F = \{k_{\lambda 1}, k_{\lambda 2}, \ldots, k_{\lambda m}\}$, тогда $L = L_1 \cup L_2 \cup \ldots \cup L_m$, где L_i – язык, принимаемый автоматом $M_i = (K, T, t, k_1, \{k_{\lambda i}\})$. Поскольку операции объединения по определению соответствует операция сложения, то достаточно показать, что каждому языку L_i соответствует регулярное выражение. Таким образом, достаточно показать, что множеству строк, принимаемых детерминированным конечным автоматом, имею-

щим одно начальное и одно конечное состояния, может быть поставлено в соответствие некоторое регулярное выражение.

Рассмотрим автомат $M = (K, T, t, k_1, F)$, состояния которого помечены следующим образом: $K = \{k_1, k_2, \ldots, k_n\}$; $F = \{k_n\}$.

Обозначим T_{ij}^{l} $(1 \leqslant i, j \leqslant n,\ 0 \leqslant l \leqslant n)$ язык, все элементы которого представляют собой такие строки, что $t(k_i, x) = k_j$ и для всех собственных префиксов y строки x верно $t(k_i, y) \in \{k_m \mid m \leqslant l\}$ (собственным *префиксом строки* $x \in T^*$ называют такую строку $y \in T^+$, что $x = yv$, где $v \in T^+$ – некоторая строка, y – непустая строка, не совпадающая со строкой x). Воспользуемся методом индукции и покажем, что языку T_{ij}^{0} всегда соответствует некоторое регулярное выражение.

Если $i \neq j$, то языку T_{ij}^{0} соответствует регулярное выражение $a_1 + a_2 + \ldots + a_r$, где $a_1, a_2, \ldots, a_r$ – символы, которыми помечены дуги, соединяющие узлы k_i и k_j автомата между собой; если же таких дуг нет, то языку T_{ij}^{0} соответствует 0. Если $i = j$, то языку T_{ij}^{0} соответствует регулярное выражение $1 + a_1 + a_2 + \ldots + a_r$, где $a_1, a_2, \ldots, a_r$ – символы, которыми помечены дуги, соединяющие узел автомата k_i с самим собой; если же таких дуг нет, то языку T_{ij}^{0} соответствует 1. Таким образом, мы показали, что любому языку T_{ij}^{l} соответствует некоторое регулярное выражение, если $l = 0$. Далее, воспользовавшись соотношением $T_{ij}^{l} = T_{ij}^{l-1} + T_{il}^{l-1}(T_{ll}^{l-1})^* T_{lj}^{l-1}$, $1 \leqslant l \leqslant n$, можно показать, что каждому языку T_{ij}^{l} соответствует некоторое регулярное выражение, поэтому $T(M) = T_{1n}^{n}$ и теорема доказана.

Если это соотношение вам неочевидно, рассмотрите направленный граф недетерминированного конечного автомата M и любой путь x по этому графу. Пусть это будет путь от узла, помеченного символом k_i, до узла, помеченного символом k_j, и проходит через все узлы, помеченные символами из множества $\{k_1, k_2, \ldots, k_l\}$. Кроме того, этот путь должен либо вовсе не проходить через узел, помеченный символом k_l, и тогда, очевидно, $x \in T_{ij}^{l-1}$, либо проходить через него m раз $(m \geqslant 1)$. Разобьем путь от узла, помеченного символом k_i, до узла, помеченного символом k_j, на следующие участки: "0" – от узла, помеченного символом k_i, до узла, помеченного символом k_l, при первом прохождении через узел k_l; "1" – от узла k_l до узла k_l при втором прохождении через узел k_l, ..., "$m-1$" – от узла k_l до узла k_l при m-м прохождении через узел k_l; "m" – от узла k_l до узла, помеченного символом k_j. Теперь можно записать, что $x = x_0 x_1 \ldots x_m$, где x_i – символ, помечающий i-й участок пути. Так как $x_0 \in T_{il}^{l-1}, x_1, \ldots, x_{m-1} \in T_{ll}^{l-1}, x_m \in T_{lj}^{l-1}$, следовательно, $x \in T_{il}^{l-1}(T_{ll}^{l-1})^* T_{lj}^{l-1}$. Легко видеть, что любая строка из этого множества строк также является элементом множества T_{ij}^{l}, что и доказывает эквивалентность этих строк.

МИНИМИЗАЦИЯ

При определении детерминированного конечного автомата, который должен принимать язык L, вообще говоря, выгодно, чтобы этот автомат имел как можно меньше состояний. Такой детерминированный конечный автомат называют *минимальным детерминированным конечным автоматом*. В этом

разделе будет показано, как строится минимальный автомат и, более того, будет доказана его единственность.

В гл. 1 было введено понятие отношения эквивалентности на множестве A, которое разбивало последнее на несколько непересекающихся классов эквивалентности и при этом класс эквивалентности, содержащий элемент $a \in A$ обозначают $\bar{a}$. *Индексом отношения эквивалентности* называют число таких классов.

Рассмотрим теперь более подробно отношение эквивалентности, определенное на множестве T^*. Отношение эквивалентности R называют правым инвариантом, если

из xRy для любых $z \in T^*$ следует $xzRyz$.

Пусть L — некоторый язык, $L \subset T^*$, определим R_L – *отношение, ассоциированное* с L, следующим образом:

$xR_Ly \Leftrightarrow$ для любого $z \in T^*, xz \in L,$ если $yz \in L$.

Поскольку R_L – рефлексивно, симметрично и транзитивно, оно является отношением эквивалентности.

Если $M = (K, T, t, k_1, F)$ – детерминированный конечный автомат, то можно таким же образом определить и отношение R_M, ассоциированное с автоматом M, т. е.

$$xR_My \Leftrightarrow t(k_1, x) = t(k_1, y).$$

Иначе говоря, две строки x и y удовлетворяют отношению R_M тогда и только тогда, когда соответствующие им пути по графу соединяют начальный его узел с одним и тем же узлом. Как и в предыдущем случае, отношение R_M является отношением эквивалентности и, более того, оно является правым инвариантом.

Предположим для простоты, что рассматриваемый детерминированный конечный автомат имеет полную переходную функцию, и ограничимся рассмотрением только такого случая. Тогда отношение R_M разбивает множество T^* на несколько непересекающихся классов эквивалентности. Любой класс эквивалентности содержит строки, каждая из которых соответствует пути по графу от начального его узла к некоторому заранее заданному узлу. Таким образом, каждый класс эквивалентности естественно ассоциирован с некоторым состоянием конечного автомата – элементом множества K. Поскольку рассматриваемый конечный автомат имеет $\#(K)$ состояний, то существует не более $\#(K)$ классов эквивалентности и, следовательно, R_M имеет конечный индекс.

Поскольку R_M – правый инвариант, то из xR_My следует, что для всех $z \in T^*$ справедливо xzR_Myz. Пусть L – язык, принимаемый автоматом M, т. е. $L = T(M)$. Тогда $xzR_Myz \Rightarrow t(k_1, xz) = t(k_1, yz) \Rightarrow t(k_1, xz) \in F \Leftrightarrow t(k_1, yz) \in F \Rightarrow xz \in L \Leftrightarrow yz \in L$. Итак, мы показали, что из xR_My следует xR_Ly и, следовательно, любой класс эквивалентности, определяемый отношением R_M, является подмножеством класса эквивалентности, определяемого соответствующим отношением R_L Поскольку отношение R_M разбивает множество T^* на конечное число классов эквивалентности, то индекс отношения

эквивалентности R_L конечен. Каждый класс эквивалентности, определяемый отношением R_L, включает один или несколько классов эквивалентности, определяемых отношением R_M, и следовательно, каждому классу эквивалентности, определяемому отношением R_L, можно поставить в соответствие некоторое подмножество множества состояний конечного автомата M. Два любых подмножества множества состояний автомата M, ассоциированные с двумя несовпадающими классами эквивалентности, определяемыми отношением R_L, не пересекаются. Если k и k' – два различных состояния из множества состояний K, которые являются элементами одного подмножества, ассоциированного с некоторым классом эквивалентности, определяемым отношением R_L, то для всех $z \in T^*$, $t(k, z) \in F \Leftrightarrow t(k', z) \in F$. В таком случае состояния k и k' конечного автомата называют *неразличимыми*.

Поскольку высказанное утверждение справедливо для любого детерминированного конечного автомата, имеющего полную переходную функцию и принимающего язык L, кажется очевидным, что любой автомат, удовлетворяющий этим требованиям, должен иметь не меньше, чем n состояний, где n – индекс отношения эквивалентности R_L. Более того, если $M = (K, T, t, k_1, F)$ и $M' = (K', T', t', k_1', F')$ – два детерминированных конечных автомата, каждый из которых имеет по n состояний и принимает язык L, то одно и только одно из состояний каждого автомата может быть ассоциировано с одним и тем же классом эквивалентности, определяемым отношением R_L. Таким образом, можно говорить о взаимно однозначном отображении множества K на множество K' $\theta : K \to K'$, для которого, как легко показать, выполняется соотношение $\theta(k) = \theta(k') \Rightarrow \theta(t(k, a)) = \theta(t'(k', a))$ для любого $a \in T$. Это соотношение и доказывает единственность детерминированного конечного автомата, имеющего n состояний и принимающего язык L. Теперь мы попытаемся доказать, что такой автомат всегда существует.

Допустим, что отношение эквивалентности R_L задает в множестве T^* классы эквивалентности $\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n$. Определим тогда детерминированный конечный автомат M_L, имеющий входной алфавит T, множество состояний $k = \{\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n\}$, начальное состояние $k_1 = \bar{\varepsilon}$, множество конечных состояний F, где $\bar{x} \in F \Leftrightarrow x \in L$. Переходную функцию t автомата определим следующим образом: $t(\bar{x}, a) = \overline{xa}$. Поскольку R_L – правый инвариант, то из xR_Ly следует, что xaR_Lya и, следовательно, приведенное определение детерминированного конечного автомата корректно. Итак,

$$x \in T(M) \Leftrightarrow t(\bar{\varepsilon}, \bar{x}) = \bar{x} \in F \Leftrightarrow x \in L$$

и, следовательно, $T(M) = L$. В результате всех этих рассуждений доказана следующая теорема.

ТЕОРЕМА 4.3 (ТЕОРЕМА МАЙХИЛЛА–НЕРОДА)

Для любого регулярного языка L существует детерминированный конечный автомат, который принимает язык L, имеет полную переходную функцию и конечное число состояний, равное индексу отношения эквивалентности R_L, ассоциированного с языком L.

Любой узел автомата, определенный в соответствии с требованиями

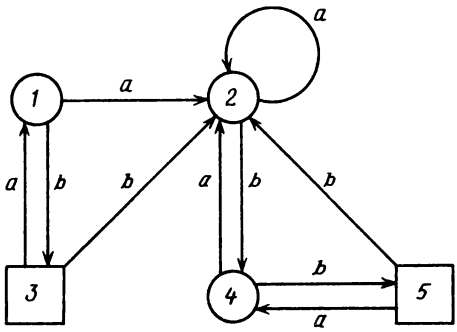


Рис. 4.2

теоремы Майхилла–Нерода и не лежащий на пути от начального узла к конечному, может быть исключен из множества состояний автомата вместе с дугами, которые соединяют его с другими узлами. Детерминированный конечный автомат, полученный в результате выполнения подобных операций, называют минимальным детерминированным конечным автоматом, принимающим язык L и обозначают M_L.

Пусть $M = (K, T, t, k_1, F)$ – некоторый детерминированный конечный автомат, принимающий язык L и имеющий полную переходную функцию. Опишем алгоритм построения минимального детерминированного конечного автомата M_L исходя только из языка L. Наша задача – разбить множество состояний автомата M на непересекающиеся подмножества неразличимых состояний. Определим отношения $D_0, D_1, \ldots$ на множестве K следующим образом: состояния k и k' различимы по строке длины 0, т. е. $kD_0k' \Leftrightarrow$ $\Leftrightarrow k \in F$ и $k' \notin F$, или $k \notin F$ и $k' \in F$.

Пусть $i > 0$, состояния k и k' различимы по строке длиной $\leqslant i$, т. е.

$$kD_i k' \Leftrightarrow kD_{i-1}k',$$

т.е. существует $a \in T$, такое, что $t(k, a) D_{i-1} t(k', a)$. Говорят, что состояние k различимо от состояния k', если существует такое $i \geqslant 0$, что $kD_i k'$, т. е.

$$kDk' \Leftrightarrow \text{существует } i \geqslant 0 | kD_i k'.$$

Очень просто можно показать, что $kD_ik' \Leftrightarrow$ существует такая строка x длины $\leqslant i$, что либо $t(k, x) \in F$ и $t(k', x) \notin F$, либо $t(k, x) \notin F$, а $t(k', x) \in F$. Итак, чтобы вычислить отношение D, необходимо, очевидно, последовательно вычислить отношения $D_0, D_1, \ldots$, и если в процессе этих вычислений, на некотором шаге r, окажется, что $D_{r+1} = D_r$, то это будет означать, что итерационный процесс окончен и $D = D_r$. Если $m = \#(K)$, то существует только $(m^2 - m)$ пар (k_i, k_j), где $i \neq j$. В худшем случае всякое D_{i+1} отличается от D_i двумя такими парами, и учитывая, что $D_0 \neq 0$, получим

$$D = D_r, \text{ где } r < (m^2 - m)/2.$$

Проиллюстрируем доказанное на примере детерминированного конечного автомата, изображенного на рис. 4.2. Представим отношение D_i в виде

булевой матрицы размером 5×5, у которой j, k-й элемент есть символ $T \Leftrightarrow j D_i k$. Поскольку отношение D_i симметрично и поскольку $k \not\mathbb{D}_i k$, для построения такой матрицы достаточно определить лишь те ее элементы, которые находятся над главной диагональю. Последовательность матриц, соответствующих отношениям D_0, D_1, D_2, ..., приведена в табл. 4.1. Например, $1 D_1 2$, поскольку $t(1, b) D_0 t(2, b)$ и $2 D_1 4$, поскольку $t(2, b) D_0 t(4, b)$. Отсюда видно, что пары состояний, помеченных символами 1 и 4, 3 и 5, неразличимы.

Таблица 4.1

D_0	D_1
FTFT .TFT .TF .T	TTFT .TTT .TF .T

Так как все неразличимые состояния детерминированного конечного автомата найдены, то определить соответствующий минимальный автомат очень просто. Известно, что состояния минимального конечного автомата есть множества неразличимых состояний. Пусть $K' \subset K$ – одно из таких множеств, определим переходную функцию автомата M_L следующим образом: $t_L(K', a) = K''$, где K'' – множество неразличимых состояний, содержащее состояние $t(k, a)$ для всех $k \in K'$. Тогда, очевидно, начальным состоянием автомата M_L является множество неразличимых состояний, содержащее начальное состояние автомата M, конечными состояниями автомата M_L являются те множества состояний автомата M, которые содержат конечные состояния автомата M. Вернемся к примеру. На рис. 4.3 приведен минимальный детерминированный конечный автомат. В этом случае граф не содержит узлов, которые не лежат на пути от начального узла до конечного.

Если исходный детерминированный конечный автомат имеет неполную переходную функцию, то, определив ее на некотором фиктивном состоянии Δ, мы определим новый конечный автомат, равносильный исходному, который имеет уже полную переходную функцию. Если в этом случае попытаться определить минимальный конечный автомат, то окажется, что ни один из путей от начального узла к любому конечному узлу не проходит через узел, соответствующий состоянию из множества состояний неразличимых от состояния Δ. Таким образом, ни состояние Δ, ни одно из неразличимых от него состояний, не попадет в множество состояний минимального конечного автомата.

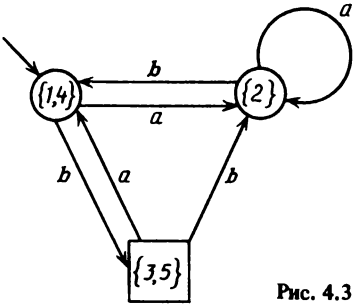


Рис. 4.3

АЛГОРИТМЫ ОПРЕДЕЛЕНИЯ РЕГУЛЯРНОСТИ ГРАММАТИКИ

Пусть L – произвольный регулярный язык, для которого определен такой минимальный детерминированный конечный автомат M_L, что $T(M_L) = L$. Предположим, что M_L имеет множество состояний $K = \{k_1, \ldots, k_n\}$, где k_1 – начальное состояние, и рассмотрим строку $z \in L$, имеющую длину $\geqslant n$. Для того чтобы строка z была принята автоматом M_L, достаточно, чтобы $t(k_1, z) \in F$. Поскольку $|z| \geqslant n$, должно существовать некоторое состояние $k \in K$ такое, что $z = wxy$, где $|x| \geqslant 1$ и $t(k_1, w) = k$, $t(k, x) = k$ и $t(k, y) \in F$. Но тогда $t(k_1, wx^i y) \in F$ для $i \geqslant 0$, а следовательно, $wx^i y \in L$ для $i \geqslant 0$ и, следовательно, доказана приведенная ниже теорема.

ТЕОРЕМА 4.4

Если L – регулярный язык и $z \in L$ имеет длину, большую или равную числу состояний минимального конечного автомата M_L, то $z = wxy$, где $|x| \geqslant 1$ и $wx^i y \in L$ для $i \geqslant 0$.

Из этой теоремы можно сделать важный вывод: если $L \neq \emptyset$, то, очевидно, существует строка $z \in L$, имеющая длину $< n$, где n – число состояний минимального детерминированного конечного автомата M_L. Тогда простой алгоритм[1] определения пустоты множества $L(G)$ может выглядеть следующим образом: определим минимальный детерминированный конечный автомат M_L для $L = L(G)$. Если M_L имеет n состояний, то $L = \emptyset \Leftrightarrow M_L$ не принимает ни одной строки, имеющей длину $< n$ из множества T^*. Поскольку число таких строк конечно, то выявить это можно за конечное время. Приведенное рассуждение доказывает следующую теорему.

ТЕОРЕМА 4.5

Проблема непустоты регулярных грамматик разрешима, т. е. если задана регулярная грамматика G, то существует алгоритм определения, пуст язык $L(G)$ или нет.

Сформулируем следующую теорему.

ТЕОРЕМА 4.6

а) Проблема равносильности регулярных грамматик разрешима, т. е. если G_1 и G_2 – две регулярные грамматики, то существует алгоритм определения справедливости равенства $L(G_1) = L(G_2)$.

б) Проблема конечности регулярных грамматик разрешима, т. е. если G – регулярная грамматика, то существует алгоритм определения конечности грамматики G.

Доказательство

а) $L(G_1) = L(G_2) \Leftrightarrow$ минимальные детерминированные конечные автоматы, соответствующие грамматикам G_1 и G_2, совпадают с точностью до символов, помечающих состояния автоматов.

---

[1] Алгоритмом решения некоторой проблемы π называют процедуру, с помощью которой, использовав некоторые данные о проблеме π в качестве входных, можно получить ее гарантированное решение.

б) Вернемся к минимальному конечному автомату $M_L = (K, T, t, k_1, F)$, определенному таким образом, чтобы он принимал язык $L = L(G)$. Если M_L имеет n состояний, то из теоремы 4.4 следует, что L – бесконечное множество $\Leftrightarrow$ существует строка $z \in L$, имеющая длину $\geqslant n$. Пусть такие строки существуют и пусть w – одна из них и $|w| \geqslant n$, если $|z| \geqslant n$, то $|z| \geqslant |w|$. Покажем, что $|w| < 2n$. Предположим, что это не так, тогда можно написать $w = w_1 w_2 w_3$, где $1 \leqslant |w_2| \leqslant n$ и $t(k_1, w_1) = k$, $t(k, w_2) = k$, $t(k, w_3) \in F$ для некоторого $k \in K$. Значит, $t(k_1, w_1 w_3) \in F$, $w_1 w_3 \in L$, но $|w_1 w_3| = |w| - |w_2| \geqslant 2n - n = n$ и $|w_1 w_3| < |w|$, следовательно, получаем противоречие. Таким образом, данный алгоритм позволяет определить минимальный детерминированный конечный автомат, который принимает язык $L = L(G)$.

Если этот минимальный детерминированный конечный автомат имеет n состояний, то необходимо выяснить, может ли он принять такую строку z, что $n \leqslant |z| < 2n$. Множество таких строк конечно и L бесконечное множество $\Leftrightarrow$ хотя бы одна из таких строк принимается автоматом M_L.

Внимательный читатель наверняка заметил, что доказательства теорем 4.5 и 4.6 в действительности не зависят от того, является конечный детерминированный автомат, принимающий строку z, минимальным или нет. Действительно, все эти рассуждения справедливы для любого детерминированного конечного автомата, имеющего n состояний. Минимальный детерминированный конечный автомат выгодно использовать потому, что число его состояний n – минимально возможное число, и следовательно, число строк, которые необходимо проверить, тоже минимально возможное число. И в заключение докажем следующую теорему.

ТЕОРЕМА 4.7

Проблемы непустоты, равносильности и конечности разрешимы для детерминированных (или недетерминированных) конечных автоматов.

УПРАЖНЕНИЯ

4.1. Определите детерминированный конечный автомат, который принимает строки

а) $a(ba + b)^* + b$;

б) $(ab + b^*)^* ba + b$;

в) $((b^*a)^* ab^*)^*$.

4.2. Найдите регулярное выражение, соответствующее множеству $T(M)$, где M – недетерминированный конечный автомат, определенный в упражнении 3.4.

4.3. Покажите, что если $R_1, \ldots, R_n$ – регулярные выражения над множеством T и $f(R_1, \ldots, R_n)$ – произвольная функция (в том числе функции вычисления суммы, произведения, дополнения), то

а) $f(R_1, \ldots, R_n) + (R_1 + R_2 + \ldots + R_n)^* = (R_1 + \ldots + R_n)^*$;

б) $(f(R_1^*, \ldots, R_n^*))^* = (R_1 + \ldots + R_n)^*$.

4.4. Воспользуйтесь теоремой 4.4 и покажите, что следующие языки – регулярные языки:

а) $\{a^n b^n \mid n \geqslant 0\}$;

б) $\{xx^r \mid x \in T^*\}$.

4.5. Определите детерминированный конечный автомат, принимающий язык L, которому соответствует регулярное выражение $(ab)^*+(a)(ba+a)^*$. Определите минимальный детерминированный конечный автомат, который принимает язык L. Воспользуйтесь этим автоматом для определения детерминированного конечного автомата, который принимает язык $\bar{L}=\{a, b\}^* \setminus L$.

4.6. Пусть R_M – отношение, ассоциированное с детерминированным конечным автоматом $M=(K, T, t, k_1, F)$ таким, что $T(M)=L$. Пусть $\bar{x}$, $\bar{y}$ – два несовпадающих класса эквивалентности отношения R_M. Покажите, что если $t(k_1, x)=k$ и $t(k_1, y)=k'$, то $xR_Ly \Leftrightarrow k$ и k' – неразличимы.

4.7. Вернитесь к решению упр. 4.1 и определите, какой из построенных вами детерминированных конечных автоматов, минимальный.

ГЛАВА 5

КОНТЕКСТНО СВОБОДНЫЕ ЯЗЫКИ

"Указать на форму – значило бы скрыть остальное"

Вальтер Скотт "Лорд островов"

Во второй главе было приведено два эквивалентных определения контекстно свободных грамматик. Сначала контекстно свободная грамматика была определена как контекстная грамматика (N, T, P, S), каждая продукция которой имеет вид $A \to \alpha$, $A \in N$, $\alpha \in (N \cup T)^*$. В этом случае контекстно свободная грамматика может иметь произвольное число ε-продукций. С другой стороны, если $\varepsilon \notin L(G)$, и все продукции контекстно свободной грамматики имеют вид $A \to \alpha$, $A \in N$, $\alpha \in (N \cup T)^+$, то такая грамматика, очевидно, не имеет ε-продукций. Снимем это последнее ограничение, пополнив набор продукций рассматриваемой грамматики продукцией $S \to \varepsilon$ при условии, что нетерминальный символ S не будет содержаться ни в одной из правых частей остальных продукций. В дальнейшем для упрощения доказательства некоторых теорем чаще будет использоваться второе определение контекстно свободной грамматики.

Если язык L порожден контекстно свободной грамматикой, то его называют контекстно свободным языком. Поскольку всякая регулярная грамматика является контекстно свободной, то всякий регулярный язык является контекстно свободным. Обратное неверно, т. е. существуют такие контекстно свободные языки, которые не являются регулярными языками. В качестве примера рассмотрим язык $\{a^n b^n | n \geqslant 1\}$. Он не является регулярным языком и порожден контекстно свободной грамматикой, имеющей продукции вида

$$S \to aSb | ab.$$

Пусть $G=(N, T, P, S)$ – контекстно свободная грамматика, предполо-

жим, что $L(G) \neq \emptyset$. Тогда любому $w \in T^+$ можно поставить в соответствие дерево вывода в грамматике G. Определим глубину этого дерева равной длине самого длинного пути по дереву от корня до узла, соответствующего указанному терминальному символу w. Например, пусть контекстно свободная грамматика G_1 имеет продукции вида

$S \to AB,$

$A \to aA \mid a,$

$B \to bB \mid b.$

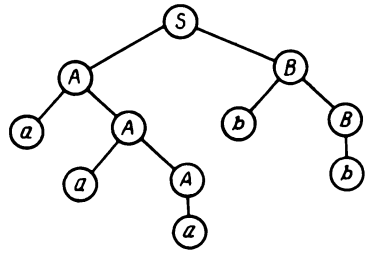


Рис. 5.1

На рис. 5.1 приведено дерево вывода для строки $a^3b^2 \in L(G)$, которое в данном случае имеет глубину 4.

Если мы говорим, что некоторое дерево вывода в контекстно свободной грамматике $G = (N, T, P, S)$ имеет глубину n, то это означает, что существует путь по графу, который проходит через узлы, помеченные символами A_1, $A_2, \ldots, A_n, a$, где $A_1, A_2, \ldots, A_n \in N, a \in T$. Если $n > \#(N)$, то два или более узлов, через которые проходит указанный путь по графу, помечены одними и теми же символами, например $A_i = A_j, i < j$ Построим новое дерево вывода следующим образом: заменим в уже построенном дереве вывода поддерево с корнем в узле, помеченном символом A_i на поддерево с корнем в узле, помеченном символом A_j. Пусть теперь $w \in L(G)$ и соответствующее ему дерево вывода имеет глубину $n > \#(N)$, тогда, воспользовавшись описанным выше методом, можно добиться, чтобы для некоторого $w' \in L(G)$ дерево вывода имело глубину $n \leqslant \#(N)$. Докажем следующую теорему.

ТЕОРЕМА 5.1.

Проблема непустоты разрешима для контекстно свободной грамматики. То есть, если $G = (N, T, P, S)$ – некоторая контекстно свободная грамматика, то существует алгоритм, позволяющий определить, справедливо или нет следующее утверждение: $L(G) = \emptyset$.

Доказательство

Функция алгоритма заключается в определении – имеет ли контекстно свободная грамматика продукцию вида $S \to \varepsilon$. Если продукция $S \to \varepsilon$ является элементом множества P, то поскольку $\varepsilon \in L(G)$, очевидно, что $L(G) \neq \emptyset$. Пусть продукция $S \to \varepsilon$ не является элементом множества P, тогда построим все возможные деревья вывода, имеющие глубину дерева, равную $\#(N)$. Если число таких деревьев конечно и ни одно из них не является деревом вывода строки терминальных символов, тогда и только тогда справедливо утверждение, что $L(G) = \emptyset$.

Продукция контекстно свободной грамматики $G = (N, T, P, S)$ тогда и только тогда называется *релевантной*, когда существует вывод некоторого $x \in L(G)$, в котором эта продукция используется, т. е. $A \to \alpha$ – релевант-

ная продукция $\Leftrightarrow$ существует $x \in L(G) | S \overset{*}{\Rightarrow} \alpha_1 A \alpha_2 \Rightarrow \alpha_1 \alpha \alpha_2 \overset{*}{\Rightarrow} x$. Всякая продукция контекстно свободной грамматики, которая не является релевантной продукцией, называется *иррелевантной* продукцией. Пусть теперь некоторое $A \in N$, поставим ему в соответствие некоторую грамматику $G_A = (N, T, P, A)$. Если $L(G_A) = \emptyset$, то вывести произвольную строку терминальных символов из нетерминального символа A с помощью продукций из множества P невозможно, т. е. не существует такой сентенциальной формы грамматики G, содержащей нетерминальный символ A, из которой можно было бы вывести некоторую строку терминальных символов. Таким образом, если удалить из множества P грамматики G все продукции, содержащие в левой или правой своей части нетерминальный символ A, то на языке $L(G)$ это никак не отразится. Помимо таких продукций грамматика G может, вообще говоря, иметь иррелевантные продукции, т. е. такие продукции, с помощью которых из некоторого $A \in N$ может быть выведена строка терминальных символов, которая не входит ни в одну сентенциальную форму G. Воспользовавшись теоремой 5.1, можно показать, что если существует сентенциальная форма, содержащая нетерминальный символ A, то она может быть выведена с помощью частичного вывода и соответствующее дерево вывода будет иметь глубину $< \#(N)$. Такое дерево называют *деревом частичного вывода*, так как его листья не помечены терминальными символами. Рассмотрев все такие деревья, можно выяснить, существует или нет сентенциальная форма, содержащая нетерминальный символ A. Если такой сентенциальной формы не существует, то все продукции, содержащие в левой и правой части нетерминальный символ A, могут быть исключены из рассмотрения как иррелевантные.

Далее будем предполагать, что любая контекстно свободная грамматика имеет лишь релевантные продукции.

НОРМАЛЬНАЯ ФОРМА ХОМСКОГО

Формат рассмотренных выше продукций был чрезвычайно общим, что значительно усложняет доказательство ряда важных теорем. К счастью, существует возможность существенно ограничить формат продукций, но так, чтобы это не отразилось на выразительных свойствах грамматик.

ТЕОРЕМА 5.2. (НОРМАЛЬНАЯ ФОРМА ХОМСКОГО.)

Любой ε-свободный контекстно свободный язык может быть порожден некоторой контекстно свободной грамматикой в форме Хомского, т. е. грамматикой, все продукции которой имеют вид

$$A \to BC,\ A, B, C \in N$$

или вид

$$A \to a,\ A \in N,\ a \in T$$

Доказательство

Пусть $L \subset T^*$ — ε-свободный контекстно свободный язык, пусть $G = (N, T, P, S)$ — ε-свободная контекстно свободная грамматика, порождающая

этот язык. Определим эквивалентную ей грамматику в нормальной форме Хомского, воспользовавшись следующим алгоритмом.

Шаг 1. Отметим все продукции вида $A \to B$, $A, B \in N$. (Будем называть такие продукции первичными продукциями.) Пусть $A \in N$ – некоторый нетерминальный символ; обозначим $U(A)$ множество первичных продукций, содержащих в левой части нетерминальный символ A, а множество продукций, не являющихся первичными, но содержащих в левой части нетерминальный символ A, обозначим $N(A)$. Тогда множеству $U(A)$ для каждого $A \in N$, такого, что $U(A) \neq \emptyset$, поставим в соответствие множество $\{A \to \alpha | A \overset{+}{\underset{G}{\Rightarrow}} B$ и $B \to \alpha$ – элемент множества $N(B)\}$.

Шаг 2. Отметим все продукции, правые части которых представляют строки длиной > 1 и включающие в себя подстроки терминальных символов. Такие продукции будем называть вторичными продукциями.

Каждому терминальному символу a, входящему в правую часть некоторой вторичной продукции, поставим в соответствие новый нетерминальный символ A_a и новую продукцию вида $A_a \to a$. В результате каждой вторичной продукции вида $A \to x_1 x_2 \ldots x_m$, $x_i \in N \cup T$ будет поставлена в соответствие продукция вида $A \to Y_1 Y_2 \ldots Y_m$, где

$$Y_i = \begin{cases} X_i, \text{ если } X_i \in N; \\ X_a, \text{ если } X_i = a, \text{ где } a \in T. \end{cases}$$

Расширенное таким образом множество нетерминальных символов обозначим N'.

Шаг 3. Отметим все продукции, правые части которых представляют собой строки, содержащие не менее трех нетерминальных символов. Такие продукции мы будем называть третичными продукциями. Любой третичной продукции, имеющей вид $A \to B_1 B_2 \ldots B_m$, $m > 2$, $B_1, B_2, \ldots, B_m \in N'$, поставим в соответствие совокупность продукций вида $A \to BB'_1, B'_1 \to B_2 B'_2, \ldots$ $\ldots, B'_{m-1} \to B_{m-1} B_m$, где $B'_1, B'_2, \ldots, B'_{m-1}$ – новые нетерминальные символы, не содержащиеся более ни в одной продукции.

Теперь ясно, что грамматика, полученная в результате описанных выше преобразований грамматики G, эквивалентна последней и представляет собой грамматику в нормальной форме Хомского. Рассмотрим пример. Пусть контекстно свободная грамматика G_2 имеет продукции вида

$S \to A | A B A,$

$A \to aA | a | B,$

$B \to bB | b.$

Шаг 1. Множество $U(S)$ содержит продукцию вида $S \to A$, множество $U(A)$ включает в себя продукцию вида $A \to B$, множество $U(B)$ – пустое. Поскольку $S \overset{+}{\Rightarrow} A$ и $S \overset{+}{\Rightarrow} B$, продукции $S \to A$ поставим в соответствие про-

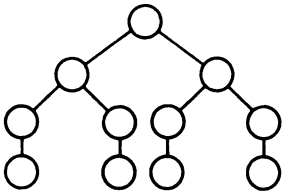


Рис. 5.2

дукцию $S \to aA|a|bB|b$, а поскольку $A \stackrel{+}{\Rightarrow} B$, продукции $A \to B$ поставим в соответствие продукцию $A \to bB|b$. В результате такого преобразования получим эквивалентную грамматику, имеющую продукции вида

$$S \to aA|a|bB|b|ABA,$$
$$A \to aA|a|bB|b,$$
$$B \to bB|b.$$

Шаг 2. Терминальным символом a и b, содержащимся в правых частях вторичных продукций поставим в соответствие два новых нетерминальных символа A_a и A_b и две новые продукции вида $A_a \to a$ и $A_b \to b$. Переписав таким образом вторичные продукции, вновь получим эквивалентную грамматику, имеющую продукции следующего вида:

$$S \to A_a A|a|A_b B|b|ABA,$$
$$A \to A_a A|a|A_b B|b,$$
$$B \to A_b B|b,$$
$$A_a \to a,$$
$$A_b \to b.$$

Шаг 3. Единственная третичная продукция имеет вид $S \to ABA$. Поставим в соответствие такой продукции совокупность двух продукций вида

$$S \to AB',$$
$$B' \to BA,$$

получив таким образом эквивалентную G грамматику в нормальной форме Хомского.

Пусть $L \subset T^*$ – некоторый произвольный контекстно свободный язык, и пусть $L \backslash \{\varepsilon\}$ – язык, порожденный контекстно свободной грамматикой G в нормальной форме Хомского. Если некоторому $x \in L(G)$ соответствует дерево вывода, имеющее глубину дерева m, то можно доказать, что $|x| \leqslant \leqslant 2^{m-1}$. Это заключение – тривиально и следует из простого утверждения, что каждый родительский узел дерева вывода может иметь не более двух узлов-потомков, а родительские узлы для узлов, помеченных терминальными символами, могут иметь лишь по одному узлу-потомку. Для иллюстрации сказанного рассмотрите дерево вывода глубины 3, приведенное на рис. 5.2.

ТЕОРЕМА 5.3

Если L – контекстно свободный язык, тогда существуют целые числа l_1 и l_2, такие, что для любого $z \in L$, удовлетворяющего условию $|z| > l_1$, справедливо утверждение: $z = uvwxy$, и при этом

(I) $|vwx| \leqslant l_2$;

(II) $vx \neq \varepsilon$

(III) для любого целого $i \geqslant 0$ справедливо $uv^i wx^i y \in L$.

Доказательство

Пусть $L\backslash\{\varepsilon\}$ – язык, порожденный грамматикой $G = (N, T, P, S)$ в нормальной форме Хомского. Положим $n = \#(N)$, $l_1 = 2^{n-1}$, $l_2 = 2^n$. Пусть $z \in L$ и $|z| > 2^{n-1}$, отсюда следует, что максимально длинный путь по любому дереву вывода строки z (обозначим его через P) должен иметь длину $> n$, т. е. проходить более чем через $n+1$ узел графа, из которых лишь один помечен терминальным символом.

Но тогда, очевидно, должны существовать два узла, через которые проходит путь P, помеченные одним и тем же нетерминальным символом. Обозначим эти узлы N_1 и N_2, и пусть узел N_1 находится ближе к корню дерева, чем узел N_2. Если путь P проходит через несколько таких пар узлов, выберем в качестве N_1 и N_2 тот из них, у которого узел N_1 находится на большей глубине на дереве. Предположим теперь, что оба узла, N_1 и N_2, помечены нетерминальным символом A. Поскольку путь P был выбран как самый длинный путь по графу, то очевидно, тот его отрезок, который принадлежит поддереву с корнем в узле N_1, тоже будет самым длинным на указанном поддереве, следовательно, глубина этого поддерева равна, по крайней мере, $n+1$. А это означает, что указанное поддерево может стать деревом вывода из нетерминального символа A строки терминальных символов длиной 2^n. Обозначим одну из таких строк z_1. Следовательно, $|z_1| \leqslant 2^n$ и $A \stackrel{+}{\Rightarrow} z_1$. Если T_2 – поддерево с корнем в узле N_2 и w – строка терминальных символов, выведенная из N_2, то можно написать $z_1 = vwx$, где подстроки v и w не могут одновременно быть пустыми, так как первая продукция, использованная для вывода строки z_1, должна иметь вид $A \to BC$, $B, C \in N$, и не может быть пустой.

Далее, строка z_1 – подстрока строки z и, следовательно, справедлива следующая запись:

$$z = uz_1y = uvwxy.$$

Таким образом, мы показали, что существует вывод вида

$$S \stackrel{*}{\Rightarrow} uAy \stackrel{*}{\Rightarrow} uvAxy \stackrel{*}{\Rightarrow} uvwxy,$$

где $|vwx| \leqslant l_2$. Поскольку $A \stackrel{*}{\Rightarrow} vAx \stackrel{*}{\Rightarrow} vwx$, то $A \stackrel{*}{\Rightarrow} v^iwx^i$ для любого $i > 0$, следовательно, $S \stackrel{*}{\Rightarrow} uAy \stackrel{*}{\Rightarrow} uv^iwx^iy$ – корректный вывод для любого $i \geqslant 0$ (рис. 5.3).

Проиллюстрируем теорему 5.3: пусть G_3 – грамматика, имеющая продукции вида

$$S \to AS \mid AB,$$

$$A \to BS \mid a,$$

$$B \to AA \mid b$$

(в данном случае $l_1 = 4$). Рассмотрим дерево вывода строки a^3b^2, приведенное на рис. 5.4, *а*. Путь P в нашем примере выделен как путь максимальной длины по дереву и проходит через четыре узла, следовательно, два из этих узлов должны быть помечены одним и тем же нетерминаль-

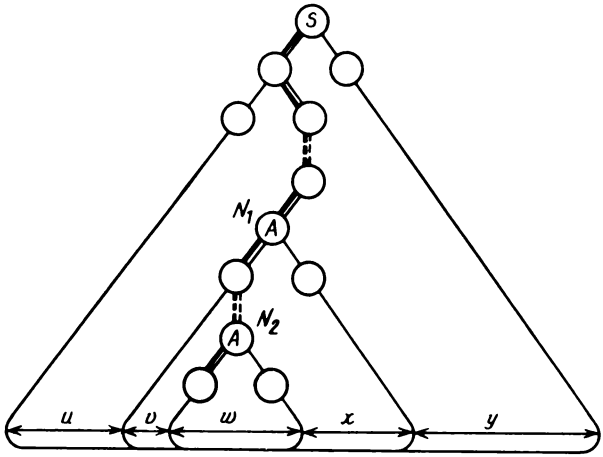


Рис. 5.3

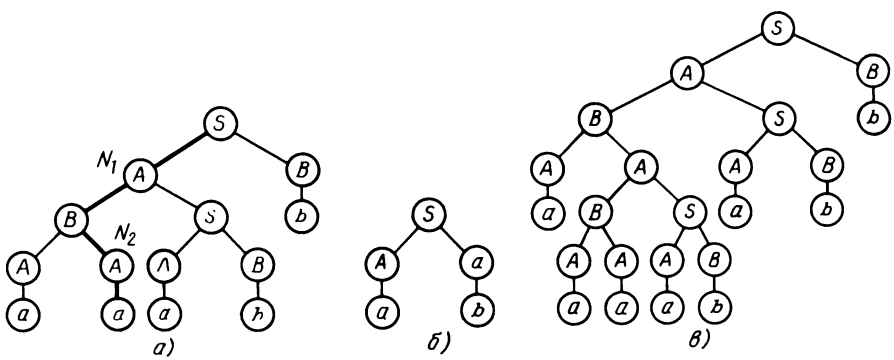


Рис. 5.4

ным символом, в данном случае символом A. Обозначим тот из узлов, который находится ближе к корню дерева, N_1, а другой N_2. Пусть T_1 – поддерево исходного дерева вывода, имеющее корень в узле N_1 и соответствующее схеме вывода вида $A \stackrel{+}{\Rightarrow} a^3 b$. Обозначим z_1 строку $a^3 b$. Пусть T_2 – поддерево, имеющее корень в узле N_2 и соответствующее схеме вывода вида $A \Rightarrow a$. Тогда $z_1 = vwx$, где $v = a$, $w = a$, $x = ab$, а $z = uvwxy$, где $u = \varepsilon$, $y = b$. Из теоремы 5.3 следует, что для любого целого $i \geqslant 0$ строка $uv^i wx^i y$ является элементом языка L, т. е. $uv^i wx^i y \in L$. На рис. 5.4, *б* и 5.4, *в* приведены деревья вывода для случаев $i = 0$ и $i = 2$ соответственно. Рисунок 5.4, *б* отличается от рис. 5.4, *а* заменой поддерева T_1 на поддерево T_2, и рис. 5.4, *в* отличается от рис. 5.4, *а* заменой поддерева T_2 на поддерево T_1.

Особенно полезна теорема 5.3 бывает в тех случаях, когда необходимо доказать, что рассматриваемый язык не является контекстно свободным языком. Например, рассмотрим язык $L\{a^p | p$ – простое число$\}$. Предполо-

жим, что язык L – контекстно свободный язык, тогда согласно теореме 5.3 существует целое число k, $0 \leqslant k \leqslant p$, такое, что $p + 2ki$ – простое число для любого $i \geqslant 0$. Очевидно, что это утверждение несправедливо и, следовательно, L не является контекстно свободным языком.

Пусть G – произвольная контекстно свободная грамматика в нормальной форме Хомского. Предположим, что грамматика G имеет n нетерминальных символов, $l_1 = 2^{n-1}$ и $l_2 = 2^n$. Тогда по теореме 5.3 язык $L(G)$ конечен $\Leftrightarrow$ $L(G)$ содержит строку длины большей, чем l_1. Действительно, предположим, что $L(G)$ – бесконечен, пусть z – самая короткая из его строк, имеющих длину большую, чем l_1. Докажем методом "от противного", что $l_1 < |z| \leqslant l_1 + l_2$. Предположим, что $|z| > l_1 + l_2$ и, следовательно, (по определению z) язык $L(G)$ не содержит строки, меньшей по длине, чем $l_1 + l_2$, но большей, чем l_1. Но по теореме 5.3 $z = uvwxy$ и $uwy \in L$ – строка длины $|z| - l_2 > l_1$. С другой стороны, очевидно, что $|uwy| < |z|$, и полученное противоречие доказывает неверность нашего предположения.

Итак, мы только что доказали, что $L(G)$ – бесконечное множество $\Leftrightarrow$ существует $z \in L(G)$, такое, что $l_1 < |z| \leqslant l_1 + l_2$. Заметим, что число строк, являющихся элементами языка $L(G)$ и удовлетворяющих указанному соотношению, конечно. Чтобы убедиться в этом, достаточно рассмотреть все сентенциальные формы грамматики G, имеющие длину k, $l_1 < k \leqslant l_1 + l_2$. Таким образом, мы получили алгоритм определения конечности языка и доказали следующую теорему.

ТЕОРЕМА 5.4

Проблема конечности разрешима для контекстно свободных грамматик, т.е. существует алгоритм, позволяющий определить, конечный или бесконечный язык порождает контекстно свободная грамматика.

НОРМАЛЬНАЯ ФОРМА ГРЕЙБАХА

При рассмотрении регулярных грамматик в гл. 3 всякий раз нам удавалось каким-то чудесным образом найти способ ответить на вопрос: "является ли строка $x \in T^*$ элементом языка $L(G)$?" Тем не менее формально ответить на такой вопрос непросто, и он порождает проблему, называемую "проблемой принадлежности". Для контекстно свободных грамматик эта проблема разрешима приведенным выше способом – достаточно рассмотреть все возможные сентенциальные формы грамматики G, имеющие длину $|x|$. Если ни одна из них не соответствует выводу строки x, то $x \notin L(G)$, в противном случае наоборот. Однако отыскание всех сентенциальных форм грамматики G, имеющих длину $|x|$ – мало эффективный путь решения проблемы принадлежности. Кроме того, интересно было бы решить еще одну важную проблему – проблему построения реального дерева вывода строки x (ее называют "проблемой выводимости"). Понятно, что если построен эффективный алгоритм решения проблемы выводимости, то его можно использовать и для решения проблемы принадлежности.

Практика показывает, что эффективное решение проблемы выводимости возможно лишь в случае наложения ряда ограничений на определение

контекстно свободной грамматики. При этом вид накладываемых ограничений отразится на виде конкретных алгоритмов. К сожалению, эти ограничения сформулированы так, что не всякий, порожденный контекстно свободной грамматикой язык, может быть порожден контекстно свободной грамматикой, на которую наложены эти ограничения. (Этот вопрос будет рассмотрен в гл. 6, 7 и 8.) Кроме того, чрезвычайно важно, чтобы читатель получил навыки осуществления процедуры вывода, изменения формата продукций, не меняющего содержимого множества $L(G)$.

Один из рассматриваемых здесь приемов осуществления подобных манипуляций является результатом теоремы 5.5, доказательство которой мы оставляем читателю.

ТЕОРЕМА 5.5

Если $A \to \alpha_1 B \alpha_2$ – продукция контекстно свободной грамматики G и $B \to \beta_1 | \beta_2 | \ldots | \beta_k$ – все продукции этой грамматики, содержащие в своих левых частях нетерминальный символ B, тогда продукции вида $A \to \alpha_1 B \alpha_2$ можно поставить в соответствие продукцию вида $A \to \alpha_1 \beta_1 \alpha_2 | \alpha_1 \beta_2 \alpha_2 | \ldots | \alpha_1 \beta_k \alpha_2 |$ и это никак не отразится на языке L.

Следующая теорема не столь очевидна, поэтому мы приведем ее доказательство. Теорема обращена к "проблеме удаляемости" из контекстно свободных грамматик *рекурсивных слева продукций*, т. е. продукций вида $A \to A\alpha, A \in N, \alpha \in (N \cup T)^*$.

ТЕОРЕМА 5.6

Если все продукции некоторой контекстно свободной грамматики являются рекурсивными слева продукциями, $A \to A\alpha_1 | A\alpha_2 | \ldots | A\alpha_m, A \to \beta_1 | \beta_2 | \ldots | \beta_n$ – оставшиеся продукции грамматики, имеющие нетерминальный символ A в своей левой части, то новая грамматика может быть построена добавлением в множество N нетерминального символа A и полной заменой ее продукций исходной грамматики продукциями вида

$A \to \beta_1 | \beta_2 | \ldots | \beta_n | \beta_1 A' | \beta_2 A' | \ldots | \beta_n A'$;

$A' \to \alpha_1 | \alpha_2 | \ldots | \alpha_m | \alpha_1 A' | \alpha_2 A' | \ldots | \alpha_m A'$.

Доказательство

В обоих случаях множество строк, которые могут быть выведены из A, имеет вид $\{\beta_1, \beta_2, \ldots, \beta_n\}\{\alpha_1, \alpha_2, \ldots, \alpha_m\}^*$.

Для иллюстрации применения этой теоремы преобразуем продукции грамматики G_3 к виду $A \to a\alpha$, $A \in N$, $a \in T$, $\alpha \in (N \cup T)^*$. Если все продукции некоторой контекстно свободной грамматики имеют такой вид, то говорят, что эта грамматика – контекстно свободная грамматика в *нормальной форме Грейбаха*. Для любого ε-свободного контекстно свободного языка L можно определить контекстно свободную грамматику в нормальной форме Грейбаха такую, что она порождает язык L. Вернемся к примеру.

Прежде всего переименуем нетерминальные символы грамматики G_3 следующим образом: S переименуем в A_1, A переименуем в A_2, B переименуем в A_3. В результате продукции грамматики G_3 будут иметь вид

$$A_1 \to A_2A_1 \mid A_2A_3,$$
$$A_2 \to A_3A_1 \mid a,$$
$$A_3 \to A_2A_2 \mid b.$$

На первом шаге попытаемся преобразовать продукции грамматики G_3 к виду $A_i \to A_j\alpha$, где $j > i$. В данном случае из всех продукций грамматики G продукция $A_3 \to A_2A_2$ не имеет такого вида, поэтому, воспользовавшись теоремой 5.5, преобразуем такую продукцию следующим образом:

$$A_3 \to A_3A_1A_2 \mid aA_2,$$

а затем воспользовавшись теоремой 5.6, вновь преобразуем данную продукцию. В результате продукции грамматики G_3 примут вид

$$A_1 \to A_2A_1 \mid A_2A_3,$$
$$A_2 \to A_3A_1 \mid a,$$
$$A_3 \to aA_2 \mid b \mid aA_2A_3' \mid bA_3',$$
$$A_3' \to A_1A_2 \mid A_1A_2A_2'.$$

Очевидно, что каждая продукция, содержащая в левой части нетерминальный символ A_3, удовлетворяет условиям нормальной формы Грейбаха.

Вновь воспользуемся теоремой 5.5 и преобразуем продукции

$$A_2 \to A_3A_1,$$
$$A_1 \to A_2A_1 \mid A_2A_3,$$

в результате чего все полученные продукции грамматики G_3, содержащие в левых частях нетерминальные символы A_1, A_2 или A_3, будут удовлетворять условиям нормальной формы Грейбаха:

$$A_1 \to aA_2A_1A_1 \mid bA_1A_1 \mid aA_2A_3'A_1A_1 \mid bA_3'A_1A_1 \mid aA_1$$
$$\mid aA_2A_1A_3 \mid bA_1A_3 \mid aA_2A_3'A_1A_3 \mid bA_3'A_1A_3 \mid aA_3;$$
$$A_2 \to aA_2A_1 \mid bA_1 \mid aA_2A_3'A_1 \mid bA_3'A_1 \mid a;$$
$$A_3 \to aA_2 \mid b \mid aA_2A_3' \mid bA_3';$$
$$A_3' \to A_1A_2 \mid A_1A_2A_3'.$$

И наконец, преобразуем согласно теореме 5.5 все те продукции, в левых частях которых содержатся символы A_3':

$$A_3' \to aA_2A_1A_1A_2 \mid bA_1A_1A_2 \mid aA_2A_3'A_1A_1A_2 \mid bA_3'A_1A_1A_2 \mid aA_1A_2$$
$$\mid aA_2A_1A_3A_2 \mid bA_1A_3A_2 \mid aA_2A_3'A_1A_3A_2 \mid bA_3'A_1A_3A_2 \mid aA_3A_2$$
$$\mid aA_2A_1A_1A_2A_3' \mid bA_1A_1A_2A_3' \mid aA_2A_3'A_1A_1A_2A_3'$$
$$\mid bA_3'A_1A_1A_2A_3' \mid aA_1A_2A_3' \mid aA_2A_1A_3A_2A_3' \mid bA_1A_3A_2A_3'$$
$$\mid aA_2A_3'A_1A_3A_2A_3' \mid bA_3'A_1A_3A_2A_3' \mid aA_3A_2A_3'.$$

Итак, рассматриваемая нами контекстно свободная грамматика в нормальной форме Грейбаха имеет 39 продукций. Покажем, что все выше-

приведенные преобразования справедливы и для общего случая, а полученная в результате грамматика эквивалентна исходной.

Шаг 1. Пусть G – произвольная ε-свободная грамматика. Преобразуем G в эквивалентную ей грамматику G' в нормальной форме Хомского.

Шаг 2. Определим множество нетерминальных символов грамматики G' как $\{A_1, A_2, \ldots, A_m\}$, $m \geqslant 1$, где A_1 соответствует начальному нетерминальному символу. Пусть $N = \{A_1, A_2, \ldots, A_m\}$.

Шаг 3. По теореме 5.5 каждую продукцию грамматики G', содержащую в левой части нетерминальный символ A_i, преобразуем к виду $A_i \to a\alpha$, $a \in T$, или к виду $A_i \to A_j\alpha$, где $j > i$, воспользовавшись следующим алгоритмом

begin
var i: integer;
$i := 0$;
while $i \langle\rangle m$ *do*
begin $i := i + 1$;
while существует продукция вида $A_i \to A_j\alpha$, $j < i$ *do* подставить вместо A_j соответствующее ему согласно теореме 5.5 выражение;
if существуют рекурсивные слева продукции, содержащие символ A_i в левой части;
then ввести новый нетерминальный символ A_i' и по теореме 5.6 заменить рекурсивную слева продукцию эквивалентным ей набором продукций.
end
end

Шаг 4. Пусть N – множество нетерминальных символов, введенных на третьем шаге. Тогда справедливо следующее утверждение: после третьего шага преобразования все продукции грамматики будут иметь вид $A_i \to A_j\alpha$, $j > i$, $\alpha \in (N \cup N' \cup T)^*$; или вид $A_i \to a\alpha$, $a \in T$, $\alpha \in (N \cup N' \cup T)^*$; или вид $A_i' \to X\alpha$, $X \in (N \cup T)$, $\alpha \in (N \cup N' \cup T)^*$.

Применим ко всем продукциям, имеющим первый из перечисленных видов, следующее преобразование:

begin
var i: integer;
$i := m$;
while $i \langle\rangle 1$ *do*
begin $i := i - 1$;
while существует продукция вида $A_i \to A_j\alpha$, $j > i$ *do* заменить A_i согласно теореме 5.5
end
end

Шаг 5. Исключить продукции вида $A_i' \to A_j\alpha$, заменив A_j согласно теореме 5.5. Полученная таким образом грамматика является грамматикой в нормальной форме Грейбаха.

Понятно, что грамматика, полученная после второго шага, порождает язык $L(G)$, если в качестве начального символа выбран символ A_1. Чтобы доказать, что шаг 3 выполняет свое назначение, воспользуемся методом индукции по параметру основного цикла, т. е. покажем, что после i итераций будут преобразованы все продукции, содержащие в левой части нетерминальный символ A_k, $k \leqslant i$. Для $i = 0$ никаких преобразований не производилось и, следовательно, ни одна продукция не преобразована, значит, для $i = 0$ утверждение справедливо. Предположим, что утверждение верно для всех $i < n$ итераций и рассмотрим n-ю итерацию. Если существуют продукции вида $A_n \to A_j \alpha$, где $j < n$, то по теореме 5.5 можно подставить вместо символов A_j правые части тех продукций, которые содержат символ A_j в левых частях.

Поскольку $j < n$, то согласно предположению индукции правые части таких продукций должны начинаться либо с терминального символа, либо с символа A_k, где $k > j$. Ясно, что продукции первого типа удовлетворяют условиям нормальной формы Грейбаха, и потому интереса не представляют, что касается продукций второго типа, для $k < n$, то после $n-1$ итерации внутреннего цикла все продукции, левые части которых содержат символ A_n, содержат в правых частях в качестве первого символа либо терминальный символ, либо нетерминальный символ A_k, где $k \geqslant n$. Следовательно, для $i = n$ предположение индукции справедливо, и алгоритм, реализуемый на 3-м шаге, выполняет свое назначение, так как заканчивается при $i = m$ и все продукции имеют нужный вид.

Можно заметить, что правая часть каждой продукции, левая часть которой содержит вновь введенный нетерминальный символ, должна содержать в качестве первого символа элемент множества $N \cup T$, и следовательно, шаг 3 выполняет свое назначение. Из этого, в частности, следует, что правая часть продукции, левая часть которой содержит символ A_m, должна в качестве первого символа содержать терминальный символ. Любая продукция, содержащая в левой части символ A_{m-1}, содержит в правой части в качестве первого символа либо терминальный символ, либо символ A_m. В то же время все продукции вида $A_{m-1} \to A_m \alpha$ могут быть преобразованы по теореме 5.5, и тогда все продукции, содержащие в левой части символы A_m или A_{m-1}, будут иметь в правой части в качестве первого символа терминальный символ. Этот процесс надо повторить для всех продукций, имеющих в левой части один из символов A_m, $A_{m-1}, \ldots, A_1$. Методом индукции по числу итераций легко показать, что после n итераций цикла, на шаге 4, все продукции, содержащие в левой части один из символов A_m, $A_{m-1}, \ldots$ $\ldots, A_{m-n}$, в правой части обязательно будут содержать в качестве первого символа терминальный символ. Шаг 4 преобразования, таким образом, выполняет свое назначение, так как описанный в нем алгоритм заканчивается после выполнения $m-1$ итерации. Шаг 5 тривиален и доказательства не требует. Следовательно, мы доказали следующую теорему.

ТЕОРЕМА 5.7. НОРМАЛЬНАЯ ФОРМА ГРЕЙБАХА

Для каждого ε-свободного контекстно свободного языка существует контекстно свободная грамматика в нормальной форме Грейбаха, такая, что $L = L(G)$.

КОНТЕКСТНО СВОБОДНЫЙ ЯЗЫК КАК РЕШЕНИЕ УРАВНЕНИЯ

Рассмотрим контекстно свободную грамматику, имеющую продукции вида

$$S \to AB,$$

$$A \to aA \mid a,$$

$$B \to bB \mid b.$$

Поставим в соответствие каждому нетерминальному символу множество строк, которые могут быть из него выведены, например нетерминальному символу A из рассмотренной выше грамматики соответствует множество $\{a^m \mid m \geqslant 1\}$, нетерминальному символу B соответствует множество $\{b^n \mid n \geqslant 1\}$, а нетерминальному символу S соответствует множество $\{a^m b^n \mid m, n \geqslant 1\}$. Эти множества строк будем обозначать непосредственно теми символами, которым они соответствуют, например,

$$A = \{a^m \mid m \geqslant 1\}, B = \{b^n \mid n \geqslant 1\}, S = \{a^m b^n \mid m, n \geqslant 1\}.$$

С другой стороны, из определения продукции грамматики, множества S, A, B должны удовлетворять следующим условиям, имеющим вид уравнений:

$$S = AB,$$

$$A = \{a\}A \cup \{a\},$$

$$B = \{b\}B \cup \{b\}.$$

Система таких уравнений может иметь бесконечное множество решений. Например, уравнение

$$A = A \cup \{a\}$$

имеет решение $\{a\}$, но в то же время любое множество, включающее множество $\{a\}$, очевидно, также будет решением указанного уравнения. Вообще говоря, далее мы будем называть решением уравнения минимальное из таких множеств[1].

Рассмотрим следующую систему уравнений:

$$X_1 = f_1(X_1, X_2, \ldots, X_n);$$

$$X_2 = f_2(X_1, X_2, \ldots, X_n);$$

$$\ldots\ldots\ldots\ldots\ldots$$

$$X_n = f_n(X_1, X_2, \ldots, X_n),$$

---

[1] Имеется в виду мощность множества. – *Прим. ред.*

где каждая функция $f_i(X_1, X_2, \ldots, X_n)$ – конечная функция, а каждое множество X_i – конечное множество строк из множества T^*, причем на множестве переменных $X_1, X_2, \ldots, X_n$ введены всего две операции – операция объединения и операция конкатенации. Такую систему уравнений будем называть *системой уравнений* над множеством T^*. Совокупность $(X_1, X_2, \ldots, X_n)$ будем обозначать символом $\mathbf{X}$, а систему уравнений записывать в виде $\mathbf{X} = f(\mathbf{X})$. Решение такой системы уравнений, очевидно, будет совокупностью подмножеств множества T^* (обозначим его $\mathbf{S} = (S_1, S_2, \ldots, S_n)$), следовательно,

$$\mathbf{S} = f(\mathbf{S}).$$

Если $\mathbf{T} = (T_1, T_2, \ldots, T_n)$, то будем считать, что

$$\mathbf{S} \subset \mathbf{T} \Leftrightarrow S_1 \subset T_1, S_2 \subset T_2, \ldots, S_n \subset T_n.$$

ТЕОРЕМА 5.8

Система уравнений $\mathbf{X} = f(\mathbf{X})$ имеет решение

$$\mathbf{S} = \bigcup_{i=1}^{\infty} f^i(\varnothing),$$

более того, если T – некоторое другое решение, то $S \subset T$.

Доказательство

Так как $\varnothing = (\varnothing, \varnothing, \ldots, \varnothing)$, то по определению $\varnothing \subset f(\varnothing)$. Воспользовавшись решением упр. 1.11 можно показать, что если $\mathbf{A} \subset \mathbf{B}$, то $f(\mathbf{A}) \subset f(\mathbf{B})$. Таким образом, $f(\varnothing) \subset f(f(\varnothing)) = f^2(\varnothing)$, и продолжая это рассуждение, получаем неубывающую последовательность $\varnothing \subset f(\varnothing) \subset f^2(\varnothing) \subset \ldots$. Пусть

$$\mathbf{S} = \bigcup_{i=1}^{\infty} f^i(\varnothing)$$

– предел этой последовательности; тогда можно показать, что $f(\mathbf{S}) = \mathbf{S}$, т. е. $\mathbf{S}$ – решение системы уравнений.

Если $\mathbf{T}$ – некоторое другое решение этой же системы уравнений, то $\mathbf{T} = f(\mathbf{T})$. По определению, $\varnothing \subset \mathbf{T}$, следовательно, $f(\varnothing) \subset f(\mathbf{T}) = \mathbf{T}$. Повторяя это рассуждение, получим, что $f^i(\varnothing) \subset \mathbf{T}$ для всех $i \geqslant 0$. Таким образом,

$$\bigcup_{i=1}^{\infty} f^i(\varnothing) \subset \mathbf{T}, \text{ т. е. } \mathbf{S} \subset \mathbf{T}.$$

Итак, всякой заданной контекстно свободной грамматике $G = (N, T, P, S)$ может быть поставлена в соответствие система уравнений множеств над T^* и наоборот. Более того, если $\mathbf{S} = (S_1, S_2, \ldots, S_n)$ – единственное минимальное решение этой системы уравнений, и если первое уравнение указанной системы уравнений соответствует тем продукциям, в левых частях которых содержится символ S, то $L(G) = S_1$.

Проиллюстрируем сказанное на следующем примере:

$$S = AB = f_1(S, A, B);$$

$$A = \{a\}A \cup \{a\} = f_2(S, A, B);$$

$$B = \{b\}B \cup \{b\} = f_3(S, A, B).$$

$$f(\varnothing) = (f_1(\varnothing), f_2(\varnothing), f_3(\varnothing)) = (\varnothing, \{a\}, \{b\});$$

$$f^2(\varnothing) = (f_1(\varnothing, \{a\}, \{b\}), f_2(\varnothing, \{a\}, \{b\}), f_3(\varnothing, \{a\}, \{b\})) =$$
$$= (\{ab\}, \{a, aa\}, \{b, bb\}),$$
$$f^3(\varnothing) = f(\{ab\}, \{a, aa\}, \{b, bb\}) =$$
$$= (\{ab, aab, abb, aabb\}, \{a, aa, aaa\}, \{b, bb, bbb\}).$$

Методом индукции по i можно показать, что

$$f^i(\varnothing) = (\{a^m b^n | i > m, n \geqslant 1\}, \{a^m | i \geqslant m \geqslant 1\}, \{b^n | i \geqslant n \geqslant 1\}),$$

и, следовательно, $\bigcup_{i=1}^{\infty} f^i(\varnothing) = (\{a^m b^n | m, n \geqslant 1\}, \{a^m | m \geqslant 1\}, \{b^n | n \geqslant 1\})$, а значит $L(G) = \{a^m b^n | m, n \geqslant 1\}$.

Представление контекстно свободного языка как решения системы уравнений помогает лучше понять его природу. Кроме того, все элементы множества $L(G)$ могут быть получены за конечное число шагов $\varnothing \subset f(\varnothing) \subset \subset f^2(\varnothing) \ldots$, следовательно, этот метод может быть использован для решения проблемы принадлежности.

УПРАЖНЕНИЯ

5.1. Опишите алгоритм, с помощью которого по заданной контекстно свободной грамматике G и целому числу $k \geqslant 0$, можно построить все возможные деревья вывода глубины k.

5.2. Пусть L – контекстно свободный язык, покажите, что L^* и $L^r = \{x^r | x \in L\}$ контекстно свободные языки.

5.3. Пусть L_1, L_2 – контекстно свободные языки, покажите, что $L_1 \cup L_2$ и $L_1 L_2$ – тоже контекстно свободные языки.

5.4. Определите контекстно свободную грамматику в нормальной форме Хомского, порождающую арифметические выражения над множеством $\{a, b, c\}$.

5.5. Укажите, какие из приведенных ниже языков – контекстно свободные языки:

а) $\{a^i b^j c^k | 0 \leqslant i < j < k\}$,
б) $\{a^i b^j c^k | 0 \leqslant i = j = k\}$,
в) $\{a^i b^j c^k | 0 \leqslant i = j, k \geqslant 0, i \neq k\}$,
г) $\{a^i b^j c^k | 0 \leqslant i = j, k \geqslant 0\}$.

5.6. Покажите, что если $L \subset T_1^*$ – контекстно свободный язык и $\varphi: T_1^* \to T_2^*$ – гомоморфизм, то $\varphi(L)$ – контекстно свободный язык.

5.7. Напишите формальное доказательство теорем 5.5 и 5.6.

5.8. Сформулируйте и докажите теорему, аналогичную теореме 5.6, для продукций рекурсивных справа, т. е. для продукций, имеющих вид $A \to \alpha A$, $A \in N$, $\alpha \in (N \cup T)^*$.

5.9. Приведите к нормальной форме Грейбаха грамматику, имеющую продукции вида

$A_1 \to A_2 A_3$,
$A_2 \to A_3 A_1 | b$,
$A_3 \to A_1 A_2 | a$.

5.10. Пусть G – контекстно свободная грамматика, имеющая продукции вида

$S \to AS|BS|a$,

$A \to BB|a$,

$B \to AA|b$.

Выпишите систему уравнений множеств, решением которой является множество $L(G)$, и с ее помощью найдите все $x \in L(G)$ имеющие длину не более 5.

5.11. Воспользуйтесь решением упр. 5.5 и постройте контекстно свободные языки L_1, L_2, такие, что $L_1 \cap L_2$ не являются контекстно свободным языком.

5.12. Воспользуйтесь решением упр. 5.11 и покажите, что существует контекстно свободный язык $L \subset T^*$, такой, что $\bar{L} = T^* \setminus L$ не является контекстно свободным языком.

ГЛАВА 6

МАГАЗИННЫЙ АВТОМАТ

"Если кому-то чего-то не хватает, он найдет это в той куче".

Уильям Шенк Гильберт "Колдунья"

Конечный автомат, рассмотренный в гл. 3 и гл. 4, служит прекрасным инструментом для обработки регулярных языков. В этой главе мы рассмотрим и определим автомат для обработки контекстно свободного языка и тем самым сделаем важный шаг по пути проектирования и практической реализации синтаксических анализаторов, являющихся важной частью любого компилятора.

Прежде всего, определим, что такое стек. *Стек* – это специально организованная память, выборка и занесение данных в которую подчиняется дисциплине LIFO, т. е. "последним вошел – первым обслужен". Подобная организация оперативной памяти ЭВМ, а также физические устройства, ее порождающие, наверняка знакомы большинству читателей. Адрес памяти, по которому может быть осуществлено запоминание в стеке очередного элемента, называют *вершиной стека*, а операции запоминания в стеке и выборки из стека очередного элемента называют соответственно *занесением в стек и извлечением из стека*. С помощью операции занесения в стек информационное слово может быть помещено в вершину стека, а с помощью операции извлечения из стека из его вершины может быть выбрано информационное слово, которое в таком случае и считается результатом операции извлечения из стека. В данном случае такими информационными словами, а следовательно, и элементами стека, являются символы некоторого алфавита, а размеры стека считаются неограниченными. Последнее требование исключает *переполнение стека* и делает, таким образом, всегда корректным выполнение операции занесения в стек. Понятно, что для корректности выполнения операции извлечения из стека стек должен содержать, по крайней мере, один элемент.

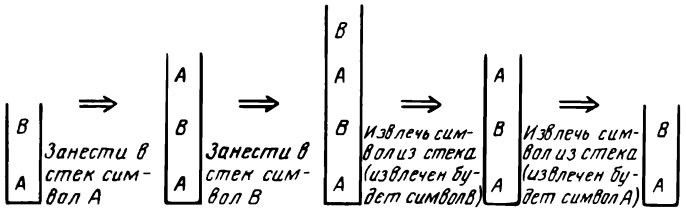


Рис. 6.1

Пусть V обозначает множество символов, которые могут быть занесены и соответственно извлечены из стека; в таком случае содержимое стека, очевидно, может быть представлено в виде строки — элемента множества V^*. При этом позиция первого символа строки соответствует вершине стека. Например, последовательность преобразований стека, изображенную на рис. 6.1, можно записать эквивалентным образом:

$$BA \Rightarrow ABA \Rightarrow BABA \Rightarrow ABA \Rightarrow BA.$$

Пустому стеку в таком случае соответствует строка ε, а операции занесения в стек и извлечения из стека эквивалентны операциям над строками и могут быть определены следующим образом:

$$\mathrm{PUSH}\ (A, X) = AX,$$

т. е. символ $A \in V$ занести в стек, содержащий строку $X = V^*$.

$$\mathrm{POP}_1(X) = \begin{cases} \text{неопределено}, & \text{если } X = \varepsilon \\ A, & \text{если } X = AY, A \in V, Y \in V^*, \end{cases}$$

т. е. извлечь символ из стека, содержащего строку $X \in V^*$.

$$\mathrm{POP}_2(X) = \begin{cases} \text{неопределено}, & \text{если } X = \varepsilon \\ Y, & \text{если } X = AY, A \in V, Y \in V^*, \end{cases}$$

т. е. определить содержимое стека после одной операции извлечения из стека.

НЕДЕТЕРМИНИРОВАННЫЙ МАГАЗИННЫЙ АВТОМАТ

Недетерминированный магазинный автомат кратко можно определить как такой недетерминированный конечный автомат, вид переходной функции которого зависит от содержимого вершины некоторого стека. Поясним это определение: набор возможных состояний недетерминированного конечного автомата определяется его текущим состоянием и входным алфавитом, аналогично набор возможных состояний недетерминированного автомата определяется его текущим состоянием, входным алфавитом и неким символом, который может быть в данный момент извлечен из некоторого стека. Заметим, однако, что при переходе недетерминированного магазинного автомата из одного состояния в другое в указанный стек может быть занесено произвольное конечное число некоторых символов.

Приведем теперь строгое определение недетерминированного магазинно-

го автомата: *недетерминированным магазинным автоматом* называют совокупность семи объектов $M = (K, T, V, p, k_1, A_1, F)$, где

а) K – конечное *множество состояний*;

б) T – конечный *входной алфавит*;

в) V – конечный *стековый алфавит* (его элементы обозначаются обычно прописными буквами, имеющими, быть может, нижние индексы);

г) p – полная функция, называемая *магазинной функцией* и отображающая множество $K \times V \times (T \cup \{\varepsilon\})$ на множество конечных подмножеств множества $K \times V^*$;

д) $k_1 \in K$ – *исходное состояние* автомата;

е) $A_1 \in V$ – *исходный символ* (символ, инициализирующий стек);

ж) $F \subset K$ – конечное множество *конечных состояний*.

Конфигурация недетерминированного магазинного автомата $M = (K, T, V, p, k_1, A_1, F)$ в любой момент времени описывается элементом множества $K \times V^*$, при этом первый элемент упорядоченной пары задает текущее состояние автомата, а второй элемент – текущее содержимое стека. Если недетерминированный магазинный автомат имеет в данный момент конфигурацию c, определим переходную функцию автомата t_M следующим образом: значением переходной функции $t_M(c, x)$ является множество конфигураций, которые может иметь автомат M после ввода строки $x \in T^*$, т. е. $t_M : (K \times V^*) \times (T \cup \{\varepsilon\}) \to 2^{K \times V^*}$.

Наложим на определенную таким образом переходную функцию t_M ограничения: пусть $c = (k, Ax)$, $k \in K$, $A \in V$, $x \in V^*$ и $c' = (k', Yx)$, $k' \in K$, $Y \in V^*$, $x \in V^*$ – две конфигурации автомата M, и пусть в результате одного вычисления магазинной функции p при условии ввода входного символа $a \in T \cup \{\varepsilon\}$ автомат M переходит из конфигурации c в конфигурацию c'[1], и $c' \in t_M(c, a)$.

Для определенной таким образом переходной функции справедливо следующее утверждение:

$$c' = (k', YX) \in t_M((k, AX), a) \Leftrightarrow (k', Y) \in p(k, A, a).$$

Заметим, что множество $t_M(c, \varepsilon)$ имеет единственный элемент c, и лишь в случае, если автомат M может перейти из одной конфигурации в другую без ввода входного символа, множество $t_M(c, \varepsilon)$ может иметь другие элементы, иначе говоря, если конфигурация $c = (k, Ax)$ такова, что $p(k, A, \varepsilon) \neq \neq \varnothing$. Такие перемещения автомата M называются ε-перемещениями. И, завершая определение переходной функции t_M, сделаем еще одно замечание:

$$t_M((k, \varepsilon), a) = \varnothing \quad \text{для всех } k \in K, a \in T \cup \{\varepsilon\},$$
$$t_M((k, \varepsilon), \varepsilon) = \{(k, \varepsilon)\} \text{ для всех } k \in K,$$

т. е. пустота стека является достаточным условием невозможности каких-либо перемещений автомата, кроме ε-перемещений.

---

[1] В процессе вычисления функции $p(k, A, a)$ из стека будет извлечен элемент A, в результате чего в вершине стека окажется элемент Y. – *Прим. ред.*

Расширим теперь функцию t_M до функции, осуществляющей отображение вида $(K \times V^*) \times T^* \to 2^{K \times V^*}$, следующим образом: $k \in K$, $X \in V^*$, $a_1, a_2, \ldots, \ldots, a_n \in T \cup \{\varepsilon\}$, тогда

$t_M((k, X), a_1 a_2 \ldots a_n)$ содержит $(k', X'), k' \in K, X' \in V^*$

тогда и только тогда, когда

$t_M((k, X), a_1 a_2 \ldots a_n)$ содержит $(k', X'), k' \in K, X' \in V^*$

$t_M((k, X), a_1)$ содержит (k_2, X_2),

$t_M((k_2, X_2), a_2)$ содержит (k_3, X_3),

. . .

. . .

. . .

$t_M((k_n, X_n), a_n)$ содержит (k', X')

для некоторых $k_2, \ldots, k_n \in K$, $X_2, \ldots, X_n \in V^*$. Понятно, что некоторые из входных символов a_i могут быть равны ε, и автомат M в этих случаях будет выполнять ε-перемещения.

Определим *множество принимаемых автоматом M строк* $T(M)$ следующим образом:

$T(M) = \{x \in T^* \mid t_M((k_1, A_1), x)$ содержит элемент множества $F \times V^*\}$.

Иначе говоря, $T(M)$ – множество таких строк, что в результате ввода любой из них автомат M перейдет из исходной конфигурации в конфигурацию с конечным состоянием автомата.

Далее везде, где это не вызовет неопределенности, будем опускать у функции t_M ее нижний индекс и писать просто "функция t".

Приведем примеры магазинных автоматов. Построим магазинный автомат M_1 такой, что $T(M_1) = \{xx^r \mid x \in \{a, b\}^*\}$. Пусть этот автомат решает задачу последовательного ввода строки и сравнения первой половины строки со второй половиной и пользуется при этом следующим алгоритмом: после занесения в стек первой половины строки продолжает посимвольный ввод второй половины строки, одновременно извлекая из стека символы первой половины строки и сравнивая пары получаемых символов. Поскольку строка вводится посимвольно, то длина ее заранее не известна, и поэтому определить момент, когда необходимо прекратить занесение в стек введенных входных символов, невозможно. Из этого следует, что построенный автомат является недетерминированным магазинным автоматом. Пусть при выполнении операции занесения в стек первой половины строки автомат находится в исходном состоянии k_1. В некоторый произвольный момент времени он переходит в состояние k_2, т. е. в состояние извлечения элементов из стека и сравнения их с вводимыми символами. Запишем теперь формальное определение автомата M_1 следующим образом: $M_1 = (\{k_1, k_2, k_3\}, \{a, b\}, \{a, b, \dashv\}, p, k_1, \dashv, \{k_3\})$, где символ "$\dashv$" обозначает нижнюю границу стека. Ниже приведено формальное определение магазинной функции p (далее при описании магазинной функции p соотношения, задающие значения мага-

зинной функции, совпадающие с пустым множеством $\mathcal{O}$, будут опускаться). Итак,

$$p(k_1, a, a) = \{(k_1, aa), (k_2, \varepsilon)\},$$
$$p(k_1, b, a) = \{(k_1, ab)\},$$
$$p(k_1, a, b) = \{(k_1, ba)\},$$
$$p(k_1, b, b) = \{(k_1, bb), (k_2, \varepsilon)\},$$
$$p(k_1, \dashv, a) = \{(k_1, a \dashv)\},$$
$$p(k_1, \dashv, b) = \{(k_1, b \dashv)\},$$
$$p(k_2, a, a) = \{(k_2, \varepsilon)\},$$
$$p(k_2, b, b) = \{(k_2, \varepsilon)\},$$
$$p(k_2, \dashv, \varepsilon) = \{(k_3, \varepsilon)\}.$$

Предлагаем читателю самостоятельно убедиться в том, что $T(M_1) = \{xx^r \mid x \in \{a, b\}^*\}$. Рассмотрим, например, строку $abba$, значением функции $t((k_1, \dashv), abba)$, очевидно, может быть любая из конфигураций: $(k_1, abba \dashv)$, $(k_2, \dashv)(k_3, \epsilon)$, и, следовательно, $abba \in T(M_1)$.

Согласно теореме 3.3 каждый регулярный язык $L \subset T^*$ может быть принят некоторым недетерминированным конечным автоматом, например автоматом $M = (K, T, t, k_1, F)$. Воспользуемся этим обстоятельством и построим недетерминированный магазинный автомат $M' = (K, T, \{\dashv\}, p, k_1, \dashv, F)$ такой, что $T(M') = L$. Фактически автомат M тождествен автомату M', ибо стек не используется. Достичь этого можно следующим определением магазинной функции p:

$$(k_1', \dashv) \in p(k_1, \dashv, a) \Leftrightarrow k' \in t(k, a).$$

Таким образом, стек остается пустым для всех перемещений, выполняемых автоматом M', и очевидно, что

$$x \in T(M') \Leftrightarrow t_{M'}((k_1, \dashv), x) \cap (F \times \{\dashv\}) \neq \varnothing \Leftrightarrow t(k_1, x) \cap F \neq \varnothing \Leftrightarrow x \in T(M) \Leftrightarrow$$
$$\Leftrightarrow x \in L.$$

Приведенное рассуждение доказывает, что всякий регулярный язык может быть принят некоторым недетерминированным магазинным автоматом. Кроме того, из рассмотренного выше примера следует, что можно построить недетерминированный магазинный автомат, принимающий нерегулярный язык $\{xx^r \mid x \in \{a, b\}^*\}$, поэтому можно утверждать, что недетерминированный магазинный автомат может принимать языки из более широкого класса, нежели недетерминированный конечный автомат. На самом деле, и мы это позже покажем, таким классом является класс всех контекстно свободных языков. У читателя может возникнуть вопрос: необходимо ли условие недетерминированности магазинного автомата? К сожалению, необходимо (в отличие от конечных автоматов, которые независимо от их детерминированности принимают языки из одного и того же класса). В этом легко убедиться, если вернуться к примеру со строками $\{xx^r \mid x \in \{a, b\}^*\}$. Позже мы подробнее рассмотрим вопрос недетерминированности магазинного автомата.

НЕДЕТЕРМИНИРОВАННЫЕ МАГАЗИННЫЕ АВТОМАТЫ И КОНТЕКСТНО СВОБОДНЫЕ ЯЗЫКИ

Основным результатом этой главы является утверждение: для любого контекстно свободного языка существует такой недетерминированный магазинный автомат, который принимает его, и наоборот – если некоторый недетерминированный магазинный автомат принимает некоторый язык, то этот язык является контекстно свободным языком. Прежде чем приступить к доказательству этого важного утверждения, приведем пример, с помощью которого покажем, как может быть построен недетерминированный магазинный автомат, принимающий простые арифметические выражения.

Пусть $G = (N, T, P, E)$ – контекстно свободная грамматика, где $N = \{E, T, F\}$, $T = \{a, b, c, (,), +, -, \times, /\}$, а множество продукций P имеет элементы

$$E \to T \mid E + T \mid E - T,$$

$$T \to F \mid T \times F \mid T/F,$$

$$F \to a \mid b \mid c \mid (E).$$

Из гл. 2 известно, что любой $x \in L(G)$ может быть получен в результате левостороннего вывода. Пусть недетерминированный магазинный автомат, который мы хотим построить, эмулирует указанный левосторонний вывод с помощью операций со стеком, например следующим образом: пусть в исходном состоянии автомата содержимое стека имеет вид $E \dashv$. Поставим в соответствие продукциям вида $E \to T$, $E \to E+T$, $E \to E - T$ следующие строки: $T \dashv$, $E+T \dashv$ и $E-T \dashv$ соответственно. При каждом перемещении, выполняемом автоматом, из стека извлекается один символ; если извлеченный символ оказывается нетерминальным, то ему в соответствие ставится некая продукция, правая часть которой в таком случае заносится в стек. Если же извлеченный из стека символ оказывается терминальным, то он используется в качестве входного символа и, следовательно, определяет следующее перемещение автомата.

Итак, мы определили недетерминированный магазинный автомат $M = (K, T, V, p, k_1, \dashv, \{k_3\})$, где

$$K = \{k_1, k_2, k_3\},$$
$$V = N \cup T \cup \{\dashv\},$$

а магазинная функция p задана следующими соотношениями:

$$p(k_1, \dashv, \varepsilon) = \{(k_2, E \dashv)\};$$
$$p(k_2, E, \varepsilon) = \{(k_2, T), (k_2, E + T), (k_2, E - T)\};$$
$$p(k_2, T, \varepsilon) = \{(k_2, F), (k_2, T \times F), (k_2, T/F)\};$$
$$p(k_2, F, \varepsilon) = \{(k_2, a), (k_2, b), (k_2, c), (k_2, (E))\};$$
$$p(k_2, a, a) = \{(k_2, \varepsilon)\}, p(k_2, b, b) = \{(k_2, \varepsilon)\};$$
$$p(k_2, c, c) = \{(k_2, \varepsilon)\};$$
$$p(k_2, +, +) = \{(k_2, \varepsilon)\}, p(k_2, -, -) = \{(k_2, \varepsilon)\};$$

$$p(k_2, \times, \times) = \{(k_2, \varepsilon)\}, p(k_2, /, /) = \{(k_2, \varepsilon)\};$$
$$p(k_2, (, () = \{(k_2, \varepsilon)\}, p(k_2,),)) = \{(k_2, \varepsilon)\};$$
$$p(k_2, \dashv, \varepsilon) = \{(k_3, \varepsilon)\}.$$

В исходном состоянии автомата k_1 содержимое стека автомата принимает вид $E \dashv$, и переход к состоянию k_3 возможен лишь в том случае, если стек пуст (содержит лишь символ $\dashv$), следовательно, эмуляция левостороннего вывода осуществляется автоматом лишь в состоянии k_2. В табл. 6.1 приведена последовательность конфигураций автомата M, принимающего арифметическое выражение вида $a \times (b + c)$.

В заключение заметим, что построить эффективный грамматический анализатор для разбора арифметического выражения с помощью определенного выше недетерминированного магазинного автомата M – задача непростая. Основная трудность заключается в том, что (как видно из определения магазинной функции p) автомат M имеет три ε-перемещения. Если $x \in T(M)$, то может быть построена некоторая последовательность конфигураций автомата M, гарантирующая, что данная строка x действительно принимается автоматом M; однако такой строке может соответствовать целый ряд корректных перемещений автомата M, не являющихся необходимыми для приема

Таблица 6.1

Невведенная часть входной строки	*Последовательность конфигураций*	
	Состояние	*Содержимое стека*
$a \times (b + c)$	k_1	$\dashv$
$a \times (b + c)$	k_2	$E \dashv$
$a \times (b + c)$	k_2	$T \dashv$
$a \times (b + c)$	k_2	$T \times F \dashv$
$a \times (b + c)$	k_2	$F \times F \dashv$
$a \times (b + c)$	k_2	$a \times F \dashv$
$\times (b + c)$	k_2	$\times F \dashv$
$(b + c)$	k_2	$F \dashv$
$(b + c)$	k_2	$(E) \dashv$
$b + c)$	k_2	$E) \dashv$
$b + c)$	k_2	$\neq + T) \dashv$
$b + c)$	k_2	$T + T) \dashv$
$b + c)$	k_2	$F + T) \dashv$
$b + c)$	k_2	$b + T) \dashv$
$+ c)$	k_2	$+ T) \dashv$
$c)$	k_2	$T) \dashv$
$c)$	k_2	$F) \dashv$
$c)$	k_2	$c) \dashv$
$)$	k_2	$) \dashv$
ε	k_2	$\dashv$
ε	k_3	ε

строки x. В следующих главах мы покажем, что при наложении определенных ограничений на грамматику, порождающую язык, последний приобретет ряд свойств, благодаря которым принимающий его недетерминированный магазинный автомат будет лишен подобных неприятных особенностей. В этом случае построение эффективного грамматического анализатора станет возможным.

ТЕОРЕМА 6.1

Для любого контекстно свободного языка L существует недетерминированный магазинный автомат M такой, что $L = T(M)$.

Доказательство

Пусть язык $L \subset T^*$ порожден контекстно свободной грамматикой $G = (K, T, P, S)$; определим автомат M следующим образом:

$M = (K, T, V, p, k_1, \dashv, \{k_3\})$, где

$K = \{k_1, k_2, k_3\}$;

$V = N \cup T \cup \{\dashv\}$, $\dashv \notin N \cup T$;

$p(k_1, \dashv, \varepsilon) = \{(k_2, S\dashv)\}$;

$p(k_2, A, \varepsilon) = \{(k_2, \alpha) | A \to \alpha$ – продукция из $P\}$ для любого $A \in N$;

$p(k_2, a, a) = \{(k_2, \varepsilon)\}$ для любого $a \in T$;

$p(k_2, \dashv, \varepsilon) = \{(k_3, \varepsilon)\}$.

Это определение гарантирует, что если $x \in T^*$, $\alpha \in (N \cup T)^*$, то $S \overset{*}{\Rightarrow} x\alpha$ – левосторонний вывод тогда и только тогда, когда $t((k_1, \dashv), x)$ содержит конфигурацию $(k_2, \alpha\dashv)$. Следовательно, $x \in L \Leftrightarrow S \overset{*}{\Rightarrow} x \Leftrightarrow (k_2, \dashv) \in t((k_2, \dashv), x) \Leftrightarrow$ $\Leftrightarrow (k_3, \dashv) \in t((k_1, \dashv), x) \Leftrightarrow x \in T(M)$.

Верна и обратная теорема, т. е. любой принимаемый недетерминированным магазинным автоматом язык является контекстно свободным языком. Однако для доказательства этого утверждения нам понадобится один очень важный результат. Пусть $M = (K, T, V, p, k_1, A_1, F)$ – произвольный недетерминированный магазинный автомат. Построим новый недетерминированный магазинный автомат $M' = (K', T, V', p', k_1', \dashv, F)$ такой, что $K' = K \cup \{k_1', k_2'\}$, где $k_1', k_2' \in K$, $V' = V \cup \{\dashv\}$ при $\dashv \notin V$. Функция p' задается теми же соотношениями, что и функция p, за исключением соотношений, соответствующих состояниям k_1' и k_2':

$p'(k_1', \dashv, \varepsilon) = \{(k_1, A_1\dashv)\}$;

$p'(k, A, \varepsilon) = \{(k_2', \varepsilon)\}$ для любых $A \in V', k \in F$;

$p'(k_2', A, \varepsilon) = \{(k_2', \varepsilon)\}$ для любого $A \in V'$;

$F' = \{k_2'\}$.

Первое из этих соотношений гарантирует занесение в стек символа $\dashv$ прежде чем автомат M начнет осуществлять какие-либо перемещения, а это означает, что даже если автомат M достигнет одного из состояний, являющихся элементами множества F', в стеке останется один элемент. Второе из этих соотно-

шений указывает на возможность перехода автомата M' в состояние k_2' без необходимого ввода одного из символов входного алфавита. Поскольку $k_2' \in F'$, то, очевидно, $T(M) = T(M')$. По определению автомат M имеет единственное конечное состояние, после перехода в которое в соответствии с третьим соотношением стек очищается.

ТЕОРЕМА 6.2

Если L – некоторый язык, принимаемый недетерминированным магазинным автоматом M, то он может быть принят и недетерминированным магазинным автоматом M', имеющим единственное конечное состояние, при переходе в которое (и только в этом случае) стек автомата очищается.

Сформулировав эту теорему, мы можем перейти к доказательству второго важного утверждения.

ТЕОРЕМА 6.3

Если $L = T(M)$, где M – некоторый недетерминированный магазинный автомат, то L – контекстно свободный язык.

Доказательство

Пусть L – множество строк, принимаемых некоторым недетерминированным магазинным автоматом $M = (\{k_1, k_2, \ldots, k_n\}, T, V, p, k_1, A_1, \{k_n\})$, обладающим указанными в теореме 6.2 свойствами, и пусть его состояния помечены $k_1, k_2, \ldots, k_n$, где k_1 – исходное, а k_n – конечное состояния автомата. Определим контекстно свободную грамматику G такую, что $T(M) = L(G)$.

Поставим в соответствие каждому символу $A \in V$ такой нетерминальный символ A^{ij} определяемой грамматики G, что из него могут быть выведены все строки, в результате ввода которых автомат переходит из состояния k_i в состояние k_j и одновременно из стека автомата извлекается символ A. В таком случае, $T(M)$ – язык, порождаемый грамматикой $G = (N, T, P, S)$, где $N = \{A^{ij} \mid A \in V,\ 1 \leqslant i, j \leqslant n\}$ $S = A_1^{1n}$, а множество продукций определяется следующим образом: пусть p – магазинная функция, тогда если $p(k_i, A, a)$, $k_i \in K$, $A \in V$, $a \in T \cup \{\varepsilon\}$ содержит $(k_j B_1 B_2 \ldots B_m)$, $k_j \in K$, то $A^{ij} \to a B_1^{in_1} B_2^{n_1 n_2} \ldots B_m^{n_{m-1} j}$ – элемент P для всех $1 \leqslant n_1, n_2, \ldots, n_{m-1} \leqslant n$; если $p(k_i, A, a)$, $k_i \in K$, $A \in V$, $a \in T \cup \{\varepsilon\}$ содержит (k_j, ε), $k_j \in K$, то $A^{ij} \to a$ – элемент P. Следовательно,

$$x \in T(M) \Leftrightarrow (k_n, \varepsilon) \in t((k_1, A_1), x) \Leftrightarrow A^{1n} \overset{*}{\Rightarrow} x \Leftrightarrow x \in L(G),$$

что и требовалось доказать.

Теперь можно определить контекстно свободный язык как язык, который может быть принят некоторым недетерминированным магазинным автоматом. Воспользуемся этим результатом и докажем ряд свойств контекстно свободных языков.

ТЕОРЕМА 6.4

Если L – контекстно свободный язык, а R – регулярное множество строк, то $L \cap R$ – контекстно свободный язык.

Доказательство

$L = T(M)$, где $M = (K, T, V, p, k_1, A_1, F)$ – некоторый недетерминированный магазинный автомат и $R = T(\tilde{M})$, где $\tilde{M} = (\tilde{K}, T, \tilde{t}, \tilde{k}_1, \tilde{F})$ – некоторый недетерминированный конечный автомат. Построим такой недетерминированный магазинный автомат $M \times \tilde{M}$, чтобы он принимал язык $L \cap R$. Прежде всего, перемещения, осуществляемые автоматом $M \times \tilde{M}$, соответствует, очевидно, одновременным перемещениям автоматов M и $\tilde{M}$, и строка x будет принята им в том и только том случае, если она может быть принята одновременно автоматами M и $\tilde{M}$. Формально автомат $M \times \tilde{M} = (K \times \tilde{K}, T, V, p', [k_1, \tilde{k}_1], A_1, F \times \tilde{F})$ [1], где магазинная функция p определяется следующими соотношениями: для $k, k' \in K$, $\tilde{k}, \tilde{k}' \in \tilde{K}$, $A \in V, a \in T, X \in V^*$

а) $([k', \tilde{k}'], X) \in p'([k, \tilde{k}], A, a) \Leftrightarrow (k', X) \in p(k, A, a)$ и $\tilde{k}' \in \tilde{t}(\tilde{k}, a)$.

б) $([k', \tilde{k}'], X) \in p'([k, \tilde{k}], A, \varepsilon) \Leftrightarrow (k', X) \in p(k, A, \varepsilon)$ и $\tilde{k}' = \tilde{k}$.

Методом индукции по числу перемещений можно показать, что

$([k, \tilde{k}], X) \in t_{M \times \tilde{M}}(([k_1, \tilde{k}_1], A_1), x) \Leftrightarrow (k, X) \in t_M((k_1, A_1), x)$ и $\in \tilde{t}(\tilde{k}_1, x)$,

следовательно, $x \in T(M \times \tilde{M}) \Leftrightarrow ([k, \tilde{k}], x) \in t_{M \times \tilde{M}}(([k_1, \tilde{k}_1], A_1), x)$ для некоторых $k \in F$, $\tilde{k} \in \tilde{F}$, $X \in V^* \Leftrightarrow (k, X) \in t_M((k_1, A_1), x)$, $k \in F$, $X \in V^*$ и $\tilde{k} \in \tilde{t}(\tilde{k}_1, x)$, $\tilde{k} \in \tilde{F} \Leftrightarrow$ $\Leftrightarrow x \in T(M)$ и $x \in T(\tilde{M})$. Таким образом, $T(M \times \tilde{M}) = T(M) \cap T(\tilde{M}) = L \cap R$.

ДЕТЕРМИНИРОВАННЫЕ МАГАЗИННЫЕ АВТОМАТЫ

Если любой конфигурации магазинного автомата соответствует единственное перемещение, то такой магазинный автомат называют *детерминированным магазинным автоматом*. Приведем теперь строгое определение детерминированного магазинного автомата: магазинный автомат $M = (K, T, V, p, k_1, A_1, F)$ называется детерминированным магазинным автоматом, если для любых $k \in K$, $a \in T$ и $A \in V$ выполняются следующие условия:

1) $p(k, A, a)$ имеет не более одного элемента;

2) $p(k, A, \varepsilon)$ имеет не более одного элемента;

3) если $p(k, A, \varepsilon) \neq \emptyset$, то $p(k, A, a) = \emptyset$ для любого $a \in T$.

Поясним последнее условие – если из некоторой конфигурации автомат может осуществить хотя бы одно ε-перемещение, то оно является единственным перемещением, которое он может из этой конфигурации осуществить.

Из приведенного определения детерминированного магазинного автомата можно сделать вывод, что правые части всех соотношений, задающих магазинную функцию такого автомата, определяют пустые множества или синглетоны. Ниже соотношения, правые части которых определяют пустые множества, мы не будем выделять, а потому можно утверждать, что магазинная функция любого детерминированного автомата задается соотношениями вида

$$f(k, A, a) = \{(k', X)\}$$

---

[1] Запись вида $[k, \tilde{k}]$ используется автором для обозначения такого состояния автомата $M \times \tilde{M}$, которое соответствует одновременно состоянию k автомата M и состоянию $\tilde{k}$ автомата $\tilde{M}$. – *Прим. ред.*

или, как это принято в теории формальных языков, соотношениями вида

$$f(k, A, a) = (k', X).$$

Язык, принимаемый детерминированным магазинным автоматом, называют *детерминированным контекстно свободным языком*, или просто *детерминированным языком*. К сожалению, не всякий контекстно свободный язык является детерминированным языком, тем не менее детерминированные языки составляют очень важный класс языков. Дело в том, что в классе детерминированных языков значительно упрощается решение проблемы выводимости. Синтаксические определения у большинства современных языков программирования сформулированы таким образом, что процесс компиляции написанных на них программ весьма прост, так как в каждом конкретном случае могут быть разработаны более или менее эффективные грамматические анализаторы. Ограничения же, накладываемые на указанные синтаксические определения требованием детерминированности языка программирования, являются при этом самыми слабыми (в следующих двух главах мы покажем, почему). Приведем несколько примеров и изучим еще несколько свойств детерминированных языков.

Сначала рассмотрим регулярные языки. Поскольку всякий регулярный язык может быть принят некоторым детерминированным конечным автоматом и поскольку всякий такой автомат может быть представлен детерминированным магазинным автоматом, не использующим свой стек, то, очевидно, всякий регулярный язык является детерминированным языком. Обратное неверно: не всякий детерминированный контекстно свободный язык является регулярным языком. Для иллюстрации сказанного обратимся к примеру, и рассмотрим язык $\{a^n b^n | n \geqslant 1\}$, не являющийся регулярным языком, и детерминированный магазинный автомат $M = (\{k_1, k_2, k_3\}, \{a, b\}, \{a, \dashv\}, p, k_1, \dashv, \{k_3\})$, где

$$p(k_1, \dashv, a) = (k_1, a\dashv);$$
$$p(k_1, a, a) = (k_1, aa);$$
$$p(k_1, a, b) = (k_2, \varepsilon);$$
$$p(k_2, a, b) = (k_2, \varepsilon);$$
$$p(k_2, \dashv, \varepsilon) = (k_3, \varepsilon).$$

Находясь в состоянии k_1, автомат осуществляет занесение в стек всех вводимых входных символов a до тех пор, пока не будет введен символ b, в результате чего автомат перейдет в состояние k_2. В состоянии k_2 автомат ставит в соответствие каждому введенному символу b извлеченный из стека символ a до тех пор, пока в стеке не останется только символ $\dashv$, после чего автомат переходит в конечное состояние k_3.

Любой детерминированный язык, очевидно, является контекстно свободным языком, так как по определению может быть принят некоторым детерминированным магазинным автоматом, который, очевидно, является ограничением недетерминированного магазинного автомата. Из упр. 5.12 читатель должен был узнать, что существует контекстно свободный язык

$L \subset T^*$ такой, что его дополнение в $\bar{L} = T^* \backslash L$ не является контекстно свободным языком. Покажем, что дополнение всякого детерминированного контекстно свободного языка в T^* является детерминированным контекстно свободным языком. Доказательство этого факта весьма сложно, и поэтому, читая книгу первый раз, его можно пропустить, достаточно только хорошо понимать один из промежуточных результатов доказательства, из которого прямо вытекает возможность существования таких контекстно свободных языков, которые не являются детерминированными языками.

Пусть $L \subset T^*$ – детерминированный язык, принимаемый детерминированным магазинным автоматом $M = (K, T, V, p, k_1, A_1, F)$. Построим детерминированный автомат $\bar{M}$ (при этом конечные и "не" конечные состояния, очевидно, поменяются ролями), который принимал бы язык $\bar{L} = T^* \backslash L$. На этом пути нас ожидает ряд непростых проблем, которые придется решить.

Первая из них заключается в том, что автомат M может иметь одно или несколько "тупиковых" состояний, т. е. может существовать $x \in T^*$ такой, что в результате ввода собственного префикса строки x, автомат M получит конфигурацию, из которой не существует ни одного перемещения, кроме, быть может, бесконечного числа ε-перемещений. В любом случае $x \in \bar{L}$, и надо быть чрезвычайно осторожным при построении автомата $\bar{M}$, если мы хотим, чтобы он принимал такие строки. С такого рода трудностями мы уже сталкивались при доказательстве теоремы 3.4, в которой утверждалось, что дополнение регулярного языка есть регулярный язык. Возникшая проблема может быть решена удачной модификацией автомата M. Построим некий детерминированный магазинный автомат M' такой, что $T(M) = T(M')$, но автомат M' всегда использует все символы введенной строки. Строго говоря, автомат M' совпадает с автоматом M за исключением того, что его стек содержит маркер нижней границы $\dashv$, а сам автомат имеет три дополнительных состояния $s, d, f \in K$. Итак, s – исходное состояние автомата M', в котором стек автомата содержит символ $\dashv \in V$, и существует лишь ε-перемещение, переводящее автомат M' в состояние k_1 (при этом в стек заносится символ A_1). Перейдя в состояние k_1, автомат M' эмулирует автомат M, осуществляя при этом ряд важных дополнительных действий. Если автомат M перешел в состояние, в котором не существует ни одного перемещения, то автомат M' осуществляет переход в фиктивное состояние d, игнорируя последующий ввод любых входных символов. Аналогично новое конечное состояние автомата M' используется для решения проблемы с бесконечным числом ε-перемещений автомата M. Пусть автомат M имеет конфигурацию c, в которой существует бесконечное число ε-перемещений автомата; тогда, очевидно, должна существовать конфигурация $c' = (k, AX)$, $k \in K$, $A \in V \cup \{\dashv\}$, $X \in (V \cup \{\dashv\})^*$, которую будет иметь автомат M' в результате осуществления конечного числа ε-перемещений, и при этом автомат M', имеющий конфигурацию c', может осуществить бесконечное число ε-перемещений, не извлекая из стека более ни одного элемента. После осуществления указанных ε-перемещений автомат может перейти или не перейти в конечное состояние. Определим автомат M' таким

образом, чтобы в первом случае он имел бы конфигурацию (f, AX), а во втором – конфигурацию (d, AX). И наконец, для того чтобы автомат M' использовал всю введенную строку, состояние f дополним ε-перемещением, в результате осуществления которого автомат M' переходит вновь в состояние d.

Приведем теперь формальное определение автомата M':

$M' = (K', T, V', p', s, \dashv, F')$, где

$K' = K \cup \{s, d, f\}$, $s, d, f \notin K$,

$V' = V \cup \{\dashv\}$, $\dashv \notin V$,

$F' = F \cup \{f\}$, а магазинная функция p' задается соотношениями

а) $p'(s, \dashv, \varepsilon) = (k_1, A_1 \dashv)$;

б) для всех $k \in K, A \in V$ и $a \in T$ таких, что $p(k, A, a) = p(k, A, \varepsilon) = \emptyset$, $p'(k, A, a) = (d, A)$;

в) $p'(d, A, a) = (d, A)$ для всех $A \in V', a \in T$;

г) для всех $k \in K$, $A \in V$ таких, что бесконечная последовательность ε-перемещений может быть осуществлена автоматом, имеющим начальную конфигурацию (k, A),

$$p'(k, A, \varepsilon) = \begin{cases} (d, A), & \text{если ни одно из этих перемещений не переводит автомат } M \text{ в конечное состояние;} \\ (f, A) & \text{в противном случае;} \end{cases}$$

д) $p'(f, A, \varepsilon) = (d, A)$ для любых $A \in V'$;

е) если $p'(k, A, a)$ не определена предыдущими соотношениями, то $p'(k, A, a) = p(k, A, a)$, $k \in K$, $A \in V$, $a \in T \cup \{\varepsilon\}$.

Предоставляем читателю самостоятельно убедиться в том, что определенный таким образом автомат M действительно использует всю введенную строку и $T(M) = T(M')$. Особое внимание советуем обратить на то, как была получена указанная конструкция. В частности, может возникнуть вопрос: можно определить, является ли бесконечной последовательность ε-перемещений в конфигурации (k, a)? Если да, то есть ли среди них ε-перемещения, переводящие автомат в конечное состояние? На оба эти вопроса следует ответить утвердительно (см. упр. 6.11).

Рассмотрим еще одну проблему: после ввода строки $x \in T^*$ автомат M' осуществляет несколько ε-перемещений. Допустим, что в результате осуществления части из них автомат M' переходит в конечное состояние, а в результате осуществления остальных в "не" конечное состояние и обмен ролей конечных и "не" конечных состояний ничего не изменит в этой ситуации. Для решения этой проблемы построим еще один автомат – детерминированный конечный автомат M'' такой, что $T(M'') = T(M') = T(M)$. Автомат M'' имеет те же состояния, что и автомат M', но помечены они следующим образом: если в результате последнего непустого ввода автомат M' перешел в конечное состояние, то соответствующее состояние автомата M'' помечается так же, как и состояние автомата M', но с присоединением индекса 1; в противном случае к обозначению состояния автомата M' присоединяется индекс 2, и полученная таким образом запись является обозна-

чением соответствующего состояния автомата M''. Приведем формальное определение автомата M:

$M' = (K',\ T,\ V',\ p',\ s',\ \dashv,\ F')$,
$M'' = (K'', T, V', p'', s'', \dashv, F'')$, где
$K'' = \{k_i | k \in K', i = 1 \text{ или } 2\}$;

$$s'' = \begin{cases} s_1, \text{ если } \varepsilon \in T(M), \\ s_2 \text{ в противном случае;} \end{cases}$$

$F'' = \{k_1 | k \in K'\}$,

а магазинная функция p задается следующими соотношениями:

а) для $k \in K'$, $A \in V'$, если $p'(k, A, \varepsilon) = (k', X)$, $k' \in K'$, $X \in (V')^*$, то
$p''(k_1, A, \varepsilon) = (k_1', X)$,

$$p''(k_2, A, \varepsilon) = \begin{cases} (k_1', X) \text{ при } k' \in F', \\ (k_2', X) \text{ в противном случае;} \end{cases}$$

б) для $k \in K'$, $A \in V'$, $a \in T$, если $p'(k, A, a) = (k', X)$, $k' \in K'$, $X \in (V')^*$, то

$$p''(k_1, A, a) = p''(k_2, A, a) = \begin{cases} (k_1', X) \text{ при } k' \in F', \\ (k_2', X) \text{ в противном случае.} \end{cases}$$

Теперь, наконец, стало возможным поменять ролями конечные и "не" конечные состояния автомата M'' и получить, таким образом, детерминированный магазинный автомат $\bar{M}$, принимающий язык $\bar{L}$.

ТЕОРЕМА 6.5

Если $L \subset T^*$ – детерминированный язык, то $\bar{L} = T^* \setminus L$ – детерминированный язык.

ТЕОРЕМА 6.6

а) Любой регулярный язык является детерминированным языком, обратное неверно.

б) Любой детерминированный язык является контекстно свободным языком, обратное неверно.

Свойства детерминированных языков отличаются от свойств контекстно свободных языков, например класс детерминированных языков замкнут относительно операции дополнения, т. е. если язык L принадлежит к классу детерминированных языков, то язык $\bar{L}$ также принадлежит этому классу. В табл. 6.2 приведены основные свойства некоторых классов языков.

Таблица 6.2

Операция	Замкнутость относительно операции		
	Регулярное множество	Детерминированный язык	Контекстно свободный язык
Объединение	Да	Нет	Да
Конкатенация	"	"	"
Дополнение	"	Да	Нет

Окончание табл. 6.2

Операция	Замкнутость относительно операции		
	Регулярное множество	Детерминированный язык	Контекстно свободный язык
Пересечение	Да	Нет	Нет
Замыкание Клини	"	"	Да
Пересечение с регулярным множеством	"	Да	"
Гомоморфизм	"	Нет	"

УПРАЖНЕНИЯ

6.1. Пусть входная строка имеет вид 012345, S – стек. Какая из следующих строк может быть получена в результате последовательного использования операций "занесение в стек" и "извлечение из стека"?

а) 543210, б) 534210, в) 431250, г) 415320, д) 542301.

6.2. Постройте детерминированные магазинные автоматы и покажите с их помощью, что языки $\{wcw^r \mid w \in \{a, b\}^*\}$, $\{a^i b^i \mid j > i\}$ – детерминированные языки.

6.3. Докажите, что если L – детерминированный, а R – регулярный языки, то $L \cap R$ – детерминированный язык.

6.4. Пусть недетерминированный магазинный автомат $M(K, T, V, p, k_1, A_1, F)$ принимает язык $E(M) \subset T^*$, где $E(M) = \{x \mid$ существует $k \in K$ такое, что $(k, \varepsilon) \in t_m((k_1, A_1), x)\}$. Покажите, что $E(M)$ – контекстно свободный язык (из теоремы 6.2 следует, что обратное также верно).

6.5. Постройте недетерминированный магазинный автомат, принимающий язык, порождаемый грамматикой, имеющей продукции вида

$S \to aA \mid aBB,$
$A \to Ba \mid Sb,$
$B \to bAS \mid \varepsilon.$

6.6. Пусть L – детерминированный язык. Покажите, что $\min \{x \mid x \in L$ и не существует $w \in L$ – собственного префикса $x\}$ – также детерминированный язык.

6.7. Используя то, что язык $\{a^i b^j c^k \mid i \neq j$ и $j \neq k\}$ не является контекстно свободным языком, докажите, что $\{a^i b^j c^k \mid i = j$ или $j = k\}$ не является детерминированным языком, т. е. существуют два детерминированных языка L_1 и L_2 такие, что $L_1 \cup L_2$ не является детерминированным языком.

6.8. Используя результат упр. 6.7, покажите, что класс детерминированных языков незамкнут относительно операции пересечения.

6.9. Покажите, что любой детерминированный контекстно свободный язык может быть принят некоторым детерминированным магазинным автоматом M таким, что ни на одном из перемещений автомат не осуществляет извлечения из стека более двух символов.

6.10. Покажите, что каждый детерминированный контекстно свободный язык $L \subset T^*$ принимается некоторым детерминированным магазинным автоматом $M = (K, T, V, p, k_1, A_1, F)$ таким, что если осуществляемое автоматом перемещение удовлетворяет условию

$$p(k, A, a) = (k', X), k, k' \in R, a \in T \cup \{\varepsilon\}, A \in V, X \in V^*,$$

то оно специфицирует одну из следующих операций:

а) извлечение из стека одного символа, т. е. $X = \varepsilon$;

б) занесение в стек одного символа, т. е. $X = BA$, $A, B \in V$;

в) содержимое стека не изменяется, т. е. $X = A$;

6.11. Рассмотрим детерминированный магазинный автомат $M = (\{k_1, \ldots, k_n\}, T\{A_1 \ldots A_m\}, p, k_1, A_1, F)$, удовлетворяющий требованиям упр. 6.10, определим булевы трехмерные массивы B_1, B_2, B_3 следующим образом:

$B_1 [i, j, k] = true \Leftrightarrow (k_j, A_k) \in t_M((k_i, A_k), \varepsilon)$,

$B_2 [i, j, k] = true \Leftrightarrow (k_j, \varepsilon) \in t_M((k_i, A_k), \varepsilon)$,

$B_3 [i, j, k] = true \Leftrightarrow$ существует $X \in V^* | (k_j, X) \in t_M((k_i, A_k), \varepsilon)$.

Разработайте итерационный алгоритм построения B_1, B_2 и B_3.

ГЛАВА 7

СИНТАКСИЧЕСКИЙ АНАЛИЗ "СВЕРХУ ВНИЗ"

Iam nova progeniec caelo demittitur alto

(Все новое приходит к нам свыше)

Вергилий "Эклоги"

В гл. 6 мы показали, что любой контекстно свободный язык может быть принят некоторым недетерминированным магазинным автоматом. Из недетерминированности указанного автомата, вообще говоря, следует, что построенный на его основе синтаксический анализатор контекстно свободного языка может осуществлять возврат к предыдущему состоянию в процессе функционирования. Сам алгоритм синтаксического разбора определяет вполне однозначный процесс, поэтому при возникновении необходимости выбора одной из возможных логических ветвей, алгоритм выбирает и в дальнейшем следует всегда единственной из возможных логических ветвей. Это означает, что, обнаружив ошибку выбора, необходимо вернуться к моменту осуществления выбора, вновь осуществить выбор и выполнить другое перемещение. Однако если на порождающую язык грамматику были наложены определенные ограничения, а автомат эффективно использует свой стек, то становится корректным применение ряда хорошо изученных и отработанных приемов, позволяющих построить эффективный синтаксический анализатор для такого языка. На практике все реальные проекты языков программирования, безусловно, содержат все необходимые ограничения, поскольку они получат практическую ценность лишь в том случае, если будут поддержаны достаточно эффективными компиляторами.

При решении задачи синтаксического разбора строки вида $x = a_1 a_2 \ldots a_n \in L(G)$ используют, как правило, один из двух основных методов. Первый из них заключается в построении дерева вывода, корень которого помечен символом S, а листьями являются узлы, помеченные символами

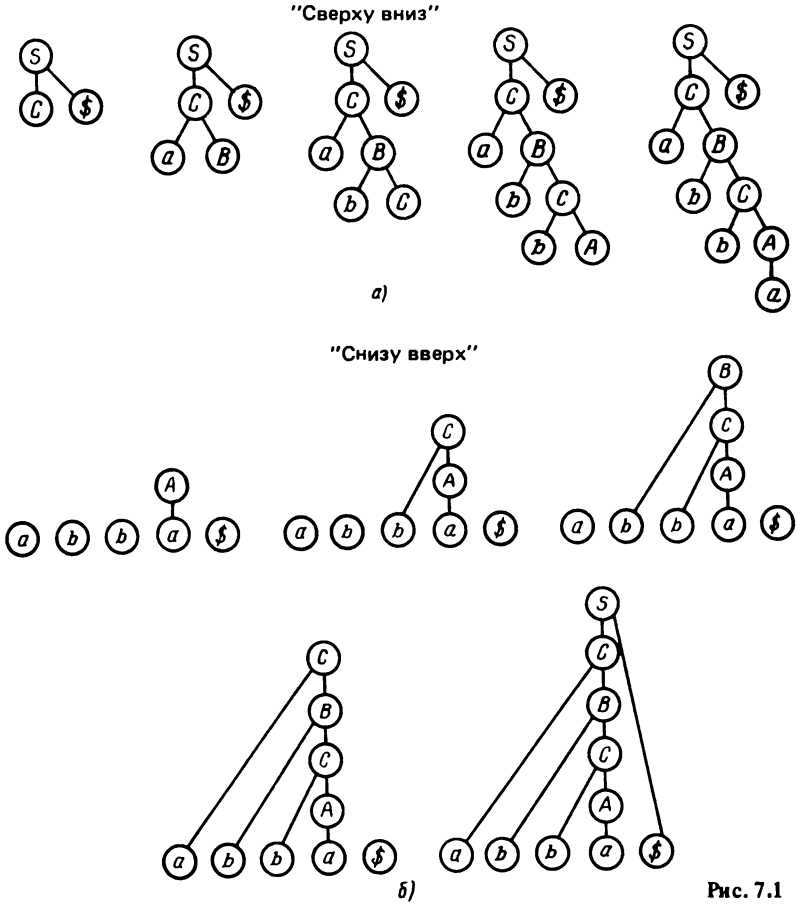


Рис. 7.1

$a_1, a_2, \ldots, a_n$. Второй прием заключается в *восходящем* построении дерева, листьями которого являются узлы, помеченные $a_1, a_2, \ldots, a_n$, а корнем – узел, помеченный символом S. На рис. 7.1 применение обоих методов проиллюстрировано на примере строки $abba\$$ и контекстно свободной грамматики G_1, имеющей продукции вида

$S \to C\$$,

$C \to bA \mid aB$,

$A \to a \mid aC$,

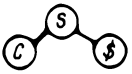
$B \to b \mid bC$.

В этой главе мы рассмотрим так называемый LL-разбор, который относится к методам синтаксического разбора типа "сверху вниз", отложив знакомство с методами синтаксического разбора "снизу вверх" до следующей главы. Для успешного LL-разбора предложений некоторого языка

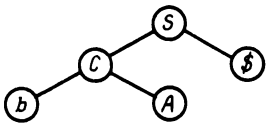
на порождающую его грамматику должны быть наложены строгие ограничения. И хотя практически это означает, что воспользоваться методом LL-разбора можно лишь на подмножестве класса детерминированных языков, он представляет значительный интерес, так как заложенные в нем принципы использованы в большинстве современных компиляторов.

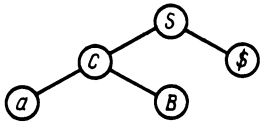
$LL(K)$-ГРАММАТИКИ

Снова рассмотрим грамматику G_1. Дерево синтаксического разбора "сверху вниз" строки $abba\$$ показано на рис. 7.1, *а*. Рассмотрим подробно его построение. Корень дерева помечен символом S, и, следовательно, единственной продукцией грамматики G_1, которая может быть использована в данном случае, является продукция $S \to C\$$; в результате будет получено дерево

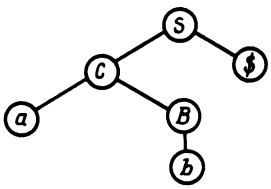


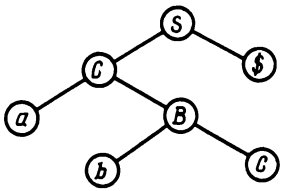
Перейдем теперь к узлу C. Мы можем воспользоваться продукциями $C \to bA$ и $C \to aB$:





Поскольку рассматриваемая строка $abba\$$ начинается с символа a, воспользуемся второй из этих продукций, или, обобщая это рассуждение, скажем, что "выбрана будет та продукция, правая часть которой начинается с очередного введенного входного символа". Итак, один из листьев дерева, которое мы пытаемся построить, уже помечен символом a. Убедимся теперь в том, что $B \overset{*}{\Rightarrow} bba$. Воспользуемся для этого продукцией $B \to b|bC$. В результате получим два дерева:





На этот раз знать следующий введенный символ входной строки недостаточно для выбора одной из этих продукций. Тем не менее, если нам станут известны два очередных символа входной строки и если это символы $b\$$, то следует воспользоваться первой продукцией, а если это символы ba или bb, то второй продукцией. В данном случае очередные символы входной строки — это символы bb, поэтому следует выбрать продукцию $B \to bC$. Убедимся в том, что $C \overset{*}{\Rightarrow} ba$, для чего, согласно следующему символу входной строки, воспользуемся продукцией $C \to bA$, а затем, согласно двум очеред-

ным символам входной строки, воспользуемся продукцией $A \to a$. В результате будет получена левосторонняя схема вывода вида

$$S \Rightarrow C\$ \Rightarrow aB\$ \Rightarrow abC\$ \Rightarrow abba\$ \Rightarrow abba\$.$$

Грамматика G_1 является примером $LL(2)$-грамматики, иначе говоря, для однозначного определения дерева вывода достаточно знать не более двух очередных символов входной строки на каждом этапе построения дерева. Вообще, $LL(k)$-грамматикой называют такую грамматику, что для любой ее сентенциальной формы

$$wA\gamma, w \in T^*, A \in N, \gamma \in (N \cup T^*),$$

полученной в результате некоторого левостороннего вывода для однозначного выбора продукции, имеющей в левой части символ A, достаточно знать k очередных символов входной строки. Аббревиатура "LL" в названии грамматики означает "левосторонний ввод – левосторонний вывод". Заметим, что, строго говоря, вид следующей используемой для вывода продукции определяется не только нетерминальным символом A и k очередными символами входной строки, но также возможно и видом сентенциальных форм $w \in T^*$ и $\gamma \in (N \cup T^*)$. Если вид следующей используемой для вывода продукции однозначно определяется символом A и очередными k символами входной строки, то такая $LL(k)$-грамматика называется *строгой* $LL(k)$-грамматикой.

Введем следующее определение:

$\mathrm{FIRST}_k : T^* \to T^*$, где

$$\mathrm{FIRST}_k(x) = \begin{cases} x, & \text{если } |x| \leqslant k \\ y, & \text{если } x = y,\ y \in T^k,\ z \in T^*. \end{cases}$$

Определим функцию FIRST_k на некотором языке L :

$$\mathrm{FIRST}_k(L) = \{\mathrm{FIRST}_k(x) | x \in L\} \text{ для любого } L \subset T^*.$$

Приведем теперь формальное определение. Пусть $G = (N, T, P, S)$ - контекстно свободная грамматика, ее называют $LL(k)$-грамматикой, если две любые левосторонние схемы вывода

$$S \overset{*}{\Rightarrow} wA\gamma \Rightarrow w\alpha\gamma \overset{*}{\Rightarrow} wy \in T^*,$$
$$S \overset{*}{\Rightarrow} wA\gamma \Rightarrow w\beta\gamma \overset{*}{\Rightarrow} wz \in T^*$$

связаны соотношением

если $\mathrm{FIRST}_k(y) = \mathrm{FIRST}_k(z)$, то $\alpha = \beta$.

Кроме этого, грамматика G называется строгой $LL(k)$-грамматикой, если любые две левосторонние схемы вывода

$$S \overset{*}{\Rightarrow} wA\gamma \Rightarrow w\alpha\gamma \overset{*}{\Rightarrow} wy \in T^*,$$
$$S \overset{*}{\Rightarrow} xA\delta \Rightarrow x\beta\delta \overset{*}{\Rightarrow} xz \in T^*$$

связаны соотношением: если $\mathrm{FIRST}_k(y) = \mathrm{FIRST}_k(z)$, то $\alpha = \beta$.

В общем случае определить, является ли некоторая грамматика однозначной, невозможно, однако если можно показать, что рассматриваемая грам-

матика является строгой $LL(k)$-грамматикой, то из приведенной ниже теоремы следует, что грамматика однозначная. На самом деле, теорема справедлива и для случая $LL(k)$-грамматики (см. упр. 7.7).

ТЕОРЕМА 7.1

Если $G = (N, T, P, S)$ – строгая $LL(k)$-грамматика ($k \geqslant 1$), то G – однозначная грамматика.

Доказательство

Предположим, что G – неоднозначная грамматика; тогда, очевидно, существует строка $w \in L(G)$, которая может быть выведена из S двумя различными способами.

Пусть T_1 и T_2 – два различных дерева вывода, соответствующие двум различным левосторонним схемам вывода строки $w = a_1 a_2 \ldots a_n$ из нетерминального символа S. Оба дерева имеют общий корень S и внешние узлы a_1, $a_2, \ldots, a_n$, но, по предположению, не совпадают. Осуществим одновременный обход узлов двух деревьев, используя для этого следующую рекурсивную процедуру: обход начинается с корня дерева, обход узлов данного уровня осуществляется слева направо. Переходя таким образом с одного уровня на другой, мы найдем два таких узла деревьев T_1 и T_2, что они помечены различными символами, например B и C (рис. 7.2). Очевидно, эти узлы не являются корнями соответствующих деревьев, так как оба дерева имеют один корень – узел S. Из этого следует, что они, очевидно, имеют родительские узлы и последние совпадают (пусть это будет узел, помеченный символом A).

Дерево T_1 соответствует левосторонней схеме вывода

$$S \overset{*}{\Rightarrow} xA\gamma_1 \Rightarrow x\alpha B\beta_1\gamma_1 \overset{*}{\Rightarrow} xy_1\gamma_1 \overset{*}{\Rightarrow} xy_1z_1 = w,$$

а дерево T_2 схеме вывода

$$S \overset{*}{\Rightarrow} xA\gamma_2 \Rightarrow x\alpha C\beta_2\gamma_2 \overset{*}{\Rightarrow} xy_2\gamma_2 \overset{*}{\Rightarrow} xy_2z_2 = w.$$

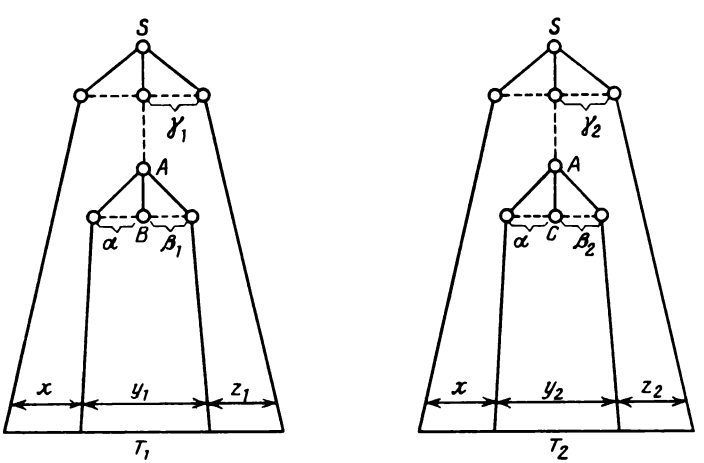


Рис. 7.2

Поскольку $\mathrm{FIRST}_k(y_1, z_1) = \mathrm{FIRST}_k(y_2, z_2)$ и $\alpha B\beta_1 \neq \alpha C\beta_2$, то, очевидно, G не может быть строгой $LL(k)$-грамматикой, следовательно, теорема доказана.

По определению всякая строгая $LL(k)$-грамматика является $LL(k)$-грамматикой, однако для $k > 1$ существуют грамматики, которые, являясь $LL(k)$-грамматиками, в то же время не являются строгими $LL(k)$-грамматиками. Следует заметить, что при $k = 1$ определения $LL(k)$-грамматики и строгой $LL(k)$-грамматики эквивалентны. Кроме того, как будет показано ниже, класс языков, порождаемых $LL(k)$-грамматиками, в точности совпадает с классом языков, порождаемых строгими $LL(k)$-грамматиками для всех $k \geqslant 1$.

ТЕОРЕМА 7.2

Контекстно свободная грамматика $G = (N, T, P, S)$ является строгой $LL(1)$-грамматикой $\Leftrightarrow G$ является $LL(1)$-грамматикой.

Доказательство

Необходимость этого условия теоремы следует непосредственно из определения строгой $LL(1)$-грамматики.

Докажем достаточность этого условия.

Пусть G – $LL(1)$-грамматика, не являющаяся строгой $LL(1)$-грамматикой; тогда среди ее продукции есть две несовпадающие продукции вида $A \to \alpha$ и $A \to \beta$ такие, что для некоторых $w, x, y_1, y_2, z_1, z_2 \in T^*$ и $\gamma, \delta \in (N \cup T)^*$, и существуют две несовпадающие левосторонние схемы вывода

$S \overset{*}{\Rightarrow} wA\gamma \Rightarrow w\alpha\gamma \overset{*}{\Rightarrow} wy_1\gamma \overset{*}{\Rightarrow} wy_1y_2$ и $S \overset{*}{\Rightarrow} xA\delta \Rightarrow x\beta\delta \overset{*}{\Rightarrow} xz_1\delta \overset{*}{\Rightarrow} xz_1z_2$, где

$\mathrm{FIRST}_1(y_1, y_2) = \mathrm{FIRST}_1(z_1, z_2)$.

Необходимо показать, что G не является $LL(1)$-грамматикой. Последнее утверждение может быть справедливым в одном из следующих двух случаев:

а) $y_1 = z_1 = \varepsilon$;

б) $y_1 z_1 \neq \varepsilon$.

Случай а) очевиден, а потому перейдем сразу к случаю б). Без потери общности можно предположить, что $y_1 \neq \varepsilon$. Следовательно, $\mathrm{FIRST}_1(y_1, y_2) = \mathrm{FIRST}_1(y_1) = \mathrm{FIRST}_1(z, z_2)$. Но в таком случае две левосторонние схемы вывода

$S \overset{*}{\Rightarrow} xA\delta \Rightarrow x\alpha\delta \overset{*}{\Rightarrow} xy_1\delta \Rightarrow xy_1z_2$ и $S \overset{*}{\Rightarrow} xA\delta \Rightarrow x\beta\delta \overset{*}{\Rightarrow} xz_1\delta \Rightarrow xz_1z_2$

удовлетворяют условию

$\mathrm{FIRST}_1(y_1) = \mathrm{FIRST}_1(z_1, z_2)$.

Однако $\alpha \neq \beta$ и, следовательно, G не является $LL(1)$-грамматикой. Это противоречие и доказывает теорему.

Следует заметить, что частный случай $LL(k)$-грамматики при $k = 1$ представляет большой интерес при разработке компиляторов. Дело в том, что очень часто разработчики языков программирования, формулируя синтаксические определения будущего языка, имеют в виду $LL(1)$-грамматику. Даже если разработчик языка программирования не позаботился об этом,

то разработчик компилятора с этого языка программирования наверняка исправит оплошность коллеги и соответствующим образом конвертирует сформулированные синтаксические определения. Существует даже созданный специально для этих целей конвертор SID (Syntax Improving Device) (детальное описание конвертора SID можно найти в журнале Computer Journal за 1968 год), который автоматически конвертирует произвольную входную грамматику в эквивалентную $LL(1)$-грамматику, если это возможно. Существуют, однако, некоторые синтаксические особенности, присущие любому языку программирования, которые не могут быть сформулированы в виде синтаксических определений языка, порожденного $LL(1)$-грамматикой. Они обрабатываются отдельно. Итак, если основная часть синтаксиса некоторого языка программирования представляет собой набор синтаксических определений языка, порожденного $LL(1)$-грамматикой, то для синтаксического разбора указанных синтаксических определений с помощью так называемого метода рекурсивного спуска, можно легко разработать (или даже автоматически сгенерировать) эффективный анализатор. Осуществить разбор других синтаксических определений языка анализатором также можно, но с помощью более хитроумных методов. Примерно таким образом разработаны почти все существующие компиляторы с языка программирования Паскаль. Метод рекурсивного спуска не следует рассматривать только как средство построения синтаксических анализаторов для языка, порожденных $LL(1)$-грамматикой, фактически он представляет собой формализацию принципов, использование которых дает нам возможность строить эффективные и одновременно элегантные синтаксические анализаторы в чрезвычайно широком классе детерминированных грамматик. Детали этой проблемы мы обсудим позднее.

Сформулируем теперь теорему, которая позволит нам построить алгоритм для определения того, является произвольная контекстно свободная грамматика G строгой $LL(1)$-грамматикой или нет. Для этого, прежде всего, понадобится определить контекстно свободную грамматику G' такую, что $L(G')=\{x\$ | x \in L(G)\}$, где $\$$ – специальный символ, который будет введен сразу же за последним входным символом, или, как его еще называют, "маркер конца ввода". Итак, если $G = (N, T, P, S)$ – произвольная контекстно свободная грамматика, то *пополненной грамматикой* называют грамматику $G' = (N', T', P', S')$, где

$$N' = N \cup \{S'\} \mid S' \notin N,$$

$$T' = T \cup \{\$\} \mid \$ \notin T,$$

$$P' = P \cup \{S' \to S\$\}.$$

Понятно, что $L(G') = \{x\$ | x \in L(G')\}$. Определим теперь на множестве $N' \cup T$ следующие функции:

$$\text{EMPTY(X)} = \begin{cases} true, & \text{если } X \overset{*}{\Rightarrow} \varepsilon, \\ false, & \text{в противном случае.} \end{cases}$$

$\text{FIRST(X)} = \{a | a \in T$ и $X \overset{*}{\Rightarrow} ax$ для некоторого $x \in (T')^*\}$.

$\text{FOLLOW(X)} = \{a | a \in T'$ и $S' \overset{*}{\Rightarrow} xXay$ для некоторого $x \in T^*$ и $y \in (T')^*\}$.

Таким образом, с каждым $X \in N' \cup T'$ оказывается ассоциированным два подмножества множества T' : FIRST(X) и FOLLOW(X).

Доопределим теперь функцию EMPTY на множестве $(N' \cup T')^*$ следующим образом:

EMPTY$(\varepsilon) = true$ и

EMPTY$(X\alpha)$ = EMPTY(X) *and* EMPTY(α), $X \in N' \cup T'$, $\alpha \in (N' \cup T')^*$.

Таким образом, если $\gamma \in (N' \cup T')^*$, то EMPTY$(\gamma) = true \Leftrightarrow \gamma \overset{*}{\underset{G'}{\Rightarrow}} \varepsilon \Leftrightarrow \gamma \overset{*}{\underset{G}{\Rightarrow}} \varepsilon$.

И наконец, определим еще одну функцию:

LOOKAHEAD$(A \to X_1 X_2 \ldots X_n)$ =
$= \cup \{\text{FIRST}(X_i) \mid 1 \leqslant i \leqslant n$ *and* EMPTY$(X_1 X_2 \ldots X_{i-1})\}$
$\cup$ *if* EMPTY$(X_1 X_2 \ldots X_n)$ *then* FOLLOW(A) *else* $\varnothing$.

Итак, сформулируем теорему.

ТЕОРЕМА 7.3

Контекстно свободная грамматика $G = (N, T, P, S)$ является строгой $LL(1)$-грамматикой $\Leftrightarrow$ для любой пары несовпадающих продукций грамматики вида $A \to \alpha$ и $A \to \beta$, имеющих одинаковые левые части, справедливо следующее утверждение:

LOOKAHEAD$(A \to \alpha) \cap$ LOOKAHEAD$(A \to \beta) = \varnothing$.

Для иллюстрации использования только что сформулированной теоремы, рассмотрим следующий пример: $G_2 = (N, T, P, S)$ – контекстно свободная грамматика, где $N = \{S, A, B\}$, $T = \{a, b, c, d\}$, а множество P содержит продукции вида

$S \to BA \mid AAd$,

$A \to a \mid \varepsilon$,

$B \to bA \mid cB$.

У соответствующей ей пополненной грамматики $N' = N \cup \{S'\} = \{S, A, B, S'\}$, $T' = T \cup \{\$\} = \{a, b, c, d, \$\}$, а множество продукций P' имеет следующие элементы:

$S' \to S\$$,

$S \to BA \mid AAd$,

$A \to a \mid \varepsilon$,

$B \to bA \mid cB$.

Кроме того, EMPTY$(A) = true$, но EMPTY$(X) = false$ для всех X, не совпадающих с A. Как и для всякой пополненной грамматики, в данном случае очевидно, что для любых $a_i \in T'$, FIRST$(a_i) = \{a_i\}$ и FOLLOW$(S') =$ $=$ FOLLOW$(\$) = \emptyset$. Кроме того, нетерминальный символ S' не входит в правую часть ни одной из иродукций, и, следовательно, значение FIRST(S') не участвует в вычислении функции LOOKAHEAD, а значит, можно ограничиться рассмотрением следующих выражений:

$\mathrm{FIRST}(A_i)$, $A_i \in N$ и $\mathrm{FOLLOW}(X)$, $X \in N \cup T$.

Значения функции $\mathrm{FIRST}(A_i)$, $A_i \in N$ можно представить как систему уравнений, построенную в соответствии со следующим правилом:

$$\mathrm{FIRST}(A_i) = \bigcup \{\mathrm{FIRST}(X_{ij}) | \text{ существует продукция } A_i \to X_{i1}X_{i2}\ldots X_{in} \text{ и } (j = 1 \text{ или } \mathrm{EMPTY}(X_{i1}X_{i2}\ldots X_{ij-1}))\}.$$

В данном случае такая система уравнений записывается следующим образом:

$$\mathrm{FIRST}(S) = \mathrm{FIRST}(B) \cup \mathrm{FIRST}(A) \cup \mathrm{FIRST}(A) \cup \{d\} = \mathrm{FIRST}(B) \cup \mathrm{FIRST}(A) \cup \{d\};$$

$$\mathrm{FIRST}(A) = \mathrm{FIRST}(a) = \{a\};$$

$$\mathrm{FIRST}(B) = \mathrm{FIRST}(b) \cup \mathrm{FIRST}(c) = \{b, c\}.$$

В общем случае найти решение этой системы уравнений можно с помощью итерационного метода, описанного в конце гл. 5. Однако в данном случае решение очевидно и приведено в табл. 7.1.

Таблица 7.1

Содержимое множества $N' \cup T'$	FIRST-множество	FOLLOW-множество
$	{$}	∅
a	{a}	{a, d, $}
b	{b}	{a, $}
c	{c}	{b, c}
d	{d}	{$}
S'	{a, b, c, d}[1]	∅
S	{a, b, c, d}	{$}
A	{a}	{a, d, $}
B	{b, c}	{a, $}

[1] $\mathrm{FIRST}(S') = \mathrm{FIRST}(S) \cup$ *if* $\mathrm{EMPTY}(S)$ *then* {$} *else* ∅.

Аналогичным образом может быть описано и множество значений функции FOLLOW. В этом случае, если $X \in N \cup T$, то искомое множество значений функции FOLLOW является решением следующей системы уравнений:

$$\mathrm{FOLLOW}(X) = \{\mathrm{FOLLOW}(A_i) | \text{ существует продукция вида } A_i \to \alpha X \beta \text{ и } \mathrm{EMPTY}(\beta)\} \cup \{\mathrm{FIRST}(X_{ij}) | \text{ существует продукция } A_i \to \alpha X X_{i1} X_{i2}\ldots X_{in} \text{ и } j = 1 \text{ или } \mathrm{EMPTY}(X_{i1}X_{i2}\ldots X_{ij-1})\}.$$

И, следовательно, для данного примера справедлива следующая система уравнений:

$$\text{FOLLOW}(a) = \text{FOLLOW}(A);$$

$$\begin{aligned}\text{FOLLOW}(b) &= \text{FOLLOW}(B) \cup \text{FIRST}(A) \\ &= \text{FOLLOW}(B) \cup \{a\};\end{aligned}$$

$$\text{FOLLOW}(c) = \text{FIRST}(B) = \{b, c\};$$

$$\text{FOLLOW}(d) = \text{FOLLOW}(S);$$

$$\text{FOLLOW}(S) = \text{FIRST}(\$) = \{\$\};$$

$$\begin{aligned}\text{FOLLOW}(A) &= \text{FOLLOW}(S) \cup \text{FOLLOW}(B) \\ &\quad \cup \text{FIRST}(A) \cup \text{FIRST}(d) \\ &= \text{FOLLOW}(S) \cup \text{FOLLOW}(B) \cup \{a, d\};\end{aligned}$$

$$\begin{aligned}\text{FOLLOW}(B) &= \text{FOLLOW}(S) \cup \text{FOLLOW}(B) \cup \text{FIRST}(A) \\ &= \text{FOLLOW}(S) \cup \text{FOLLOW}(B) \cup \{a\}.\end{aligned}$$

Эта система уравнений также может быть решена с помощью итерационного метода, описанного в гл. 5 (результат решения приведен в табл. 7.1).

Вычислив значения функций FIRST и FOLLOW для каждого элемента множества $N \cup T$, мы можем, наконец, найти область значений функции LOOKAHEAD, определенной на множестве продукций грамматики (результат приведен в табл. 7.2).

Таблица 7.2

Продукции	LOOKAHEAD-множество
$S \to BA$	$\{b, c\}$
$S \to AAd$	$\{a, d\}$
$A \to a$	$\{a\}$
$A \to \varepsilon$	$\{a, d, \$\}$
$B \to bA$	$\{b\}$
$B \to cB$	$\{c\}$

Поскольку LOOKAHEAD $(A \to a) \cap$ LOOKAHEAD $(A \to \varepsilon) = \Phi$ грамматика G_2 не является $LL(1)$-грамматикой.

Рассмотренный выше пример показывает, что всегда возможно определить, является ли произвольная контекстно свободная грамматика $LL(1)$-грамматикой или нет. К сожалению, однозначно ответить на вопрос, существует или нет $LL(1)$-грамматика, эквивалентная некоторой произвольной контекстно свободной грамматике, в общем случае невозможно, так как эта задача неразрешима. И потому из того, что произвольную контекстно свободную грамматику невозможно конвертировать в эквивалентную ей $LL(1)$-грамматику с помощью конвертора SID, не следует, что такой $LL(1)$-грамматики не существует.

РЕКУРСИВНЫЙ СПУСК

Если контекстно свободная грамматика G является $LL(k)$-грамматикой, то с помощью так называемого метода итерационного спуска можно построить синтаксический анализатор языка $L(G)$, который явно не использует стек, однако алгоритм его функционирования включает ряд рекурсивных процедур, которые обычно реализуют с помощью стека. Таким образом, несмотря на внешние отличия алгоритма синтаксического анализа, получен-

ного с помощью метода рекурсивного спуска, от алгоритма, описанного в гл. 6, принципиально они не отличаются.

В простейшей форме метод рекурсивного спуска предоставляет удобную возможность построения лексического анализатора (распознавателя символов) языка, порожденного $LL(1)$-грамматикой $G = (N, T, P, S)$. Такой лексический анализатор каждому символу из множества $N \cup T$ ставит в соответствие единственную процедуру, с помощью которой для любой строки языка можно однозначно определить, выводима она из этого символа или нет. Условимся обозначать pX такую процедуру, соответствующую символу $X \in N \cup T$. Так, если $a \in T$, то pa специфицирует ввод следующего входного символа и сравнение его с символом a. Если $A \in N$ и продукции грамматики, содержащие символ A в левых частях, имеют вид $A \to \alpha_1, A \to \alpha_2, \ldots, A \to \alpha_n$, то, сравнив очередной входной символ с элементами области значений функции LOOKAHEAD, можно определить, какой продукцией нужно воспользоваться для осуществления следующего шага вывода. Если с помощью процедуры pA и в зависимости от очередного входного символа выбрана продукция $A \to \alpha_i$, и $\alpha_i = X_{i1} X_{i2} \ldots X_{im}$, $X_{ij} \in N \cup T$, $1 \leqslant j \leqslant m$, то имеет смысл говорить соответственно о процедурах $pX_{i1}; pX_{i2}; \ldots; pX_{im}$.

Таблица 7.3

Продукции	LOOKAHEAD-множество
$S \to aAB$	$\{a\}$
$S \to bS$	$\{b\}$
$A \to aAb$	$\{a\}$
$A \to bBc$	$\{b\}$
$B \to AB$	$\{a,b\}$
$B \to c$	$\{c\}$

В качестве примера рассмотрим грамматику G_3, которая, как вы сами можете убедиться, является $LL(1)$-грамматикой, ее продукции имеют вид

$$S \to AB|bS,$$

$$A \to aA|bB,$$

$$B \to AB|c,$$

а значения функции LOOKAHEAD, приведены в табл. 7.3.

Представим себе, что функция *nextsymbol* определена таким образом, что позволяет узнать вид очередного входного символа, не выполняя его реального ввода из входного потока. Если оказывается, что такого очередного входного символа нет, то управление передается специальной процедуре обработки аварийной ситуации. Воспользуемся этими рассуждениями и напишем на языке программирования, подобном Паскалю, лексический анализатор языка $L(G_3)$:

```
begin pS; if ввод окончен then останов else авария end
procedure pS ≡ case nextsymbol of
                'a': pa; pA; pB;
                'b': pb; pS;
                'c':
                end;
procedure pA ≡ case nextsymbol of
                'a': pa; pA; pb;
```

```
                'b': pb; pB; pc;
                'c':
                end;
procedure pB ≡  case nextsymbol of
                'a','b': pA; pB;
                'c': pc
                end;
procedure pa ≡  begin
                ввести (символ); if символ ≠ 'a' then авария
                end;
procedure pb ≡  begin
                ввести (символ); if символ ≠ 'b' then авария
                end;
procedure pc ≡  begin
                ввести (символ); if символ ≠ 'c' then авария
                end.
```

Не вдаваясь в подробности, скажем лишь, что в результате выполнения процедуры *авария* осуществляется прекращение всех вычислений и вывод соответствующего сообщения.

Следующим шагом введем в действия, выполняемые анализатором, семантическую обработку, иначе говоря, потребуем от него ответа на вопрос, можно ли считать некоторое внутреннее представление синтаксически корректной последовательности введенных входных символов верным результатом. Если это внутреннее представление введенной последовательности входных символов можно рассматривать как дерево вывода, то мы столкнемся здесь с проблемой выводимости. На практике разработчики компиляторов чаще всего именно так и поступают, используя внутреннюю форму представления в виде семантического дерева, подобного тому, которое мы описали в гл. 2 как наиболее подходящую форму. Покажем это на примере грамматики, порождающей арифметические выражения, составленные из переменных a, b, c и символа $ — последнего из вводимых входных символов. Пусть ее продукции имеют вид

$$S \to E\$,$$
$$E \to T \mid E + T \mid E - T,$$
$$T \to F \mid T \times F \mid T/F,$$
$$F \to a \mid b \mid c \mid (E).$$

Ясно, что такая грамматика не может быть $LL(1)$-грамматикой и вообще $LL(k)$-грамматикой для любого k, так как для любого k нетерминальному символу E может соответствовать арифметическое выражение вида $(((\ldots\ldots(a)\ldots)))+a$ или вида $(((\ldots(a)\ldots)))-a$, содержащее $k+1$ пару скобок, и следовательно просмотр этого выражения на k элементов вперед не позволит однозначно определить, какой должна быть следующая продукция ($E \to E + T$ или $E \to E - T$). Однако не будем отчаиваться. У нас есть несколь-

ко возможностей. Прежде всего попытаемся найти грамматику, эквивалентную исходной, но являющуюся $LL(1)$-грамматикой. Понятно, что искомая грамматика не должна содержать продукций рекурсивных слева (см. упр. 7.2). Воспользуемся теоремой 5.6 и построим грамматику со следующими продукциями:

$S \to E\$,$

$E \to TA,$

$A \to \varepsilon | + TA | - TA,$

$T \to FB,$

$B \to \varepsilon | \times FB | / FB,$

$F \to a | b | c | (E).$

Построенная грамматика является $LL(1)$-грамматикой и, воспользовавшись методом рекурсивного спуска, легко можно построить искомый синтаксический анализатор. К сожалению, добавить сементический разбор оказалось возможным лишь благодаря ряду хитростей. Действительно, выбранная исходная грамматика была определена так аккуратно, что вид семантического дерева следовал из вида дерева вывода. Например, арифметическое выражение $a - b - c\$$ имеет соответственно дерево вывода, представленное на рис. 7.3, *а*, а семантическое дерево представлено на рис. 7.3, *б* (на рис. 7.3, *в* представлено некорректное семантическое дерево).

Опишем теперь исходную грамматику с помощью синтаксических диаграмм так, как это показано на рис. 7.4. Подобный подход избавляет нас от необходимости вводить нетерминальные символы A и B и позволяет построить с помощью метода рекурсивного спуска лексический анализатор, в который будет легко добавить функции семантического разбора. Действительно, каждому нетерминальному символу, использованному в синтаксических диаграммах, соответствует одна процедура (pE, pT или pF). Как это принято, вместо того, чтобы отдельно описывать процедуры ($p+$, $p-$ и т. д.) для

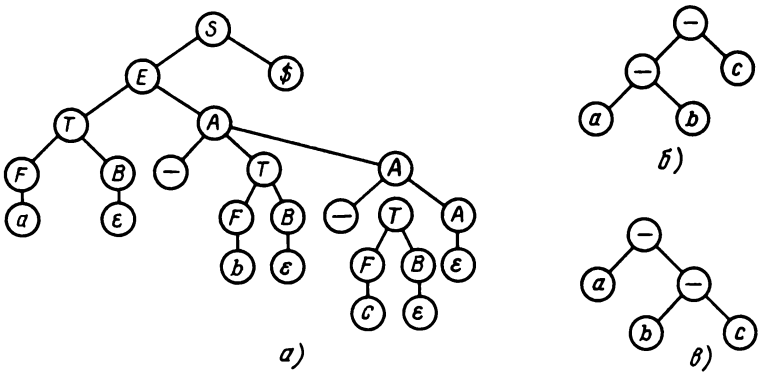


Рис. 7.3

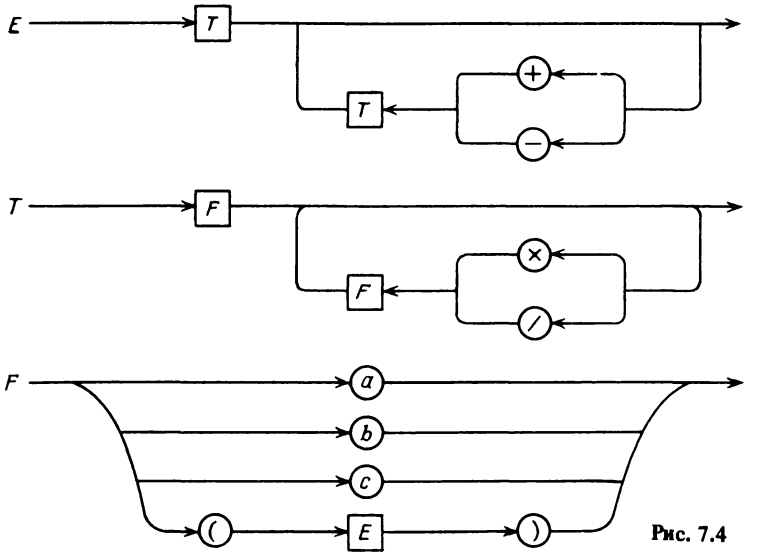


Рис. 7.4

распознавания терминальных символов, присовокупим описание этих процедур непосредственно к описаниям процедур *pE*, *pT* и *pF*.

Вновь воспользуемся языком, подобным языку программирования Паскаль, и запишем результат:

```
begin pE; if nextsymbol = '$' then останов else авария end
procedure pE ≡ begin
                 pT;
                 while nextsymbol = '+' or nextsymbol = '–' do
                     begin
                     ввести (символ); pT;
                     end
               end;
procedure pT ≡ begin
                 pT;
                 while nextsymbol = 'X' or nextsymbol = '/' do
                       begin
                       ввести (символ); pF
                       end
               end;
procedure pF ≡ begin
                 case nextsymbol of
                 'a', 'b', 'c': ввести (символ);
                 '(': ввести (символ); pE; ввести (символ);
                      if символ ≠ ')' then авария;
                 ')', '$': авария
                 end
```

Добавим теперь функции семантического разбора, поставив в соответствие каждой процедуре pX некоторую функцию fX, с помощью которой станет возможным построить семантическое дерево арифметического выражения, распознаваемого с помощью процедуры pX. Предположим, что функция fX имеет три аргумента, символ S и деревья T_1 и T_2, а значением ее является некое дерево, корень которого помечен символом S и которое распадается на два поддерева, T_1 и T_2. Обозначим пустое дерево nil. В результате можно написать следующую программу:

```
begin var t: tree; t:=fE; if nextsymbol = '$' then t else авария end
function fE: tree ≡ begin
                    var t:tree;
                    t:fT;
                    while nextsymbol='+' or nextsymbol='–' do
                          begin
                          ввести (символ) :t:=tree construct (символ, t, fT)
                          end;
                    fE:=t
                    end;
function fT:tree ≡  begin
                    var t : tree;
                    t :=fF;
                    while nextsymbol='X' or nextsymbol='/' do
                       begin
                       ввести (символ) ; t:=tree construct (символ't' fF)
                       end;
                    fT:=t
                    end;
function fF:tree ≡  begin
                    case nextsymbol of
                    'a', 'b', 'c': ввести (символ);
                                   fF:=tree construct (символ, nil, nil)
                    '(':ввести (символ); fF:=fE; ввести (символ);
                         if символ≠')' then авария;
                    ')', '$': авария
                    end
```

Таким образом, метод рекурсивного спуска является одним из наиболее эффективных способов создания компиляторов. Как мы показали, если рассматриваемая грамматика является LL (1)-грамматикой, то для построения лексического анализатора проще всего воспользоваться значениями функции LOOKAHEAD. Если затем к полученному лексическому анализатору добавить функции семантического разбора, то мы получим искомый анализатор. Однако на практике синтаксис большинства языков программирования не может быть достаточно полно специфицирован LL (1)-грамматикой. В таком случае разработчик компилятора с такого языка программирования может, конечно, воспользоваться приведенными в этой главе результатами

и рекомендациями, однако их может оказаться недостаточно. Более того, использование эквивалентной $LL(1)$-грамматики часто приводит к тому, что добавление функций семантического разбора становится невозможным. Заметим, что при необходимости анализировать k очередных входных символов перед разработчиком возникает задача обобщить решение на $LL(k)$-грамматику. При желании читатель может детально ознакомиться с использованием на практике метода рекурсивного спуска для построения компиляторов с языка программирования Алгол; для этого достаточно обратиться к книге Davie A. J. T., Morrison R. Recursive Descent Compiling. – Ellis Horwood, 1981.

Более полное теоретическое изложение проблемы $LL(k)$-грамматик, метода рекурсивного спуска и ряда интересных исследований по вопросам исправления ошибок читатель найдет в книге Backhouse R. C. Syntax of Programming Languages: Theory and Practice. – Prentice-Hall International, 1979.

УПРАЖНЕНИЯ

7.1. Назовите, какая из описанных ниже грамматик является $LL(1)$-грамматикой, и для каждой такой грамматики постройте с помощью метода рекурсивного спуска лексический анализатор.

а) $S \to A \mid B$,
$A \to aA \mid a$,
$B \to bB \mid b$;

б) $S \to AB$,
$A \to Ba \mid \varepsilon$,
$B \to Cb \mid C$,
$C \to c \mid \varepsilon$;

в) $S \to aAaB \mid bAbB$,
$A \to S \mid cb$,
$B \to cB \mid a$.

7.2. Покажите что $LL(1)$-грамматики не могут иметь леворекурсивных слева продукций. Не забудьте, что в соответствии с принятым соглашением рассматриваемые грамматики не имеют иррелевантных продукций. Справедливо ли это для $LL(k)$-грамматик?

7.3. Обобщите теорему 7.3 и предшествующие ей определения на случай $LL(2)$-грамматики в терминах k-LOOKAHEAD-функций.

7.4. Воспользуйтесь решением упражнения 7.3 и покажите, что следующая грамматика является строгой $LL(2)$-грамматикой, а затем постройте методом рекурсивного спуска соответствующий лексический анализатор:

$S \to aAS \mid AbSc \mid \varepsilon$,
$A \to cbA \mid a$.

7.5. а) Покажите, что следующая грамматика, имеющая продукции

$S \to aAaa \mid bAba$,
$A \to b \mid \varepsilon$,

является $LL(2)$-грамматикой, но не является строгой $LL(2)$-грамматикой.

б) Покажите, что грамматика, имеющая продукции

$S \to aBA \mid bBbA,$
$A \to abA \mid c,$
$B \to a \mid ab,$

является $LL(3)$-грамматикой, но не является строгой $LL(k)$-грамматикой для любого $k \geqslant 1$.

7.6. Покажите, что если L – регулярный язык, $\$ \notin L$, то $L\$ = \{x\$ \mid x \in L\}$ может быть порожден строгой $LL(1)$-грамматикой (воспользуйтесь детерминированным конечным автоматом, принимающим язык L).

7.7. Покажите, что если грамматика $G = (N, T, P, S)$ неоднозначна, то существуют w, $x \in T^*$, $A \in N$, $\gamma \in (N \cup T)^*$ такие, что $S \overset{*}{\Rightarrow} wA\gamma$ – левосторонняя схема вывода, и существуют две несовпадающие схемы вывода $A \Rightarrow \alpha \overset{*}{\Rightarrow} x$ и $A \Rightarrow \beta \overset{*}{\Rightarrow} x$, где $\alpha, \beta \in (N \cup T)^*$ и $\alpha \neq \beta$. Используя этот результат, покажите, что любая $LL(k)$-грамматика является однозначной грамматикой.

7.8. Покажите, что грамматика, имеющая продукции вида

$S \to C\$,$
$C \to bA \mid aB,$
$A \to a \mid aC \mid bAA,$
$B \to b \mid bC \mid aBB,$

не является $LL(k)$-грамматикой для любого k. Можно ли для некоторого k построить эквивалентную ей $LL(k)$-грамматику?

ГЛАВА 8

СИНТАКСИЧЕСКИЙ РАЗБОР "СНИЗУ ВВЕРХ"

"Всегда стройте все снизу вверх"

Франклин Рузвельт
7 апреля 1932 г.

Вам должно быть понятно, что каким бы методом ("сверху вниз" или "снизу вверх") мы ни воспользовались для синтаксического разбора строки $x \in L(G)$, где G – контекстно свободная грамматика, результат должен быть один (это утверждение справедливо при условии допустимости использования обоих методов), так как эти методы отличаются лишь способами построения деревьев[1]. Если метод синтаксического разбора "сверху вниз", описанный в гл. 7, рекомендует построение указанных деревьев начиная с корня, то метод "снизу вверх" рекомендует строить их наоборот, начиная с листьев и кончая корнем дерева. Введенные входные строки в последнем случае анализируются слева направо; получаемые в процессе подстроки сравниваются с правыми частями продукций грамматики, а при совпадении заменяются или *приводятся* к нетерминальному символу, стоя-

[1] Автор имеет в виду дерево вывода и семантическое дерево. – *Прим. ред.*

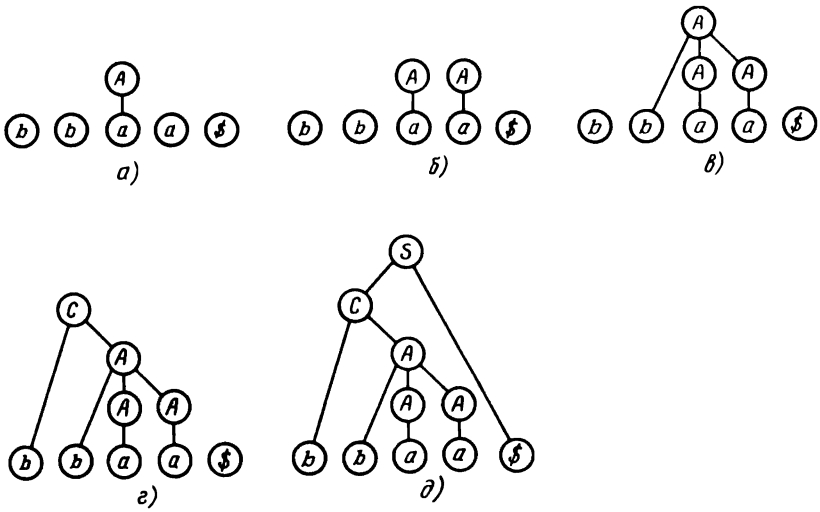


Рис. 8.1

щему в левой части таких продукций. В результате такой замены может быть получена сентенциальная форма грамматики, а затем вся процедура повторяется до тех пор, пока полученная сентенциальная форма не примет вид S, где символом S помечен корень дерева вывода. В результате будет получена последовательность схем вывода и соответствующих им приведений, позволяющая построить дерево вывода от листьев до корня. Каждому элементу этой последовательности, очевидно, соответствует один родительский узел дерева вывода.

Рассмотрим грамматику G_1, имеющую следующие продукции:

$S \to C\$,$

$C \to aBC|aB|bAC|bA,$

$A \to bAA|a,$

$B \to aBB|b.$

Из результатов упр. 2.2, читателю должно быть известно, что G_1 – однозначная грамматика, и, следовательно, любой строке $x \in L(G_1)$ соответствует единственное дерево вывода. Например, дерево вывода для строки $x = bbaa\$$ приведено на рис. 8.1, *д*. Далее, каждому дереву вывода соответствует единственная правосторонняя схема вывода, т. е. такая схема вывода, в которой всегда выводится самый правый нетерминальный символ. Так, дереву вывода, рассмотренному в вышеприведенном примере, соответствует правосторонняя схема вывода

$$S \Rightarrow C\$ \Rightarrow bA\$ \Rightarrow bbAA\$ \Rightarrow bbAa\$ \Rightarrow bbaa\$.$$

Рассматриваемый метод синтаксического разбора "снизу вверх", позволит построить такую схему вывода справа налево. Пример построения дерева вывода для строки $bbaa\$$ показан на рис. 8.1.

Подстроку, которая приводится, называют *основой правосторонней сентенциальной формы,* или *правосторонней простой фразой.* Процесс построения дерева вывода для нашего примера иллюстрируется табл. 8.1.

Таблица 8.1

Правосторонняя сентенциальная форма	Основа	Замещающий символ
$bbaa\$$	a (левый символ)	A
$bbAa\$$	a	A
$bbAA\$$	bAA	A
$bA\$$	bA	C
$C\$$	$C\$$	S

При использовании метода синтаксического разбора "снизу вверх" возникает проблема поиска и своевременного приведения основы сентенциальной формы на каждом шаге построения дерева вывода.

ГРАММАТИКА ПРОСТОГО ПРЕДШЕСТВОВАНИЯ

Решим названную проблему прежде всего в небольшом классе грамматик, называемых грамматиками с простым предшествованием.

Перечислим слева направо элементы правосторонней сентенциальной формы, рассматривая пары смежных символов, в поисках самого правого символа основы сентенциальной формы, а когда найдем его, то начиная с него продолжим перечисление символов уже справа налево и найдем самый левый символ этой сентенциальной формы.

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

а) X – часть основы сентенциальной формы, а Y – нет. Таким образом, X является окончанием основы сентенциальной формы. В этом случае говорят, что X предшествует Y и записывают это следующим образом: $X \cdot\!> Y$

б) И X, и Y входят в основу сентенциальной формы. В этом случае говорят, что X и Y имеют равное предшествование и записывают это следующим образом: $X \doteq Y$.

в) Y – часть основы сентенциальной формы, а X – нет. Таким образом, Y – заголовок основы сентенциальной формы. В этом случае говорят, что Y предшествует X, и записывают это следующим образом: $X <\cdot Y$

Отношения, обозначенные символами $\cdot\!>$, $\doteq$, $<\cdot$, называют *отношениями простого предшествования.* Из вышеприведенных определений можно сделать вывод, что

а) $X \cdot\!> Y \Leftrightarrow Y$ – терминальный символ (напомним, что $\alpha XY\beta$ – сентенциальная форма правостороннего вывода) и существует некоторая продук-

ция $A \to \gamma_1 B Z \gamma_2$, $A, B \in N$, $Z \in N \cup T$, $\gamma_1, \gamma_2 \in (N \cup T)^*$ такая, что $B \stackrel{+}{\Rightarrow} \delta_1 X$ и $Z \stackrel{*}{\Rightarrow} Y \delta_2$, $\delta_1, \delta_2 \in (N \cup T)^*$;

б) $X \doteq Y \Leftrightarrow$ существует некоторая продукция $A \to \gamma_1 X Y \gamma_2$, $A \in N$, $\gamma_1, \gamma_2 \in (N \cup T)^*$;

в) $X \lessdot Y \Leftrightarrow$ существует некоторая продукция $A \to \gamma_1 X B \gamma_2$, $B \in N$, $\gamma_1, \gamma_2 \in (N \cup T)^*$ такая, что $B \stackrel{+}{\Rightarrow} Y\delta$, $\delta \in (N \cup T)^*$.

Контекстно свободная грамматика $G = (N, T, P, S)$ называется *грамматикой простого предшествования*, если

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

2) любые два символа, составляющие элемент множества $(N \cup T) \times (N \cup T)$, связаны одним и тем же отношением предшествования.

Грамматика G_1 удовлетворяет первому из этих условий, но не удовлетворяет второму. Действительно, одновременно выполняются $a \cdot > a$ (так как $A \to bAA$, $A \stackrel{+}{\Rightarrow} a$ и $A \stackrel{*}{\Rightarrow} a$) и $a \lessdot a$ (так как $C \to aB$ и $B \stackrel{+}{\Rightarrow} aBB$).

Рассмотрим теперь грамматику простого предшествования G_2, имеющую продукции вида

$S \to (R \mid a$,

$R \to Sa)$.

Предоставляем читателю самостоятельно проверить, что приведенные в табл. 8.2 отношения являются отношениями простого предшествования.

Если $G = (N, T, P, S)$ – грамматика простого предшествования, то найти основу любой ее правосторонней сентенциальной формы очень просто. Для этого достаточно перечислить элементы указанной сентенциальной формы и найти самую левую пару символов X_j и X_{j+1} таких, что $X_j \cdot > X_{j+1} \cdot X_j$ – окончание основы сентенциальной формы. Перечислим теперь элементы сентенциальной формы грамматики G справа налево начиная с символа X_j и до тех пор, пока не будет найдена пара символов X_{i-1} и X_i таких, что $X_{i-1} \lessdot X_i \cdot X_i$. Таким образом, будет найдена основа сентенциальной формы, имеющая вид $\beta = X_i \ldots X_j$, а значит, должна существовать продукция вида $A \to \beta$, т. е. основа сентенциальной формы может быть приведена к символу A. Единственная трудность может возникнуть в случае, если окончанием основы сентенциальной формы является последний символ этой формы или же заголовок сентенциальной формы является ее первым символом. Чтобы преодолеть эту трудность, введем новый символ, "маркер конца", $\$ \in N \cup T$, который будем использовать для индикации обоих концов сентенциальной формы. Далее, без ограничения общности, предположим, что $\$ \cdot > X$, $X \cdot > \$$ для всех $X \in N \cup T$ и результатом последовательности приведений входной строки $\$ x \$$ $(x \in L(G))$ будет строка вида $\$ S \$$.

Далее, язык $L(G_2)$ содержит строки вида a, (aa), $((aa)a)$, $(((aa)a)a)$, ... Синтаксический разбор строки $\$(((aa)a)a)\$$, демонстрирующий использование отношений предшествования, приводится в табл. 8.3. Процедура разбора соответствует правосторонней схеме вывода $S \Rightarrow (R \Rightarrow (Sa) \Rightarrow ((Ra) \Rightarrow ((Sa)a) \Rightarrow (((Ra)a) \Rightarrow (((Sa)a)a) \Rightarrow (((aa)a)a)$.

Таблица 8.2

	S	*R*	*a*	(	)
S	.	.	≐	.	.
R	.	.	>	.	.
a	.	.	>	.	≐
(	<	≐	<	<	.
)	.	.	>	.	.

Таблица 8.3

Сентенциальная форма	*Основа*
$ (((a a)a)a) $ < < < < >	a
$ (((S a) a)a)$ < < < < ≐ ≐ >	Sa)
$ (((R a)a)$ < < < ≐ >	(R
$ ((S a) a) $ < < < ≐ ≐ >	Sa)
$ ((R a)$ < < ≐ >	(R
$ (S a) $ < < ≐ ≐ >	Sa)
$ (R $ < ≐ >	(R
$ S $	

При реализации приведения основы сентенциальной формы удобными средствами является стек и входной буфер, для управления которыми используются следующие элементарные операции: *занесение в стек* (т.е. помещение в вершину стека очередного входного символа). *Приведение* (т.е. синтаксический анализатор), располагая информацией о том, что основа сентенциальной формы находится в стеке, осуществляет замену ее на соответствующий ей нетерминальный символ. Синтаксический анализатор успешно осуществляет разбор входной строки, если в результате его работы стек будет содержать строку S\$ и входной буфер будет содержать символ \$. Построенные на таких принципах синтаксические анализаторы часто называют анализаторами "сдвига–приведения"[1]. В табл. 8.4 представлено содержимое стека и входного буфера на каждом шаге разбора рассмотренного выше примера.

Понятно, что при функционировании реального анализатора "сдвига–приведения" очередная операция сдвига будет выполнена при условии, что первый символ входного буфера предшествует последнему занесенному в стек символу либо оба они имеют равное предшествование. В противном случае анализатор будет извлекать символы из стека и искать основу и тем самым выполнять операцию приведения. Если в последнем случае основа не будет найдена, то возникнет аварийная ситуация.

Одной из основных проблем, возникающих при такой реализации синтаксического анализатора, является вычисление отношений предшествования. Существует простой алгоритм, решающий эту проблему, однако здесь надо быть осторожным во избежание трудностей с хранением данных. Если грамматика имеет m терминальных и n нетерминальных символов, то, очевидно, существует $(m+n)^2$ возможных отношений предшествования. Можно хранить непосредственно множество этих отношений в виде так называемой матрицы предшествования, тем более, что многие элементы матрицы будут наверняка иметь вид "·", что означает пустоту отношения. При

[1] Операцию занесения в стек иногда называют *сдвигом* стека. – *Прим. ред.*

Таблица 8.4

Содержимое стека	Входной буфер	Выполняемое действие
$\$$	$(((aa)a)a)\$$	Сдвиг стека
$\$($	$((aa)a)a)\$$	Сдвиг стека
$\$(($	$(aa)a)a)\$$	Сдвиг стека
$\$(((a$	$aa)a)a)\$$	Сдвиг стека
$\$(((a$	$a)a)a)\$$	Приведение с помощью продукции $S \to a$
$\$(((S$	$a)a)a)\$$	Сдвиг стека
$\$(((Sa$	$)a)a)\$$	Сдвиг стека
$\$(((Sa)$	$a)a)\$$	Приведение с помощью продукции $R \to Sa)$
$\$(((R$	$a)a)\$$	Приведение с помощью продукции $S \to (R$
$\$((S$	$a)a)\$$	Сдвиг стека
$\$((Sa$	$)a)\$$	Сдвиг стека
$\$((Sa)$	$a)\$$	Приведение с помощью продукции $R \to Sa)$
$\$((R$	$a)\$$	Приведение с помощью продукции $S \to (R$
$\$(S$	$a)\$$	Сдвиг стека
$\$(Sa$	$)\$$	Сдвиг стека
$\$(Sa)$	$\$$	Приведение с помощью продукции $R \to Sa)$
$\$(R$	$\$$	Приведение с помощью продукции $S \to (R$
$\$S$	$\$$	Строка принята

необходимости можно воспользоваться известными методами компактного хранения рассеянных массивов или же решить эту проблему с помощью так называемых *функций предшествования*. Пусть P – матрица предшествования; построим две целые функции f и g такие, что

$f(i) < g(j)$, если $P_{ij} \equiv \lessdot$;
$f(i) = g(j)$, если $P_{ij} \equiv \doteq$;
$f(i) > g(j)$, если $P_{ij} \equiv \gtrdot$.

Если возможно найти такие функции, то вместо $(m+n)^2$ элементов достаточно хранить лишь $2(m+n)$ элементов. Однако при таком решении теряется информация о несуществующих отношениях, которая может быть использована, например, для обнаружения и исправления ошибок ввода входных символов.

В заключение отметим, что основным недостатком описанного в этом параграфе метода разбора "снизу вверх" является применимость его лишь в узком классе грамматик простого предшествования.

$LR(0)$-ГРАММАТИКИ

Метод синтаксического разбора "снизу вверх", наиболее широко используемый разработчиками компиляторов и называемый также LR-разбором, был найден и разработан Кнутом (Knuth D. E. On the Translation of Languages from Left to Right//Information and Control, 1965, № 8, Pl 607 – 639). Аббревиатура LR означает "левосторонний ввод – правосторонний вывод". Как и в случае грамматики простого предшествования, LR-разбор может

быть реализован в виде анализатора "сдвига–приведения", однако применим в более широком классе грамматик. В этом отношении LR-разбор превосходит LL-разбор. Основным недостатком LR-разбора является то, что он не поддается интуитивному восприятию, и поэтому труден для понимания. К счастью, процедура LR-разбора может быть легко формализована, что является веской причиной использования его на практике.

На каждом шаге функционирования анализатора "сдвига–приведения" (см. табл. 8.4) содержимое стека анализатора $\alpha \in (N \cup T)^*$ и содержимое его входного буфера $z \in T^*$ таковы, что строка αz представляет собой правостороннюю сентенциальную форму. На каждом шаге такого разбора может быть осуществлена либо операция сдвига, т. е. занесения в стек очередного символа из входного буфера, либо операция приведения. Остается решить, какая конкретно операция должна быть осуществлена на каждом конкретном шаге, и если осуществляется операция приведения, необходимо еще выбрать продукцию грамматики, с помощью которой можно будет осуществить приведение.

Однако, если рассматриваемая грамматика удовлетворяет ряду условий (которые мы сформулируем позже), то на каждом шаге функционирования анализатора можно по содержимому его стека однозначно определить, какое перемещение анализатор должен совершить для построения обратной правосторонней схемы вывода.

Допустим, удовлетворяющая указанным условиям грамматика имеет продукцию $A \to \beta$ и в данный момент содержимое стека $\alpha = \gamma\beta$; это означает, что в данный момент следует осуществить операцию приведения. Итак, мы поставили в соответствие содержимому стека анализатора его перемещение. Требование однозначности этого соответствия означает, что выбранное перемещение анализатора должно быть единственным возможным перемещением в данный момент. Другими словами, не может существовать никакого другого перемещения анализатора (ε-перемещения, или перемещения, соответствующего операции сдвига), в результате выполнения которого новое содержимое стека, αx, $x \in T^*$, оказалось бы таким, что после осуществления некоторой операции приведения (допустим с помощью продукции $B \to \beta'$), можно было бы получить правостороннюю сентенциальную форму грамматики. Сформулируем теперь это требование формально: пусть существуют две правосторонние схемы вывода

$$S \overset{*}{\Rightarrow} \gamma A x y \Rightarrow \gamma\beta x y \text{ и}$$

$$S \overset{*}{\Rightarrow} \delta B y \Rightarrow \delta\beta' y = \gamma\beta x y,$$

где $\gamma, \delta, \beta, \beta' \in (N \cup T)^*$, $A, B \in N$, $x, y \in T^*$, тогда они совпадают, т. е. $\delta = \gamma$, $A = B$, $\beta = \beta'$ и $x = \varepsilon$.

Множество строк, занесенных в стек к моменту осуществления операции приведения с помощью продукции $A \to \beta$, называют LRCONTEXT-множеством продукции $A \to \beta$ и обозначают LRCONTEXT $(A \to \beta)$. Формально LRCONTEXT $(A \to \beta) = \{\alpha \mid \alpha = \gamma\beta \in (N \cup T)^*$, где $S \overset{*}{\Rightarrow} \gamma A x \Rightarrow \gamma\beta x$ – правосторонняя схема вывода, $x \in T^*$, $\gamma \in (N \cup T)^*\}$. Описанное выше условие не выполняется, если существуют несовпадающие продукции $A \to \beta$ и $B \to \beta'$, та-

кие, что $\gamma\beta \in$ LRCONTEXT $(A \to \beta)$ и $\delta\beta' \in$ LRCONTEXT $(B \to \beta')$, где $\delta\beta' = \gamma\beta x$ для некоторого $x \in T^*$.

С другой стороны, даже если рассматриваемая грамматика удовлетворяет этим условиям, существует еще проблема определения момента завершения приема. Рассмотрим, например, продукцию вида

$$S \to Sa|a,$$

тогда LRCONTEXT $(S \to a) = \{a\}$ и LRCONTEXT $(S \to Sa) = \{Sa\}$ и, таким образом, описанные выше условия удовлетворяются. Однако если S – единственный символ, содержащийся в стеке, то анализатор, в принципе, может закончить прием строки. В такой ситуации без дополнительной информации нельзя определить, нужно ли осуществить ввод символа и завершить разбор, или же осуществить операцию сдвига стека. Эту проблему легко решить перейдя к пополненной грамматике, имеющей продукции

$$S' \to S\$,$$
$$S \to Sa|a,$$

где \$ – маркер конца ввода. Тогда

$$\text{LRCONTEXT}(S' \to S\$) = \{S\$\},$$
$$\text{LRCONTEXT}(S \to Sa) = \{Sa\},$$
$$\text{LRCONTEXT}(S \to a) = \{a\}.$$

Подводя итоги, сформулируем следующее определение: контекстно свободная грамматика $G = (N, T, P, S)$ называется $LR(0)$-грамматикой, если

а) ни одна из ее продукций не содержит в своей правой части исходного символа;

б) если существует две правосторонние схемы вывода

$$S \overset{*}{\Rightarrow} \gamma Axy \Rightarrow \gamma\beta xy \quad \text{и}$$
$$S \overset{*}{\Rightarrow} \delta By \Rightarrow \delta\beta y = \gamma\beta xy,$$

где $\gamma, \delta, \beta, \beta' \in (N \cup T)^*$, $A, B \in N$ и $x, y \in T^*$, то $\delta = \gamma$, $A = B$, $\beta = \beta'$ и $x = \varepsilon$.

ТЕОРЕМА 8.1

Контекстно свободная грамматика $G = (N, T, P, S)$ является $LR(0)$-грамматикой $\Leftrightarrow$

а) ни одна из ее продукций не содержит исходного символа в своей правой части;

б) если $\alpha \in$ LRCONTEXT $(A \to \beta')$ и $\alpha x \in$ LRCONTEXT $(B \to \beta')$, где $A \to \beta, B \to \beta' \in P$, $\alpha \in (N \cup T)^*$ и $x \in T^*$, то $x = \varepsilon$, $A = B$ и $\beta = \beta'$

Чтобы проверить, является ли грамматика, удовлетворяющая первому условию этой теоремы, $LR(0)$-грамматикой, необходимо найти все LRCONTEXT-множества. Каждая строка, являющаяся элементом множества LRCONTEXT $(A \to \beta)$, имеет, очевидно, вид $\gamma\beta$, где $\gamma \in (N \cup T)^*$. Определим теперь функцию LEFT $(A) = \{\gamma | S \overset{*}{\Rightarrow} \gamma Ax$ – правосторонняя схема вывода, $x \in T^*\}$.

ТЕОРЕМА 8.2

$\mathrm{LRCONTEXT}(A \to \beta) = \mathrm{LEFT}(A) \cdot \{\beta\}$.

Таким образом, чтобы найти LRCONTEXT-множества для грамматики, достаточно найти все LEFT-множества для всех нетерминальных символов грамматики. Если предположить, что ни одна продукция грамматики $G = (N, T, P, S)$ не содержит в своей правой части исходного символа S, то утверждение $\mathrm{LEFT}(S) = \{\varepsilon\}$ будет справедливо. Кроме того, если $B \to \gamma_1 A \gamma_2$ – продукция грамматики G, то множество $\mathrm{LEFT}(B) \cdot \{\gamma_1\} \subset \mathrm{LEFT}(A)$. Например, пусть грамматика G_3 имеет продукции вида

$S \to A\$$,

$A \to AB \mid B$,

$B \to [A] \mid []$.

Строки, порожденные этой грамматикой, представляют собой сбалансированные в обычном понимании последовательности квадратных скобок и оканчиваются маркером конца строки, обозначенным символом

$[]\$$, $[[]]\$$, $[][]\$$, ...

Воспользовавшись последним замечанием по поводу функции LEFT, выпишем следующие уравнения:

$\mathrm{LEFT}(S) = \{\varepsilon\}$,

$\mathrm{LEFT}(A) = \mathrm{LEFT}(S) \cup \mathrm{LEFT}(A) \cup \mathrm{LEFT}(B) \cdot \{[\}$,

$\mathrm{LEFT}(B) = \mathrm{LEFT}(A)\{A\} \cup \mathrm{LEFT}(A)$.

Эти уравнения можно решить с помощью метода, обратного методу, описанному в гл. 5, т. е. поставив в соответствие каждому уравнению некоторую продукцию некоторой грамматики. Пусть при этом множеству $\mathrm{LEFT}(S)$ соответствует нетерминальный символ $\hat{S}$, множеству $\mathrm{LEFT}(A)$ – нетерминальный символ $\hat{A}$, а множеству $\mathrm{LEFT}(B)$ – нетерминальный символ $\hat{B}$. Таким образом, системе уравнений будет поставлена в соответствие грамматика $\hat{G}$, имеющая нетерминальные символы $\{\hat{S}, \hat{A}, \hat{B}\}$ и продукции вида

$\hat{S} \to \varepsilon$,

$\hat{A} \to \hat{S} \mid \hat{B}[$,

$\hat{B} \to \hat{A}A \mid \hat{A}$.

В общем случае LEFT-множества некоторой грамматики $G = (N, T, P, S)$ могут быть построены исходя из продукций грамматики $\hat{G} = (\hat{N}, \hat{T}, \hat{P}, \hat{S})$ при условии, что $\hat{N} = \{\hat{A} \mid A \in N\}$, $\hat{T} \subset N \cup T$. Продукции грамматики $\hat{G}$ имеют такой вид, что грамматика $\hat{G}$ является линейной слева грамматикой, а следовательно, все LEFT-множества – регулярны (см. упр. 3.1). Таким образом, в качестве следствия теоремы 8.2, можно сформулировать следующую теорему.

ТЕОРЕМА 8.3

Пусть $G = (N, T, P, S)$ – произвольная контекстно свободная грамматика, тогда ее LRCONTEXT-множества регулярны.

Это действительно важный результат! Регулярные множества представляют собой довольно широкий класс множеств, их свойства хорошо изучены, поэтому мы рассчитываем широко использовать их при проектировании синтаксических анализаторов.

Структура конечного автомата $\hat{M}$, соответствующего линейной слева грамматике $\hat{G}$, может быть описана следующим образом: каждой продукции вида $\hat{A} \to \hat{B}x$, $\hat{A}, \hat{B} \in \hat{N}$, $x \in \hat{T}$ соответствует путь, помеченный символом x, из узла, помеченного символом $\hat{A}$, в узел, помеченный символом $\hat{B}$, а каждой продукции вида $\hat{A} \to x$, $\hat{A} \in \hat{N}$, $x \in \hat{T}^*$, соответствует путь, помеченный символом x, из узла, соответствующего исходному состоянию автомата $\hat{M}$ (пусть он помечен цифрой 1) в узел, помеченный символом $\hat{A}$. Каждый из этих путей может, вообще говоря, включать одну или несколько дуг, каждая из которых помечена символом, являющимся элементом множества $\hat{T} \cup \{\varepsilon\}$. Если путь содержит в себе более одной дуги, пометим промежуточные узлы цифрами 2, 3, 4 и т. д. Обозначим $\hat{t}$ переходную функцию автомата $\hat{M}$; тогда для всех $A \in N$, $\mathrm{LEFT}(A) = \{x \in \hat{T}^* | \hat{t}(1, x) = \hat{A}\}$. На рис. 8.2, *а*

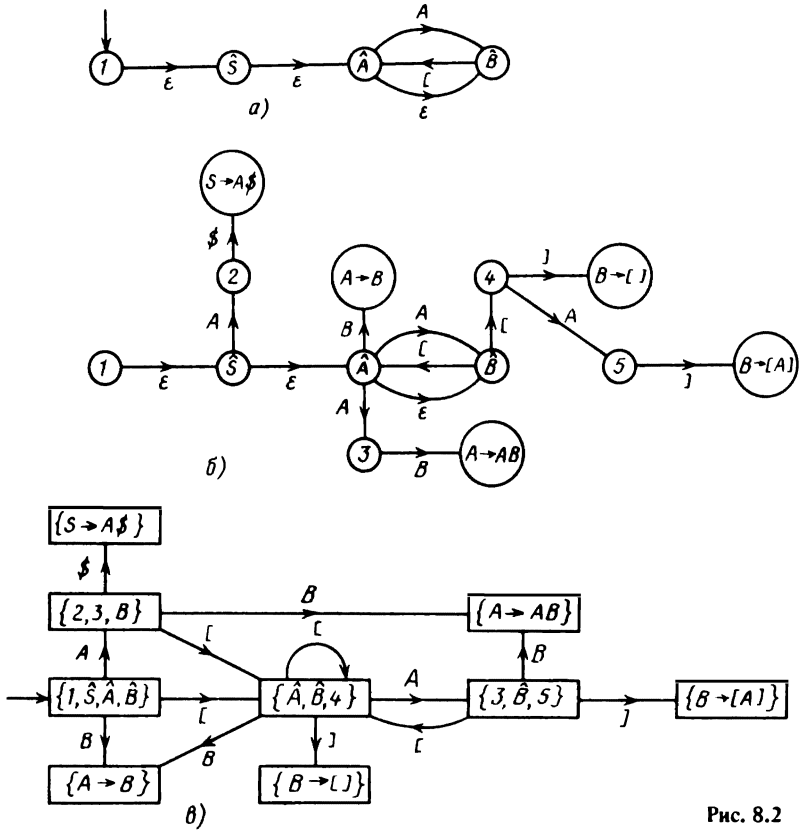


Рис. 8.2

приведен автомат, построенный для грамматики предыдущего примера. Для этого автомата справедливы следующие утверждения:

$$\mathrm{LEFT}(S) = 1;$$
$$\mathrm{LEFT}(A) = ((A + 1)[)^*;$$
$$\mathrm{LEFT}(B) = ((A + 1)[)^*(A + 1),$$

и, следовательно,

$$\mathrm{LRCONTEXT}(S \to A\$) = A\$;$$
$$\mathrm{LRCONTEXT}(A \to AB) = ((A + 1)[)^* AB;$$
$$\mathrm{LRCONTEXT}(A \to B) = ((A + 1)[)^* B;$$
$$\mathrm{LRCONTEXT}(B \to [A]) = ((A + 1)[)^*(A + 1)[A];$$
$$\mathrm{LRCONTEXT}(B \to [\,]) = ((A + 1)[)^*(A + 1)[\,].$$

Из теоремы 8.1 следует, что грамматика G_3 является $LR(0)$-грамматикой.

Воспользуемся теперь теоремой 8.2 и построим еще один конечный автомат, такой, что каждой продукции исходной грамматики соответствует одно состояние автомата, а множество входных строк, переводящее автомат из исходного состояния в состояние, соответствующее некоторой продукции, совпадает с LRCONTEXT-множеством для этой продукции. На рис. 8.2, *б* приведен такой автомат для рассмотренного выше примера. Эквивалентный ему детерминированный конечный автомат, не имеющий ε-перемещений, приведен на рис. 8.2, *в*. Такой детерминированный конечный автомат называется *характеристическим автоматом* грамматики. Состояния автомата, которым соответствуют продукции грамматики, называются *приводящими состояниями.* Грамматика является $LR(0)$-грамматикой $\Leftrightarrow$ каждому приводящему состоянию ее характеристического автомата соответствует единственная продукция, и ни один из узлов автомата не помечен терминальным символом.

Снова вернемся к примеру, и с помощью характеристического автомата выясним, принадлежит языку $L(G)$ строка [[] []] [] $ или нет. Пусть автомат осуществляет последовательный ввод входной строки до тех пор, пока не перейдет в приводящее состояние. Зная соответствующую этому состоянию продукцию, осуществим сокращение входной строки, в результате чего будет получена новая правосторонняя сентенциальная форма. Пусть теперь автомат продолжит ввод префикса этой сентенциальной формы (последовательность действий автомата приведена в табл. 8.5).

Однако действия построенного в этом примере автомата были не самыми эффективными (после каждой операции приведения строки автомат осуществлял повторный ввод строки, начиная с ее первого символа). Например, на "шаг 4" мы попадаем после только что осуществленной операции приведения [] в B. Если на предыдущем шаге мы запомнили, что между вводом и обработкой последовательности символов $[A$ и вводом последовательности символов [] автомат находился в состоянии $\{3, \hat{B}, 5\}$ (см. рис. 8.2, *в*), то можно заключить, что в результате ввода строки $[AB$ автомат

Таблица 8.5

Шаг	Введенная часть входной строки – префикс сентенциальной формы	Невведенная часть входной строки – остальная часть сентенциальной формы	Продукция, использованная для приведения	Новая сентенциальная форма
1	[[]	[]][]$	$B \to []$	[B[]][]$
2	[B	[]][]$	$A \to B$	[A[]][]$
3	[A[]	][]$	$B \to []$	[AB][]$
4	[AB	][]$	$A \to AB$	[A][]$
5	[A]	[]$	$B \to [A]$	B[]$
6	B	[]$	$A \to B$	A[]$
7	A[]	$	$B \to []$	AB$
8	AB	$	$A \to AB$	A$
9	A$	ε	$S \to AS$	S

перейдет в состояние $t(\{3, \hat{B}, 5\}, B)$. Прежде чем продолжить наши рассуждения, переименуем состояния характеристического автомата, заменив помечающую их последовательность символов на номера 1, 2, . . . , сохранив утраченную таким образом взаимосвязь между продукциями грамматики и состояниями автомата в специально составленной *таблице приведения* (рис. 8.3). Если, например, k – приводящее состояние автомата и ему соответствует, например, продукция $A \to \beta$, то k-я строка таблицы содержит запись вида $A \to \beta$.

Построим теперь эффективный алгоритм синтаксического разбора с использованием стека состояний. На каждом шаге алгоритма этот стек будет содержать те состояния характеристического автомата, через которые он прошел бы, если бы был введен текущий префикс правосторонней сентенциальной формы. Например, если текущий префикс имеет вид $[A\,[]$, стек должен содержать последовательность 1, 2, 3, 2, 9 (элементы стека перечислены снизу вверх).

На каждом этапе автомат может осуществить одно из следующих четырех перемещений:

а) *перемещение приведения:* если находящийся в вершине стека символ k помечает приводящее состояние автомата, которому соответствует, напри-

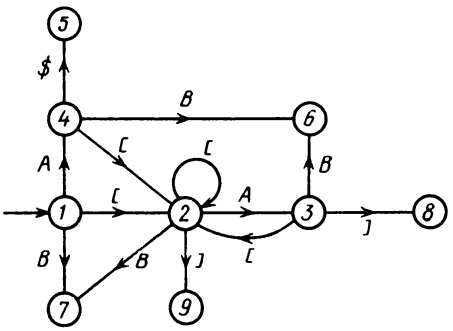


Таблица приведения

5	$S \to A\$$
6	$A \to AB$
7	$A \to B$
8	$B \to [A]$
9	$B \to B[]$

Рис. 8.3

мер, продукция $A \to X_1 X_2 \ldots X_m$ и $A \neq S$, тогда, если стек содержит последовательность символов $k_0, k_1, \ldots, k_n = k (n \geqslant m)$, осуществим операцию приведения с помощью продукции $A \to X_1 X_2 \ldots X_m$, в процессе чего из стека будет извлечено m символов, а по завершении в стек будет занесен символ, помечающий состояние $t (k_{n-m}, A)$;

б) *перемещение сдвига:* если находящийся в вершине стека символ k помечает неприводящее состояние автомата, тогда необходимо осуществить операцию сдвига, введя из входного потока очередной входной символ, например символ $a \in T$, а символ, помечающий состояние $t (k, a)$, занести в стек;

в) *перемещение приема:* если находящийся в вершине стека символ помечает приводящее состояние, которому соответствует единственная продукция, имеющая в своей левой части исходный символ S, например $S \to \alpha$, тогда при условии, что в стеке находится ровно $|\alpha| + 1$ символов, прием строки завершается и алгоритм заканчивает функционирование.

г) *аварийное перемещение*: если ни одно из вышеперечисленных перемещений осуществить невозможно, то ситуация считается аварийной, и либо входная строка признается ошибочной и функционирование алгоритма прекращается, либо предпринимаются попытки исправить ошибки.

Вернемся к примеру и проиллюстрируем сказанное, занеся предварительно в стек последовательность [[] []] [] $. Алгоритм разбора представлен в табл. 8.6.

Таблица 8.6

Содержимое стека	Выполняемое действие	Содержимое стека	Выполняемое действие
1	Сдвиг стека с занесением символа [	1, 2, 3	Сдвиг стека с занесением символа]
1,2	Сдвиг стека с занесением символа [	1, 2, 3, 8	Приведение с помощью продукции $B \to [A]$
1, 2, 2	Сдвиг стека с занесением символа]	1, 7	Приведение с помощью продукции $A \to B$
1, 2, 2, 9	Приведение с помощью продукции $B \to$ []	1, 4	Сдвиг стека с занесением символа [
1, 2, 7	Приведение с помощью продукции $A \to B$	1, 4, 2	Сдвиг стека с занесением символа]
1, 2, 3	Сдвиг стека с занесением символа [	1, 4, 2, 9	Приведение с помощью продукции $B \to$ []
1, 2, 3, 2	Сдвиг стека с занесением символа]	1, 4, 6	Приведение с помощью продукции $A \to AB$
1, 2, 3, 2, 9	Приведение с помощью продукции $B \to$ []	1, 4	Сдвиг стека с занесением символа $
1, 2, 3, 6	Приведение с помощью продукции $A \to AB$	1, 4, 5	Строка принята

Если рассматриваемая грамматика является $LR(0)$-грамматикой, то построение характеристического автомата грамматики легко поддается формализации. Если формализация осуществлена, то задача разработки син-

таксического анализатора значительно упрощается. К сожалению, реальная грамматика редко удовлетворяет условиям, достаточным, чтобы быть $LR(0)$-грамматикой. Тем не менее принципы LR-разбора успешно применяются в самых сложных случаях, что делает этот метод поистине мощным инструментом для программирования.

$LR(1)$-ГРАММАТИКИ

Рассмотрим грамматику G_4, которая имеет продукции вида

$S \to E\$,$

$E \to E + T \mid T,$

$T \to T \times F \mid F,$

$F \to a \mid (E).$

Характеристический автомат грамматики G_4 показан на рис. 8.4. Грамматика G_4 не является $LR(0)$-грамматикой, так как, например, $T \in \text{LRCONTEXT}(E \to T)$, но $T \times a \in \text{LRCONTEXT}(F \to a)$ и $T \times F \in \in \text{LRCONTEXT}(T \to T \times F)$. Таким образом, если в данный момент в стеке находится последовательность символов $\{1,6\}$, то однозначно указать, какое перемещение – приведения или сдвига – необходимо осуществить, невозможно. Решить эту проблему можно, "заглянув на один символ вперед". Если очередной входной символ имеет вид x, то автомат должен выполнить перемещение сдвига, в противном случае автомат должен осуществить перемещение приведения. Грамматики, подобные грамматике G_4, принято называть $LR(1)$-грамматиками, для таких грамматик описанный синтаксический анализатор должен быть модифицирован (пополнен функцией анали-

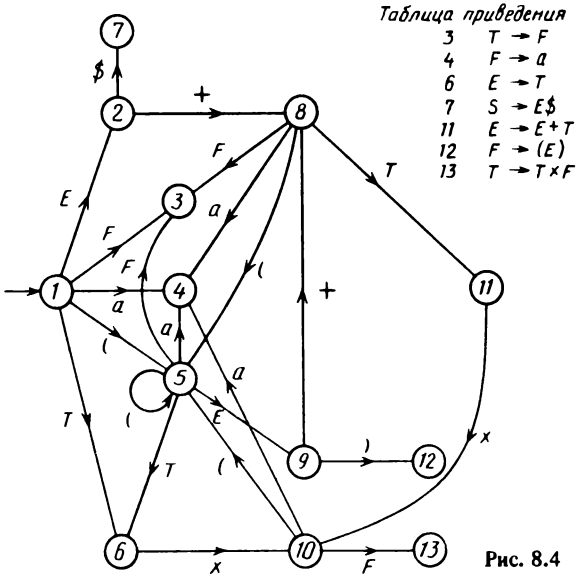


Рис. 8.4

за и разрешения конфликтных ситуаций с помощью дополнительной информации о виде очередного символа входной строки). Вообще, если грамматика является $LR(k)$-грамматикой ($k \geqslant 0$ – целое число), конфликтные ситуации, возникающие при функционировании ее характеристического автомата, должны разрешаться с помощью дополнительной информации о виде k очередных входных символов. Случай $k = 1$ очень важен, поэтому мы остановимся на нем подробно.

Пусть $G = (N, T, P, S)$ – произвольная пополненная грамматика. Единственная продукция этой грамматики, содержащая символ \$, имеет вид $S \to E\$$, $E \in N$. Определим LR(1)CONTEXT-множество для продукции $A \to \beta$:

$$\mathrm{LR(1)CONTEXT}(A \to \beta) = \{\alpha \mid \alpha = \gamma\beta a \in (N \cup T)^* T, \text{ где } S \overset{*}{\Rightarrow} \gamma A a x \Rightarrow \Rightarrow \gamma\beta a x - \text{правосторонняя схема вывода}; a \in T,\ x \in T^*,\ \gamma \in (N \cup T)^*\}.$$

Если $\alpha \in \mathrm{LRCONTEXT}(A \to \beta)$, то $\alpha = \alpha' a$ для некоторых $\alpha' \in \mathrm{LRCONTEXT}(A \to \beta)$ и $a \in T$ – символ, информация о виде которого так необходима анализатору. Обобщая теорему 8.1, сформулируем следующее определение:

пусть $G = (N, T, P, S)$ – пополненная контекстно свободная грамматика, G называют *$LR(1)$-грамматикой* $\Leftrightarrow \alpha \in \mathrm{LR(1)CONTEXT}(A \to \beta)$ и $\alpha x \in \in \mathrm{LR(1)CONTEXT}(B \to \beta')$, где $A \to \beta$, $B \to \beta' \in P$, $\alpha \in (N \cup T)^*$ и $x \in T^*$, то $x = \varepsilon$, $A = B$ и $\beta = \beta'$.

Таким образом, для того чтобы проверить, является ли рассмотренная грамматика G $LR(1)$-грамматикой, необходимо найти LR(1)CONTEXT-множества для каждой продукции этой грамматики. Пусть $A \in N$ и $a \in \in \mathrm{FOLLOW}(A)$, определим новую функцию:

$$\mathrm{LEFT}_1(A, a) = \{\gamma \mid S \overset{*}{\Rightarrow} \gamma A a x \ - \text{правосторонняя схема вывода}, \ x \in T^*\}.$$

Из этого следует, что

$$\mathrm{LR(1)CONTEXT}(A \to \beta) = \bigcup_{a \in \mathrm{FOLLOW}(A)} \mathrm{LEFT}_1(A, a) \cdot \{\beta a\}.$$

Найдем теперь все LEFT_1-множества. Сделать это можно аналогично тому, как мы находили все LEFT-множества. Прежде всего, если $B \to \gamma_1 A \gamma_2$ – продукция грамматики G, $b \in \mathrm{FOLLOW}(B)$, а $a \in \{\mathrm{FIRST}(x) \mid \gamma_2 b \overset{*}{\Rightarrow} x\}$, то, очевидно, $a \in \mathrm{FOLLOW}(A)$ и $\mathrm{LEFT}_1(A, a) \supset \mathrm{LEFT}_1(B, b) \cdot \{\gamma_1\}$.

Таким образом, можно построить систему уравнений для LEFT_1-множества и, как в случае LEFT-множеств, полученные уравнения будут соответствовать линейной слева грамматике. Таким образом, LEFT_1-множества регулярны, а следовательно, LR(1)CONTEXT-множества также регулярны. Исходя из грамматики, определяющей LEFT_1-множества, можно построить детерминированный конечный автомат, который, в свою очередь, будет определять LR(1)CONTEXT-множества. Если рассматриваемая грамматика является $LR(1)$-грамматикой, то построенный детерминированный конечный автомат можно использовать для построения синтаксического анализатора аналогично тому, как был использован характеристический автомат $LR(0)$-грамматики. К сожалению, автомат, полученный для $LR(1)$-грамматики, как правило, имеет очень много состояний, и потому построить его довольно-

но трудно. В связи с этим доказать, что данная пополненная грамматика является (или не является) $LR(1)$-грамматикой, иногда проще каким-либо другим способом, например наложив на эту грамматику ряд довольно сильных ограничений, называемых также *условиями простоты* $LR(1)$-грамматики. Определим теперь функцию SLR(1)CONTEXT[1] на множестве, состоящем из продукции $A \to \beta$ грамматики, следующим образом:

$$\text{SLR(1)CONTEXT}(A \to \beta) = \text{LRCONTEXT}(A \to \beta) \cdot \text{FOLLOW}(A).$$

Согласно определению, LR(1)CONTEXT-множество некоторой продукции является подмножеством ее SLR(1)CONTEXT-множества.

Итак, пополненная грамматика $G = (N, T, P, S)$ является *простой* $LR(1)$-грамматикой $\Leftrightarrow \alpha \in$ SLR(1)CONTEXT $(A \to \beta)$ и $\alpha x \in$ SLR(1)CONTEXT $(B \to \beta_1)$, где $A \to \beta$, $B \to \beta' \in P$, $\alpha \in (N \cup T)^*$, $x \in T^*$, то $x = \varepsilon$, $A = B$, $\beta = \beta'$. Непосредственным следствием этих определений является следующая теорема.

ТЕОРЕМА 8.4

Любая $SLR(1)$-грамматика является $LR(1)$-грамматикой. Обратное неверно (см. упр. 8.5). Пусть G_4 – $SLR(1)$-грамматика. С помощью методов, описанных в гл. 7, легко найти соответствующие ей FOLLOW-множества:

$$\text{FOLLOW}(S) = \varnothing,$$

$$\text{FOLLOW}(E) = \{+,), \$\},$$

$$\text{FOLLOW}(T) = \{\times, +,), \$\},$$

$$\text{FOLLOW}(F) = \{\times, +,), \$\}.$$

И, следовательно, SLR(1)CONTEXT-множества грамматики, характеристический автомат которой изображен на рис. 8.4, удовлетворяют условиям простоты. Воспользуемся этим выводом и построим LR-анализатор для грамматики G_4. Известно, что любой возникший при функционировании характеристического автомата конфликт может быть разрешен знанием вида очередного входного символа. Проиллюстрируем сказанное, осуществив синтаксический разбор строки $a + a \times a + a \$$ (табл. 8.7).

Итак, в результате разбора с помощью построенного синтаксического анализатора будет получено дерево вывода. Дополнить анализатор функциями семантического разбора можно без особого труда. Предположим, например, что перед нами стоит задача получить семантические деревья, соответствующие арифметическим выражениям, порожденным грамматикой G_4. Это можно сделать, например, поставив в соответствие каждому элементу X текущего префикса сентенциальной формы семантическое дерево, соответствующее поддереву дерева вывода, у которого символом X помечен корень (обозначим его T_X). В таком случае после осуществления операции сдвига к текущему префиксу присоединяется еще один символ, например a. Соответствующее этому символу дерево T_a имеет, очевидно, лишь один узел ⓐ. Если же осуществлена операция приведения с помощью продукции $A \to X_1 X_2 \ldots X_k$, то самая правая подстрока текущего префикса сентенциаль-

[1] В аббревиатуре SLR буква S от слова Symple – простой. – *Прим. ред.*

Таблица 8.7

Содержимое стека	Выполняемое действие
1	Сдвиг стека с занесением символа *a*
1, 4	Приведение с помощью продукции $F \to a$
1, 3	Приведение с помощью продукции $T \to F$
1, 6	Приведение с помощью продукции $E \to T$ (просмотр входной строки на один символ вперед)
1, 2	Сдвиг стека с занесением символа +
1, 2, 8	Сдвиг стека с занесением символа *a*
1, 2, 8, 4	Приведение с помощью продукции $F \to a$
1, 2, 8, 3	Приведение с помощью продукции $T \to F$
1, 2, 8, 11	Сдвиг стека с занесением символа × (просмотр входной строки на один символ вперед)
1, 2, 8, 11, 10	Сдвиг стека с занесением символа *a*
1, 2, 8, 11, 10, 4	Приведение с помощью продукции $F \to a$
1, 2, 8, 11, 10, 13	Приведение с помощью продукции $T \to T \times F$
1, 2, 8, 11	Приведение с помощью продукции $E \to E + T$ (просмотр входной строки на один символ вперед)
1, 2	Сдвиг стека с занесением символа +
1, 2, 8	Сдвиг стека с занесением символа *a*
1, 2, 8, 4	Приведение с помощью продукции $F \to a$
1, 2, 8, 3	Приведение с помощью продукции $T \to F$
1, 2, 8, 11	Приведение с помощью продукции $E \to E + T$ (просмотр входной строки на один символ вперед)
1, 2	Сдвиг стека с занесением символа $
1, 2, 7	Строка принята

ной формы, совпадающая со строкой $X_1 X_2 \ldots X_k$, будет заменена на символ A. *Семантическим правилом,* ассоциированным с такой операцией приведения, называют правило, описывающее дерево T_A как совокупность поддеревьев $T_{X_1}, T_{X_2}, \ldots, T_{X_k}$. Достаточным условием успешного завершения синтаксического разбора является осуществление перемещения, в результате которого текущий префикс сентенциальной формы грамматики будет содержать лишь один символ S, а одновременно с деревом вывода будет получено и семантическое дерево T_S.

В табл. 8.8 приведены семантические правила, ассоциированные с продукциями грамматики G_4.

Таблица 8.8

Приведение	Семантическое правило
$S \to E\$$	$T_S := T_E$
$E \to E + T$	$T_E := дерево(+, T_E, T_T)$
$E \to T$	$T_E := T_T$
$T \to T \times F$	$T_T := дерево(\times, T_T, T_F)$
$T \to F$	$T_T := T_F$
$F \to a$	$T_F := T_a$
$F \to (E)$	$T_F := T_E$

ТЕОРЕТИЧЕСКИЕ РАССУЖДЕНИЯ

Синтаксический LR-разбор теоретически по своим возможностям превосходит LL-разбор, поскольку, как было показано ранее, любая пополненная $LL(k)$-грамматика является $LR(k)$-грамматикой. Однако существуют грамматики (см. упр. 8.6), которые, являясь $LL(1)$-грамматиками, не являются $SLR(1)$-грамматиками. Любая $LR(k)$-грамматика обязательно порождает детерминированный язык, но, кроме того, любой детерминированный язык может быть порожден некоторой $LR(1)$-грамматикой. Таким образом, язык порожден $LR(k)$-грамматикой $\Leftrightarrow$ он порожден некоторой $LR(1)$-грамматикой. Если язык удовлетворяет *условию собственности префиксов*, то он может быть порожден $LR(0)$-грамматикой. Условие собственности префиксов означает, что если x – строка некоторого языка, то никакой ее собственный префикс не принадлежит этому же языку. Таким образом, если грамматика использует маркер конца ввода \$, то всякий порожденный ею язык удовлетворяет условию собственности префиксов и, следовательно, может быть порожден $LR(0)$-грамматикой. На практике разработчики компиляторов, как правило, используют $LR(1)$-грамматики, предпочитая им, однако, более простой случай $SLR(1)$-грамматики. Если у вас будет потребность убедиться в справедливости этих утверждений или вы захотите заняться теоретическими исследованиями в этой области, обратитесь к книге Хопкрофт Дж. Е., Ульман Дж. Д. Формальные языки и автоматы: Пер. с англ. – М.: Мир, 1982. – 346 с. или к книге Бекхауз Р. Ч. Синтаксис языков программирования: Пер. с англ. – М.: Мир, 1986. – 281 с. Читателям, интересующимся практической разработкой компиляторов, рекомендуем прочитать книгу Ахо А. В., Ульман Дж. Д. Принципы машинного проектирования: Пер. с англ. – М.: Мир, 1983. – 352 с. в которой обсуждается один из вариантов грамматики, известный под названием $LALR$-грамматики.

УПРАЖНЕНИЯ

8.1. Пусть $G = (N, T, P, S)$ – произвольная контекстно свободная грамматика, а L и F – некоторые отношения, определенные на $N \cup T$ следующим образом:

$XFY \Leftrightarrow$ грамматика G имеет продукцию вида $X \to Y\gamma$, $\gamma \in (N \cup T)^*$,

$XLY \Leftrightarrow$ грамматика G имеет продукцию вида $X \to \gamma Y$, $\gamma \in (N \cup T)^*$.

Пусть R^+, R^* и R^{-1} обозначают соответственно транзитивное замыкание отношения R, рефлексивное замыкание транзитивного замыкания отношения R и отношение, обратное отношению R. Покажите, что $(<\cdot) = (\doteq)(F^+)$ и $(\cdot >) = (L^+)^{-1}(\doteq)(F^*)$. Отношения $\doteq$, F_+ и L вычисляются легко. Вычислить отношения F^+ и L^+ можно с помощью алгоритма Уоршелла, опубликованного в журнале JACM за январь 1962 г. $F^* = F^+ \cup I$, где I – отношение тождественности. Исходя из этого легко вычислить отношения $<\cdot$ и $\cdot>$).

8.2. Вычислить перечисленные в предыдущем упражнении отношения для грамматики, имеющей продукции вида

$S \to A\$$

$A \to aABC \mid CB,$

$B \to aB \mid C,$

$C \to b.$

Является ли она грамматикой простого предшествования?

8.3. Покажите, что грамматика, описанная в упр. 8.2, является $LR(0)$-грамматикой, для чего постройте ее характеристический автомат.

8.4. Покажите, что грамматика, имеющая продукции вида

$S \to A,$
$A \to Ab \mid bBa,$
$B \to aAc \mid a \mid aAb,$

не является $LR(0)$-грамматикой, но является $SLR(1)$-грамматикой.

8.5. Пусть $G = (N, T, P, S)$ – произвольная контекстно свободная грамматика и k – положительное целое число. Предположим, что G – пополненная грамматика и каждая строка языка $L(G)$ оканчивается k символами \$. Определим следующую функцию:

$$\text{FOLLOW}_k(A) = \{x \mid \text{длина } (x) = k \text{ и } S \overset{*}{\Rightarrow} y_1 A x y_2 \text{ для некоторых } y_1, y_2 \in T^*\}.$$

В таком случае, SLR (k) CONTEXT – множество продукций $A \to \beta$ грамматики G – может быть определено следующим образом:

$$\text{SLR}(k)\text{CONTEXT}(A \to \beta) = \text{LRCONTEXT}(A \to \beta) \cdot \text{FOLLOW}_k(A).$$

Используя эти определения, определите $SLR(k)$-грамматику, для чего покажите, что грамматика, продукции которой имеют вид

$S \to A\$,$
$A \to BaCD,$
$B \to b,$
$C \to B \mid Ea,$
$D \to ba \mid Dba,$
$E \to b,$

для любого $k > 0$ является $LR(1)$-грамматикой, но не является $SLR(k)$-грамматикой.

8.6. Покажите, что грамматика, имеющая продукции вида

$S \to A\$,$
$A \to BaBb \mid CbCa,$
$B \to \varepsilon,$
$C \to \varepsilon$

является $LL(1)$-грамматикой, но не является $SLR(1)$-грамматикой.

8.7. Обобщите определение $LR(1)$-грамматики на случай $LR(k)$-грамматики.

8.8. Напишите программу на языке программирования Паскаль, которая осуществляет синтаксический разбор простых арифметических выражений. Ваша программа должна разбирать все корректные арифметические выражения, составленные из переменных $a, b, \ldots, z$ и в качестве результата выдавать соответствующие им семантические деревья. В случае некорректного арифметического выражения программа должна выдавать соответствующее сообщение.

ОГЛАВЛЕНИЕ

55 коп.

В. Дж. Рейуорд-Смит

ТЕОРИЯ ФОРМАЛЬНЫХ ЯЗЫКОВ

ВВОДНЫЙ КУРС

В книге автора из Великобритании изложены основы теории формальных языков. Использован математический аппарат теории множеств, теории графов и математической логики. Все сведения, необходимые для понимания рассмотренных в книге вопросов, приведены в соответствующих главах. Удачно подобранные задачи в конце каждой главы не только поясняют, но и дополняют основной материал книги.

Для разработчиков программного обеспечения ЭВМ.