

Министерство образования и науки Российской Федерации
Ярославский государственный университет им. П. Г. Демидова

В. А. Соколов

ВВЕДЕНИЕ В ТЕОРИЮ ФОРМАЛЬНЫХ ЯЗЫКОВ

Учебное пособие

Рекомендовано

*Научно-методическим советом университета для студентов,
обучающихся по направлению
Прикладная математика и информатика*

Ярославль
ЯрГУ
2014

УДК 004.43(075.8)
ББК В185.2я73
С59

*Рекомендовано
Редакционно-издательским советом университета
в качестве учебного издания. План 2014 года*

Рецензенты:

Д. О. Бытев, доктор технических наук, профессор;
кафедра алгебры ЯГПУ им. К. Д. Ушинского

Соколов, Валерий Анатольевич.

С59 Введение в теорию формальных языков : учебное пособие / В. А. Соколов ; Яросл. гос. ун-т им. П. Г. Демидова. – Ярославль : ЯрГУ, 2014. – 208 с.

ISBN 978-5-8397-1035-1

Пособие представляет собой вводный курс по теории формальных языков. В нем приведен материал, составляющий теоретическую основу для разработки языков программирования и конструирования компиляторов и являющийся классическим элементом системы подготовки специалистов в области информатики.

Пособие предназначено для студентов, обучающихся по направлениям 010400.62 Прикладная математика и информатика (дисциплина «Языки программирования и методы трансляции», цикл Б3), очной формы обучения.
Ил. 39.

УДК 004.43(075.8)
ББК В185.2я73

ISBN 978-5-8397-1035-1

© ЯрГУ, 2014

Оглавление

<i>Предисловие</i>	6
<i>Глава 1. Языки и грамматики</i>	8
1.1. Языки.....	8
1.2. Грамматики.....	11
<i>Глава 2. Конечные автоматы</i>	15
2.1. Автоматы.....	15
2.2. Детерминированные конечные автоматы (распознаватели).....	18
2.3. Языки и детерминированные конечные автоматы.....	20
2.4. Недетерминированные конечные автоматы (распознаватели).....	22
2.5. Эквивалентность ДКА и НКА.....	26
2.6. Минимизация конечных автоматов.....	29
<i>Глава 3. Регулярные выражения и регулярные грамматики</i>	35
3.1. Регулярные выражения.....	35
3.2. Связь между регулярными выражениями и автоматными языками.....	38
3.3. Регулярные грамматики.....	44
<i>Глава 4. Свойства регулярных языков</i>	49
4.1. Замкнутость класса регулярных языков.....	49
4.2. Алгоритмические проблемы регулярных языков.....	52
4.3. Лемма о расширении регулярных языков.....	55
<i>Глава 5. Контекстно-свободные языки</i>	58
5.1. Контекстно-свободные грамматики.....	58
5.2. Грамматический разбор.....	64
5.3. Неоднозначность грамматик и языков.....	69

<i>Глава 6. Преобразования КС-грамматик и нормальные</i>	
формы.....	74
6.1. Методы преобразования грамматик.....	74
6.2. Нормальные формы КС-грамматик.....	87
<i>Глава 7. Магазинные автоматы.....</i>	93
7.1. Недетерминированные магазинные автоматы.....	94
7.2. Магазинные автоматы и КС-языки.....	100
7.3. Детерминированные магазинные автоматы и детерминированные КС-языки.....	108
<i>Глава 8. Свойства контекстно-свободных языков.....</i>	113
8.1. Лемма о расширении.....	113
8.2. Свойства замкнутости класса контекстно-свободных языков.....	116
8.3. Некоторые алгоритмические проблемы для КС-языков.....	120
<i>Глава 9. Классификация и преобразования грамматик.....</i>	124
9.1. Классификация грамматик и языков по Хомскому.....	124
9.2. Примеры грамматик и языков.....	127
9.3. Грамматический разбор и преобразования грамматик.....	128
<i>Глава 10. Методы построения трансляторов. Лексический</i>	
анализ.....	131
10.1. Описание модельного языка.....	131
10.2. Лексический анализ.....	133
10.3. О недетерминированном разборе.....	138
10.4. Задачи лексического анализа.....	139
10.5 Лексический анализатор для М-языка.....	140

<i>Глава 11. Синтаксический и семантический анализ.....</i>	<i>147</i>
11.1. Метод рекурсивного спуска.....	148
11.2. О применимости метода рекурсивного спуска.....	151
11.3. Синтаксический анализатор для М-языка.....	156
11.4. О семантическом анализе.....	160
11.5. Семантический анализатор для М-языка.....	163
11.6. Контроль контекстных условий в операторах.....	169
<i>Глава 12. Генерация внутреннего представления программ.....</i>	<i>173</i>
12.1. Язык внутреннего представления программы.....	173
12.2. Синтаксически управляемый перевод.....	177
12.3. Генератор внутреннего представления программы на М-языке.....	178
12.4. Интерпретатор ПОЛИЗа для модельного языка.....	182
<i>Задачи и упражнения.....</i>	<i>186</i>
<i>Литература.....</i>	<i>206</i>

Предисловие

Пособие предназначено для студентов университетов, обучающихся по направлениям «Прикладная математика и информатика» и «Фундаментальная информатика и информационные технологии». В нём представлен материал, составляющий теоретическую основу для разработки языков программирования и конструирования компиляторов и ставший уже классическим элементом системы подготовки студентов в области прикладной математики и информатики.

В пособии рассматриваются детерминированные и недетерминированные конечные автоматы, регулярные и контекстно-свободные языки, регулярные выражения, формальные грамматики и выводимость в них, свойства замкнутости различных классов языков, их структурные свойства, магазинные автоматы и их связь с контекстно-свободными языками, описаны некоторые методы трансляции на примере модельного языка, а также даются задания для практических и лабораторных работ по соответствующим темам.

Пособие представляет собой расширенный предыдущий вариант книги автора [11], дополненный слегка переработанными четырьмя главами из [9], иллюстрирующими некоторые методы трансляции языков. Объем теоретического материала первых восьми глав примерно соответствует семестровому лекционному курсу, который автор в течение ряда лет читал на факультете информатики и вычислительной техники Ярославского государственного университета им. П. Г. Демидова.

При выборе способа подачи материала предпочтение отдается простым, наглядным, но строгим рассуждениям. Доказательства теорем, как правило, опираются на алгоритмические конструкции. В основу критерия отбора материала положен принцип «минимальной достаточности», при этом имеется в виду существование изданий энциклопедического характера, таких как двухтомная монография А. Ахо и Дж. Ульмана «Теория синтаксического анализа, перевода и компиляции» (М.: Мир, 1978), а также книга Дж. Хопкрофта, Р. Мотвани, Дж. Ульмана «Вве-

дение в теорию автоматов, языков и вычислений» (2-е изд. М., 2002), которые могут служить источником дополнительных сведений.

Цель пособия – дать студентам возможность быстро войти в круг идей, понятий и основных результатов теории формальных языков и методов трансляции языков программирования.

Автор выражает благодарность Марии Сергеевне Комар за большую помощь при подготовке рукописи к печати.

Глава 1. Языки и грамматики

1.1. Языки

Любой язык основан на использовании определенного алфавита. *Алфавит* – это конечное непустое множество символов, например,

$$\Sigma = \{a_1, a_2, a_3, \dots, a_n\}, n > 0.$$

Строка – это упорядоченная конечная последовательность символов алфавита Σ . Для обозначения строк будем использовать строчные буквы греческого алфавита $\alpha, \beta, \gamma, \dots$. Например, $\alpha = aabb$ означает, что строка, обозначаемая буквой α , представляет собой последовательность из четырех символов $aabb$.

Конкатенация двух строк α и β – это бинарная операция, результатом которой является строка $\alpha\beta$, полученная приписыванием к строке α справа строки β , то есть если $\alpha = a_1 a_2 a_3 \dots a_n$, $\beta = b_1 b_2 b_3 \dots b_m$, то

$$\alpha\beta = a_1 a_2 a_3 \dots a_n b_1 b_2 b_3 \dots b_m.$$

Очевидно, что конкатенация является ассоциативной, но некоммутативной операцией, т. е. для любых строк α, β, γ в алфавите Σ справедливо равенство

$$\alpha(\beta\gamma) = (\alpha\beta)\gamma,$$

но неверно, что для любых α, β

$$\alpha\beta = \beta\alpha.$$

Например, если $\alpha = ab$, $\beta = ba$, то $\alpha\beta = abba$, а $\beta\alpha = baab$, то есть $\alpha\beta \neq \beta\alpha$.

Операция *обращения* строки α обозначается α^{-1} и дает строку, полученную из α выписыванием всех входящих в нее символов в обратном порядке, т. е. если $\alpha = a_1 a_2 a_3 \dots a_n$, то $\alpha^{-1} = a_n a_{n-1} \dots a_2 a_1$. Нетрудно видеть, что для любых строк α и β верно соотношение

$$(\alpha\beta)^{-1} = \beta^{-1}\alpha^{-1}.$$

Длина строки α обозначается $|\alpha|$ и равна числу символов (с учетом повторений) в этой строке. Например, если $\alpha = a_1 a_2 a_3 \dots a_n$, то $|\alpha| = n$, $n \geq 0$.

Пустой строкой называется такая строка ε , длина которой $|\varepsilon| = 0$. Очевидно, что для любой строки $\alpha\varepsilon\alpha = \alpha\varepsilon = \alpha$, т. е.

строка ε является нейтральным элементом относительно конкатенации. В разных ситуациях бывает необходимо рассматривать структуру строки α , то есть выделять в ней отдельные части, которые сами являются строками. Пусть

$$\alpha = \beta\gamma\delta, \text{ где } |\beta| \geq 0, |\gamma| \geq 0, |\delta| \geq 0.$$

Тогда строка β называется префиксом строки α , строка δ – её *суффиксом*, а γ – её *подстрокой*. Отсюда, в частности, следует, что пустая строка ε является подстрокой любой строки α .

Длина строки связана с операцией конкатенации простым соотношением: $|\alpha\beta| = |\alpha| + |\beta|$.

Введем полезное обозначение для результата конкатенации строки α с самой собой:

$$\alpha\alpha = \alpha^2, \alpha\alpha\alpha = \alpha^3 \text{ и т. д.}$$

В общем виде эту операцию над строкой α можно определить рекурсивно: $\alpha^0 = \varepsilon$, $\alpha^{n+1} = \alpha^n\alpha$, $n \geq 0$. В частности, если $\alpha = aa \dots a$ и $|\alpha| = n$, то $\alpha = a^n$.

Теперь введем одно из основных понятий информатики – понятие формального языка.

Формальным языком называется любое множество строк в данном алфавите Σ .

Пример 1.

Пусть $\Sigma = \{a, b\}$. Тогда множество

$$L = \{a, b, aa, bb, ab, aab\}$$

является (конечным) формальным языком в алфавите Σ .

В дальнейшем слово «язык» будет обозначать формальный язык в некотором заранее фиксированном алфавите.

Пример 2.

Множество $L = \{a^n b^n \mid n \geq 0\}$ является (бесконечным) языком в алфавите $\Sigma = \{a, b\}$. Его элементами являются строки вида ab , $aabb$, $aaabbb$, ... , в том числе пустая строка ε .

Заметим, что не следует путать пустой язык $L = \emptyset$, не содержащий ни одной строки, и язык $L = \{\varepsilon\}$, состоящий из единственной (пустой) строки ε .

Язык, состоящий из всех строк в алфавите Σ , обозначается Σ^* . Если L – язык в алфавите Σ , то, очевидно, $L \subseteq \Sigma^*$.

Так как языки – это множества (строк), то над ними можно совершать обычные теоретико-множественные операции объединения \cup , пересечения \cap , разности \setminus и образования дополнения (по отношению к Σ^*), т. е. если L – язык в алфавите Σ , то его *дополнение* есть множество $\bar{L} = \Sigma^* \setminus L$.

Наряду с этими операциями над языками можно определить еще ряд специфических операций.

Пусть L_1, L_2 – языки в алфавите Σ , тогда их *сцепление* – это язык

$$L_1 \cdot L_2 = \{\alpha\beta \mid \alpha \in L_1, \beta \in L_2\},$$

состоящий из всевозможных конкатенаций строк α языка L_1 и строк β языка L_2 , образованных в указанном порядке.

Нетрудно заметить, что операция сцепления двух языков ассоциативна, но не коммутативна, то есть для любых языков L_1, L_2, L_3 из Σ^*

$$(L_1 \cdot L_2) \cdot L_3 = L_1 \cdot (L_2 \cdot L_3),$$

но существует пара языков L_1, L_2 такая, что $L_1 \cdot L_2 \neq L_2 \cdot L_1$.

Определим еще одну операцию – итерацию языка L :

$$L^0 = \{\varepsilon\}, L^1 = L, L^2 = L \cdot L, L^3 = L \cdot L \cdot L, \dots$$

В общем виде операцию L^n ($n \geq 0$) можно определить рекурсивно следующим образом: $L^0 = \{\varepsilon\}, L^{n+1} = L^n \cdot L, n \geq 0$.

Пример 3.

Если $L = \{a^n b^n \mid n \geq 0\}$, то $L^2 = \{a^n b^n a^m b^m \mid n \geq 0, m \geq 0\}$.

Полезно отметить, что если итерировать алфавит Σ , который сам, в свою очередь, является (конечным) языком, все строки которого имеют длину 1, то последовательно будем иметь:

$$\Sigma^0 = \{\varepsilon\} = \{\alpha \mid \alpha \in \Sigma^* \ \& \ |\alpha| = 0\},$$

$$\Sigma^1 = \Sigma = \{\alpha \mid \alpha \in \Sigma^* \ \& \ |\alpha| = 1\},$$

$$\Sigma^2 = \{\alpha \mid \alpha \in \Sigma^* \ \& \ |\alpha| = 2\} = \{a_1 a_2 \mid a_1 \in \Sigma, a_2 \in \Sigma\},$$

и т. д.

В общем случае, $\Sigma^n = \{\alpha \mid \alpha \in \Sigma^* \ \text{и} \ |\alpha| = n\}$, $n \geq 0$, т. е. Σ^n – это множество всех строк длины n в алфавите Σ .

В заключение этого раздела введем еще одну очень важную операцию над языками – так называемое *закрывание Клини* (или *звездочка*), которая определяется следующим образом. Пусть L – произвольный язык, тогда

$$L^* = L^0 \cup L^1 \cup L^2 \cup \dots = \bigcup_{i=0}^{\infty} L^i.$$

Для удобства, чтобы отделить пустую строку ε , которая всегда принадлежит L^* , иногда используется *позитивное замыкание* языка L , обозначаемое L^+ :

$$L^+ = \bigcup_{i=0}^{\infty} L^i = L^1 \cup L^2 \cup \dots$$

Из определения следует, что $L^* = \{\varepsilon\} \cup L^+$.

Заметим, что ранее введенное обозначение языка Σ^* вполне согласуется с только что введенной операцией замыкания: объединение $\Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \dots$ как раз и представляет собой множество всех строк в алфавите Σ (включая пустую строку).

1.2. Грамматики

Грамматика – это один из точных способов задания языка. Грамматические правила позволяют определять, является ли та или иная языковая конструкция правильной с точки зрения данного языка, то есть принадлежит ли она этому языку или нет.

Пример 4.

Рассмотрим фрагмент грамматики (обозначим его G_A) языка программирования, описывающий все арифметические выражения, которые можно построить, используя идентификаторы a, b, c :

<выражение> ::= <терм> | <выражение> + <терм> |

<выражение> - <терм>

<терм> ::= <множитель> | <терм> * <множитель> |

<терм> / <множитель>

<множитель> ::= a | b | c | (<выражение>)

Эту стандартную запись в форме Бэкуса-Наура можно упростить, введя буквенные обозначения для понятий, входящих в грамматические правила. Положим

<выражение> = E ,

<терм> = T ,

<множитель> = F .

Тогда G_A будет выглядеть следующим образом:

$E \rightarrow T \mid E + T \mid E - T$

$T \rightarrow F \mid T * F \mid T / F$

$F \rightarrow a \mid b \mid c \mid (E)$

Определение 1.

Грамматикой G называется следующая упорядоченная четверка объектов $G = (N, T, S, P)$, где

N – конечное множество нетерминальных символов, называемых *переменными* (или *нетерминалами*);

T – конечное множество терминальных символов, называемых *терминалами*;

S – начальный символ, $S \in N$;

P – конечное множество правил вывода, называемых *продукциями*. Здесь и далее предполагается, что множества N и T непустые и не пересекаются.

Продукции – это основной механизм грамматики, с помощью которого можно преобразовывать одну строку символов в другую и таким образом порождать язык, определяемый данной грамматикой. Каждая продукция $p \in P$ имеет вид $\alpha \rightarrow \beta$, где

$$\alpha \in (N \cup T)^+, \beta \in (N \cup T)^*.$$

Результатом *применения* продукции $p: \alpha \rightarrow \beta$ к строке $\gamma = \varphi\alpha\psi$ является строка $\delta = \varphi\beta\psi$, полученная из γ подстановкой вместо подстроки α (левой части продукции) строки β (правой части продукции).

Это преобразование записывается так: $\gamma \Rightarrow_p \delta$, или $\gamma \Rightarrow_G \delta$, когда просто хотят сказать, что δ получается из γ применением некоторой продукции из грамматики G . В этом случае говорят, что строка δ выводится из строки γ (в грамматике G) применением продукции p .

Если применять продукции p_1, p_2, \dots, p_{n-1} грамматики G последовательно к получающимся строкам, то это можно записать в виде:

$$\alpha_1 \Rightarrow_{p_1} \alpha_2 \Rightarrow_{p_2} \alpha_3 \Rightarrow_{p_3} \dots \Rightarrow_{p_{n-1}} \alpha_n.$$

В этом случае говорят, что строка α_n выводится из строки α_1 (в грамматике G) и пишут $\alpha_1 \Rightarrow_G^* \alpha_n$. Символ $*$ указывает здесь любое конечное число шагов, включая 0, то есть случай $\alpha \Rightarrow_G^* \alpha$. Если же надо указать, что, по крайней мере, одна продукция в действительности должна быть применена, то пишут $\alpha \Rightarrow_G^+ \beta$.

Грамматика определяет тот язык, строки которого могут быть выведены из некоторой фиксированной строки (начального элемента) применением продукций этой грамматики.

Определение 2.

Пусть $G = (N, T, S, P)$ – грамматика. Тогда множество

$$L(G) = \{\alpha \mid \alpha \in T^* \text{ и } S \Rightarrow_G^* \alpha\}$$

называется *языком, порождаемым грамматикой G*.

Если $\alpha \in (N \cup T)^*$, то любая последовательность вида

$$S \Rightarrow_G \alpha_1 \Rightarrow_G \alpha_2 \Rightarrow_G \alpha_3 \Rightarrow_G \dots \Rightarrow_G \alpha_n \Rightarrow_G \alpha$$

называется *выводом* строки α в грамматике G ; при этом если $\alpha \in T^*$, то α называется *сентенцией*, а α_i – сентенциальной формой (в грамматике G).

Если известно, в какой грамматике G рассматривается вывод $\alpha \Rightarrow_G^* \beta$, то можно не писать символ G .

Пример 5.

Рассмотрим грамматику

$$G = (\{S\}, \{a, b\}, S, P),$$

где P состоит из двух продукций:

$$S \rightarrow aSb,$$

$$S \rightarrow \varepsilon.$$

Тогда, очевидно, в G можно построить вывод:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb, \text{ то есть } S \Rightarrow_G^* a^2b^2.$$

Следовательно, строка $a^2b^2 \in L(G)$.

Как получить явное описание этого языка? К сожалению, это не всегда возможно. В данном случае можно показать, что $L(G) = \{a^n b^n \mid n \geq 0\}$. Докажем это по индукции. В качестве параметра n индукции возьмем длину вывода, то есть число применений продукций в последовательности $S \Rightarrow \alpha_1 \Rightarrow \dots \Rightarrow \alpha_n$. Покажем, что получающиеся при этом сентенциальные формы α_n имеют вид $a^n S b^n$ или $a^{n-1} b^{n-1}$, $n \geq 1$.

Пусть $n = 1$, тогда либо $\alpha_1 = aSb$, либо $\alpha_1 = \varepsilon$, что составляет базу индукции.

Допустим, что наше утверждение верно для всех $n \leq k$. Пусть теперь $n = k + 1$, $k \geq 0$. Рассмотрим вывод длины $k + 1$: $S \Rightarrow \alpha_1 \Rightarrow \dots \Rightarrow \alpha^k \Rightarrow \alpha^{k+1}$. По нашему предположению, $\alpha^k = a^k S b^k$, так как к строке $a^{k-1} b^{k-1}$ никакой продукции применить уже нельзя. На $(k+1)$ -м шаге возможно применение одной из двух продукций грамматики G . Если применить первую, то получим строку $a^{k+1} S b^{k+1}$, а если вторую, то строку $a^k b^k$, что согласуется с предположением индукции. Таким образом, наше утверждение верно, и ничего другого, кроме строк вида $a^n S b^n$ или $a^n b^n$, из S получить невозможно. В то же время легко заметить, что любая строка вида $a^n b^n$ для $n \geq 0$ выводима из S .

Может случиться, что две разные грамматики порождают один и тот же язык. Так, если в грамматику G из примера 5 добавить продукцию $S \rightarrow A$, то это не изменит язык (хотя и изменит множество сентенциальных форм).

Две грамматики G_1 и G_2 называются эквивалентными, если они порождают один и тот же язык, то есть если $L(G_1) = L(G_2)$.

Глава 2. Конечные автоматы

2.1. Автоматы

Автомат можно рассматривать как упрощенную абстрактную модель компьютера. Он имеет способность воспринимать *входную информацию*, которую мы будем представлять в виде строки символов в фиксированном алфавите и которую автомат может лишь читать, но не может изменять. Предполагается, что *входная строка* записана на ленте, разделенной на клеточки, каждая из которых может содержать не более одного символа. Автомат может различать конец входной строки (например, отмеченный специальным символом) и реагировать на прочитанную информацию тем или иным способом. У него может быть *устройство временной памяти*, содержащее потенциально неограниченное число ячеек, в каждую из которых может быть записано не более одного символа из некоторого алфавита (вообще говоря, не совпадающего с входным). Автомат может читать и изменять содержимое ячеек временной памяти. Наконец, автомат имеет *устройство управления*, которое может находиться в одном из конечного множества *внутренних состояний* и изменять его по мере необходимости.

Свойства автомата самым существенным образом зависят от характера временной памяти. Предполагается, что автомат функционирует в дискретном времени. В каждый момент времени устройство управления находится в некотором внутреннем состоянии, а читающая головка обзревает очередной символ на входной ленте. В следующий момент времени устройство управления переходит в некоторое новое состояние в зависимости от предыдущего состояния, прочитанного символа на входной ленте и содержимого временной памяти.

Это соответствие задает *функцию перехода автомата*, которая определяет его возможное поведение. В результате перехода автомата из одного состояния в другое может появиться выходная информация или может быть записана новая информация во временную память.

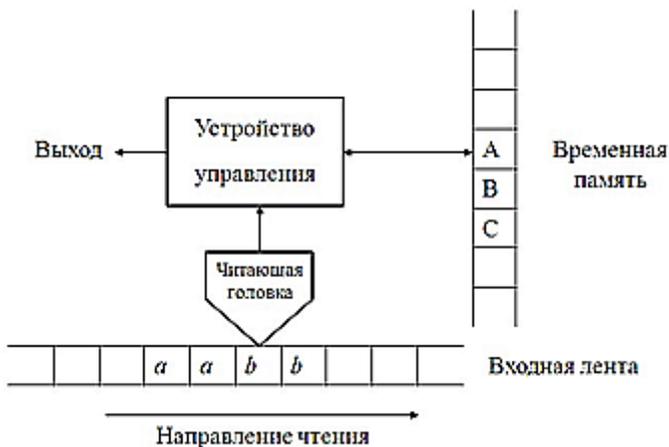


Рис. 1. Схема автомата

Состояние устройства управления, прочитанный входной символ и содержимое временной памяти определяют то, что называется конфигурацией автомата в данный момент времени. Переход от одной конфигурации в момент времени t к другой конфигурации в момент $t + 1$ называется *шагом*. Таким образом, функционирование автомата состоит из последовательности шагов (конечной или бесконечной). Различают два типа автоматов: *детерминированные* и *недетерминированные*. Поведение детерминированного автомата в каждый следующий момент времени полностью предопределено его внутренним состоянием в предыдущий момент, прочитанным символом и содержимым временной памяти, то есть если автомат может совершить следующий шаг, то он будет единственно возможным шагом.

Для недетерминированного автомата это не так: в каждый следующий момент времени автомат может выбрать шаг из некоторого (конечного) множества шагов. Таким образом, для недетерминированного автомата мы можем лишь предполагать тот или иной возможный вариант функционирования. Между детерминированными и недетерминированными автоматами существуют интересные с точки зрения формальных языков связи.

Автомат, выход которого ограничен двумя сообщениями {да, нет}, называется *распознавателем*. Реагируя на входную строку, распознаватель или распознает ее (допускает), или отвергает. Этот класс автоматов представляет наибольший интерес с точки зрения формальных языков и грамматик. Автомат более общего типа, выдающий на выходе целую строку символов, называется *преобразователем* (этот тип автоматов здесь не рассматривается).

Пример 6.

Рассмотрим грамматику G_1 , порождающую подязык идентификаторов L_1 в языке Паскаль:

<идентификатор> \rightarrow <буква><хвост>
 <хвост> \rightarrow <буква><хвост> | <цифра><хвост> | ϵ
 <буква> $\rightarrow a | b | c | \dots | A | B | \dots | Z$
 <цифра> $\rightarrow 0 | 1 | 2 | \dots | 9$

В этой грамматике G_1 :

$N = \{ \langle \text{идентификатор} \rangle, \langle \text{буква} \rangle, \langle \text{цифра} \rangle, \langle \text{хвост} \rangle \}$,

$T = \{ a, b, \dots, A, B, \dots, Z, 0, 1, 2, \dots, 9, \epsilon \}$,

$S = \langle \text{идентификатор} \rangle$.

Вывод идентификатора А1 выглядит так:

<идентификатор> \Rightarrow <буква><хвост> \Rightarrow А<хвост> \Rightarrow А<цифра>
 \Rightarrow <хвост> \Rightarrow А<цифра> $\epsilon \Rightarrow$ А1.

Построим автомат, распознающий язык L_1 и ничего другого:

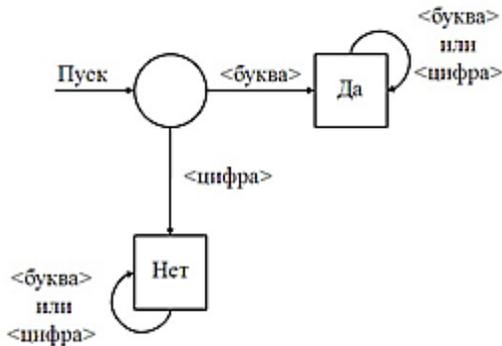


Рис. 2. Автомат, распознающий язык L_1

2.2. Детерминированные конечные автоматы (распознаватели)

Рассмотрим самый простой случай, когда у автомата отсутствует дополнительная временная память, устройство управления может находиться в одном из конечного множества состояний, а реакция на входную строку выражается ответами «да» или «нет».

Определение 3.

Детерминированный конечный автомат (распознаватель), или сокращенно ДКА, – это пятерка

$$M = (Q, \Sigma, \theta, q_0, F), \text{ где}$$

Q – конечное множество внутренних состояний;

Σ – конечное множество символов, называемое входным алфавитом;

$\theta: Q \times \Sigma \rightarrow Q$ – всюду определенная функция, называемая функцией переходов;

$q_0 \in Q$ – начальное состояние;

$F \subseteq Q$ – множество заключительных состояний.

В дальнейшем под ДКА мы будем понимать только детерминированный конечный распознаватель. ДКА функционирует следующим образом. В начальный момент времени, находясь в состоянии q_0 , он с помощью читающей головки обозревает самый левый символ входной строки. Головка читает строку слева направо, без возвратов, сдвигаясь вправо на один символ за каждый шаг работы автомата. В соответствии с заданной функцией переходов автомат изменяет свои внутренние состояния в процессе чтения входной строки. Прочитав ее до конца, автомат оказывается либо в одном из заключительных состояний (и в этом случае строка будет допущена или распознана автоматом), либо в каком-то другом, не заключительном (и в этом случае строка отвергается).

Для наглядного представления функционирования автомата используются *диаграммы переходов*. Это ориентированные конечные графы, в которых вершины соответствуют внутренним состояниям автомата (и помечаются соответствующими символами).

Если $M = (Q, \Sigma, \theta, q_0, F)$ – ДКА, тогда его диаграмма переходов – это граф DM , имеющий ровно $|Q|$ вершин, каждая из которых помечена некоторым $q_i \in Q$. Для каждого правила перехода $\theta(q_i, a) = q_j$ в DM существует дуга (q_i, q_j) , помеченная символом a . Вершина, помеченная q_0 , называется начальной, а те вершины, метки которых $q_f \in F$, называются финальными (или заключительными). Финальные вершины будем изображать квадратиками, а нефинальные – кружочками.

Нетрудно видеть, что определения ДКА с помощью пятерки $(Q, \Sigma, \theta, q_0, F)$ и посредством диаграммы переходов DM равносильны.

Полезно ввести обобщенную функцию переходов

$$\theta^*: Q \times \Sigma^* \rightarrow Q.$$

Вторым аргументом у θ^* является не отдельный символ, а целая строка, а значение функции указывает на то внутреннее состояние, в котором окажется автомат после серии переходов в результате чтения этой строки.

Пример 7.

Рассмотрим ДКА $M = (\{q_0, q_1, q_2\}, \{a, b\}, \theta, q_0, \{q_1\})$, где

$$\theta(q_0, a) = q_0, \theta(q_0, b) = q_1,$$

$$\theta(q_1, a) = q_1, \theta(q_1, b) = q_2,$$

$$\theta(q_2, a) = q_2, \theta(q_2, b) = q_1.$$

Ему соответствует диаграмма DM :

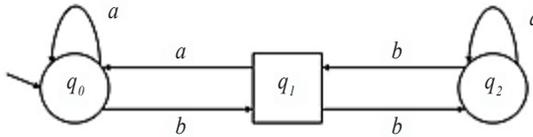


Рис. 3. Диаграмма, соответствующая ДКА

Этот автомат распознает строки $ab, bab, abbb, bbaab$, но отвергает baa или $bbaa$.

В заключение дадим рекурсивное определение функции θ^* :

$$\theta^*(q, \varepsilon) = q,$$

$$\theta^*(q, \alpha a) = \theta(\theta^*(q, \alpha), a) \text{ для всех } q \in Q, \alpha \in \Sigma^*, a \in \Sigma.$$

2.3. Языки

и детерминированные конечные автоматы

Дадим теперь строгое определение языка, связанного с конкретным ДКА.

Определение 4.

Язык, допустимый ДКАМ $= (Q, \Sigma, \theta, q_0, F)$, – это множество всех строк из Σ^* , допустимых автоматом M , то есть $L(M) = \{\alpha \mid \alpha \in \Sigma^* \text{ и } \theta^*(q_0, \alpha) \in F\}$.

Так как функции θ и θ^* являются всюду определенными и каждый шаг работы автомата определяется единственным образом, то любая строка $\alpha \in \Sigma^*$, после ее прочтения, либо допускается (распознается), либо отвергается автоматом. Отсюда следует, что

$$\overline{L(M)} = \{\alpha \mid \alpha \in \Sigma^* \text{ и } \theta^*(q_0, \alpha) \notin F\}.$$

Пример 8.

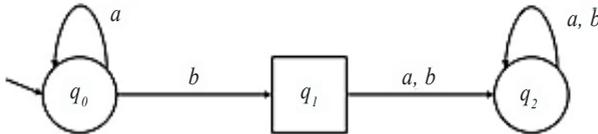


Рис. 4. Пример ДКА

Этой диаграмме соответствует ДКА, допускающий язык $L = \{a^n b \mid n \geq 0\}$.

Теорема 1.

Пусть $M = (Q, \Sigma, \theta, q_0, F)$ – детерминированный конечный автомат (распознаватель) и пусть DM – соответствующая диаграмма переходов. В этом случае для любых $q_p, q_j \in Q$ и $\alpha \in \Sigma^+$ $\theta^*(q_p, \alpha) = q_j$ тогда и только тогда, когда в DM существует путь, помеченный последовательностью символов α , из вершины q_p в вершину q_j .

Доказательство. Проведем его индукцией по длине строки α . Пусть $|\alpha| = 1$, то есть $\alpha = a$ для некоторого $a \in \Sigma$. Тогда $\theta^*(q_p, \alpha) = \theta(q_p, a) = q_j$, по определению, соответствует дуге (q_p, q_j) ,

помеченной a , в D_M . Допустим, что для всех α таких, что $|\alpha| \leq n$, утверждение верно.

Пусть $|\alpha| = n + 1$, и пусть $\theta^*(q_i, \alpha) = q_j$. Можно положить $\alpha = \beta a$, где $|\beta| = n$. Тогда для некоторого q_k

$$\theta^*(q_i, \beta) = q_k \text{ и } \theta^*(q_i, \alpha) = \theta(q_k, a) = q_j.$$

По предположению, равенство $\theta^*(q_i, \beta) = q_k$ эквивалентно существованию пути в DM из q_i в q_k , помеченному β . Кроме того, соотношение $\theta^*(q_k, a) = q_j$ эквивалентно существованию дуги (q_k, q_j) с меткой a , что дает нам в DM путь из q_i в q_j , проходящий через вершину q_k и помеченный строкой βa , который соответствует соотношению $\theta^*(q_i, \alpha) = q_j$. **Теорема доказана.**

Функция θ может быть задана таблицей:

	a_1	a_2	...	a_k	...	a_n
q_0						
q_1						
...						
q_i						
...						
q_m						

Рис. 5. Таблица, задающая функцию θ

Здесь на пересечении i -й строки и k -го столбца находится q_j тогда и только тогда, когда $\theta(q_i, a_k) = q_j$.

Если функция θ не всюду определенная, то ее можно доопределить так, что соответствующий автомату язык не изменится. Для любой пары (q, a) , для которой $\theta(q, a)$ не определено, положим $\theta(q, a) = \Delta$, где $\Delta \notin Q \cup \Sigma$. Кроме того, для любого $a \in \Sigma$ положим $\theta(\Delta, a) = \Delta$. При этом функция θ становится всюду определенной. Так как $\Delta \notin Q$, то $\Delta \notin F$, а отсюда следует, что для любой строки $\alpha \in \Sigma^*$ $\alpha \in L(M)$ тогда и только тогда, когда в модифицированном ДКА $M\Delta\theta^*\Delta(q_0, \alpha) \in F$, что означает, что $\alpha \in L(M\Delta)$. Следовательно, $L(M) = L(M\Delta)$.

Пример 9.

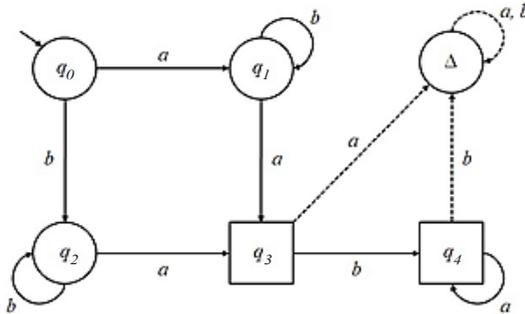


Рис. 6. Пример автомата с «состоянием-ловушкой»

У приведенного на диаграмме автомата функция переходов не является всюду определенной. Вводя новое состояние Δ , являющееся «ловушкой», мы доопределяем эту функцию (достройка диаграммы показана штрихами).

Определение 5.

Язык L называется автоматным, если существует ДКА M такой, что

$$L = L(M).$$

Таким образом, семейство всех ДКА определяет класс автоматных языков.

2.4. Недетерминированные конечные автоматы (распознаватели)

Определение 6.

Недетерминированный конечный автомат (распознаватель), или НККА, – это пятерка

$$M = (Q, \Sigma, \theta, q_0, F),$$

где Q, Σ, q_0, F определяются так же, как и для ДКА, а функция переходов θ выглядит так:

$$\theta: Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow 2^Q.$$

Пример 10.

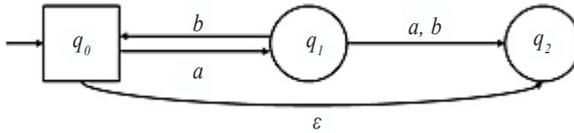


Рис. 7. Пример НКА

Заметим, что здесь

$$\theta(q_0, a) = \{q_1\},$$

$$\theta(q_0, b) = \emptyset,$$

$$\theta(q_1, a) = \{q_2\},$$

$$\theta(q_1, b) = \{q_0, q_2\},$$

$$\theta(q_2, a) = \emptyset,$$

$$\theta(q_2, b) = \emptyset,$$

значит, это НКА.

Определение 7.

Для НКА обобщенная функция переходов θ^* определяется следующим образом: для любых $q_i, q_j \in Q$ и $\alpha \in \Sigma^*$ $\theta^*(q_i, \alpha)$ содержит q_j тогда и только тогда, когда в диаграмме переходов этого НКА существует путь из q_i в q_j , помеченный α .

Определение 8.

Язык, допускаемый НКА M , определяется так:

$$L(M) = \{\alpha \mid \alpha \in \Sigma^* \text{ и } \theta^*(q_0, \alpha) \cap F \neq \emptyset\}.$$

Пример 11.

Рассмотрим НКА, заданный диаграммой:

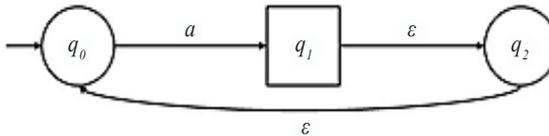


Рис. 8. НКА

Вычислим $\theta^*(q_1, a)$ и $\theta^*(q_2, \varepsilon)$. Рассмотрим сначала все пути из q_1 , соответствующие прочитанному символу a : $\{\varepsilon a(q_1), \varepsilon a \varepsilon(q_2), \varepsilon a \varepsilon \varepsilon(q_0)\}$. В круглых скобках указаны вершины, в которых оканчиваются эти пути. Тогда, по определению,

$$\theta^*(q_1, a) = \{q_0, q_1, q_2\}.$$

Аналогично,

$$\theta^*(q_2, \varepsilon) = \{q_0, q_2\}.$$

Заметим, что для любого $q: q \in \theta^*(q, \varepsilon)$.

От НКА M , содержащего ε -переходы, можно перейти к M' без ε -переходов так, что $L(M) = L(M')$. Для этого нужно для каждой пары (q_p, a) , $q_i \in Q$, $a \in \Sigma$, определить множество всех путей вида

$$\alpha = \varepsilon\varepsilon\dots\varepsilon a \varepsilon\dots\varepsilon,$$

выходящих из вершины q_i , и сформировать множество всех вершин q_j^1, \dots, q_j^k , которыми оканчиваются эти пути.

Вводим новую функцию переходов θ' для НКА M' :

$$\theta'(q_i, a) = \{q_j^1, \dots, q_j^k\},$$

и полагаем

$$M' = (Q, \Sigma, \theta', q_0, F).$$

Нетрудно видеть, что при этом

$$L(M) = L(M').$$

Это дает нам возможность в дальнейшем, при рассмотрении НКА как средства задания языков, считать, что НКА не содержит ε -переходов, то есть

$$\theta: Q \times \Sigma \rightarrow 2Q.$$

Пример 12.

Рассмотрим НКА M , заданный диаграммой D_M :

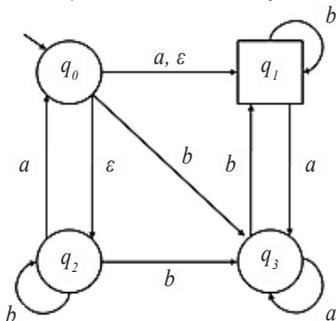


Рис. 9. Диаграмма, задающая НКА M

Здесь $Q = \{q_0, q_1, q_2, q_3\}$, $\Sigma = \{a, b\}$, $F = \{q_1\}$. Построим автомат (недетерминированный) $M' = (Q, \Sigma, q_0, \theta', F')$ без ε -переходов и такой, что $L(M) = L(M')$. Берем начальное состояние q_0

и вычисляем все состояния, в которые M переходит из q_0 по символу a и по символу b :

$$\begin{aligned} q_0 a &\rightarrow q_1, & q_0 b &\rightarrow q_3, \\ q_0 \varepsilon a &\rightarrow q_3, & q_0 \varepsilon b &\rightarrow q_2, \\ q_0 \varepsilon a &\rightarrow q_0, & q_0 \varepsilon b &\rightarrow q_3, \\ q_0 \varepsilon a \varepsilon &\rightarrow q_2, & q_0 \varepsilon b &\rightarrow q_1, \end{aligned}$$

То есть $\theta^*(q_0, a) = \{q_0, q_1, q_2, q_3\}$, $\theta^*(q_0, b) = \{q_1, q_2, q_3\}$.

Теперь то же самое сделаем для q_1 :

$$q_1 a \rightarrow q_3, \quad q_1 b \rightarrow q_1,$$

то есть $\theta^*(q_1, a) = \{q_3\}$, $\theta^*(q_1, b) = \{q_1\}$.

Далее для q_2 :

$$\begin{aligned} q_2 a &\rightarrow q_0, & q_2 b &\rightarrow q_3, \\ q_2 a \varepsilon &\rightarrow q_2, & q_2 b &\rightarrow q_2, \\ q_2 a \varepsilon &\rightarrow q_1, \end{aligned}$$

то есть $\theta^*(q_2, a) = \{q_0, q_1, q_2\}$, $\theta^*(q_2, b) = \{q_2, q_3\}$.

Для q_3 получаем:

$$q_3 a \rightarrow q_3, \quad q_3 b \rightarrow q_1,$$

то есть $\theta^*(q_3, a) = \{q_3\}$, $\theta^*(q_3, b) = \{q_1\}$.

Наконец, заметим, что $q_1 \in \theta(q_0, \varepsilon)$, значит, ε допускается автоматом M , следовательно, q_0 должно быть помещено в F^* .

Таким образом, диаграмма D_M автомата M' будет выглядеть следующим образом:

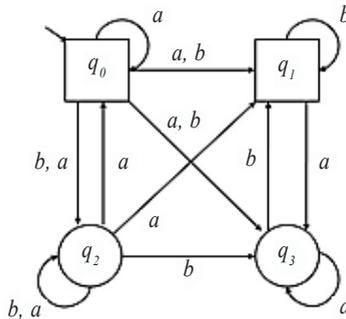


Рис. 10. Автомат M'

Нетрудно видеть, что если некоторая строка $\alpha \in L(M)$, то $\alpha \in L(M')$, и наоборот, то есть $L(M) = L(M')$.

2.5. Эквивалентность ДКА и НКА

Введем отношение эквивалентности на множестве всех конечных автоматов (распознавателей).

Определение 9.

Два автомата эквивалентны, если они допускают один и тот же язык.

Например, автоматы M и M' из предыдущего примера эквивалентны. Пусть LD – класс языков, допускаемых детерминированными конечными автоматами, а LN – класс языков, допускаемых недетерминированными конечными автоматами. Наша цель – показать, что $LD = LN$. В этом смысле понятия ДКА и НКА равносильны. Так как ДКА есть частный случай НКА (когда $|\theta(q, a)| = 1$), то, очевидно, $LD \subset LN$. Покажем, что $LN \subset LD$. Для этого для любого НКА M , допускающего язык $L(M)$, достаточно построить ДКА M' такой, что $L(M') = L(M)$, то есть эквивалентный M .

Рассмотрим преобразование НКА в эквивалентный ДКА сначала на примере.

Пример 13.

Обозначим через M следующий НКА $M = (Q, \Sigma, q_0, \theta, F)$:

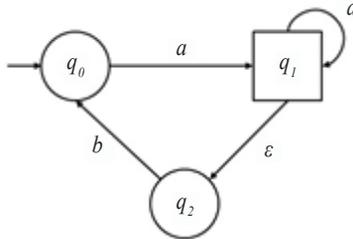


Рис. 11. НКА M

Построим эквивалентный ДКА $M' = (Q', \Sigma, \{q_0'\}, \theta', F')$.

1. Положим $q_0' = \{q_0\}$.
2. $\theta'(q_0', a) = \theta(q_0, a) = \{q_1, q_2\}$, $\theta'(q_0', b) = \theta(q_0, b) = \emptyset$.
3. Положим $q_1' = \{q_1, q_2\}$, $q_2' = \emptyset$.

4. Вычисляем:

$$\theta'(q_1', a) = \theta(q_1, a) \cup \theta(q_2, a) = \{q_1, q_2\} \cup \emptyset = \{q_1, q_2\} = q_1',$$

$$\theta'(q_1', b) = \theta(q_1, b) \cup \theta(q_2, b) = \emptyset \cup \{q_0\} = \{q_0\} = q_0'.$$

$$\theta'(q_2', a) = \emptyset = q_2', \theta'(q_2', b) = \emptyset = q_2'.$$

Так как новых состояний q' не появилось, то процесс закончен. Получился автомат M' :

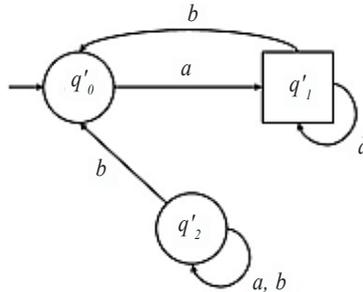


Рис. 12. ДКА M'

Легко видеть, что это ДКА и что $L(M') = L(M)$.

Теорема 2.

Пусть L – язык, допускаемый недетерминированным конечным автоматом $M_N = (Q_N, \Sigma, \theta_N, q_0, F_N)$. Тогда существует детерминированный конечный автомат $M_D = (Q_D, \Sigma, \theta_D, q_0, F_D)$ такой, что $L = L(M_D)$.

Доказательство.

Покажем, как по данному НКА M_N построить ДКА M_D . Будем использовать для этого процедуру (назовем ее Det), описанную ниже, которая по M_N строит диаграмму переходов Γ_D автомата M_D . Заметим, что, по определению, каждая вершина Γ_D должна иметь $|\Sigma|$ выходящих дуг, помеченных различными символами из Σ . Процедура Det останавливается лишь после того, как будут построены все такие дуги.

Процедура Det:

Шаг 1. Строим граф Γ_D , состоящий из единственной вершины $\{q_0\}$, которую считаем начальной.

Шаг 2. Повторяем этот шаг до тех пор, пока у каждой построенной вершины в Γ_D не будет ровно $|\Sigma|$ выходящих дуг:

1. Берем вершину $\{q_p, q_j, \dots, q_k\}$, принадлежащую Γ_D , в которой отсутствует выходящая дуга, помеченная некоторым символом $a \in \Sigma$.

2. Вычисляем $\theta_N(q_p, a)$, $\theta_N(q_j, a)$, \dots , $\theta_N(q_k, a)$.

3. Образует объединение этих множеств:

$$\theta_N(q_p, a) \cup \theta_N(q_j, a) \cup \dots \cup \theta_N(q_k, a) = \{q_p, q_m, \dots, q_n\}.$$

4. Добавляем в Γ_D вершину, помеченную $\{q_p, q_m, \dots, q_n\}$ (если она еще не была помещена в Γ_D).

5. Добавляем в Γ_D дугу из вершины $\{q_p, q_j, \dots, q_k\}$ в вершину $\{q_p, q_m, \dots, q_n\}$ и помеченную символом a .

Шаг 3. Каждая вершина в Γ_D , чья метка содержит некоторую $q_f \in F$, считается финальной и помещается в множество F_D .

Шаг 4. Если M_N допускал строку ε , то вершина $\{q_0\}$ также помещается в F_D .

Ясно, что процедура *Det* сходится (то есть заканчивает свою работу через конечное число шагов). Действительно, каждый виток цикла в *Шаге 2* добавляет дугу в Γ_D . Но Γ_D может иметь самое большее $2^{|\Omega|} \times |\Sigma|$ дуг, так что этот цикл рано или поздно останавливается.

Покажем теперь, что в результате работы процедуры *Det* мы получим требуемый автомат M_D (заданный своей диаграммой переходов Γ_D). Используем индукцию по длине n входной строки. Очевидно, что при $n = 0$ $\varepsilon \in L(M_N)$ тогда и только тогда, когда $\varepsilon \in L(M_D)$. Предположим, что для любой строки $\alpha \in \Sigma^*$, длина которой $|\alpha| \leq n$, существование в Γ_N пути, помеченного строкой α , из вершины q_0 в вершину q_i влечет существование в Γ_D пути, помеченного α , из вершины $\{q_0\}$ в вершину $S_i = \{\dots, q_p, \dots\}$, и наоборот.

Пусть теперь $|\alpha| = n + 1$. Положим $\alpha = \beta a$ и рассмотрим путь в Γ_N из q_0 в q_j , помеченный строкой α . Тогда должен существовать путь, помеченный β , из q_0 в q_i и дуга (или совокупность дуг) из q_i в q_j , помеченная символом a . По предположению индукции, в Γ_D существует путь из $\{q_0\}$ в S_i , помеченный строкой β , а по построению в Γ_D должна существовать дуга, помеченная a , из S_i в некоторую вершину, чья метка содержит q_j . Аналогичным образом, несложно провести и обратное рассуждение. Таким образом, ви-

дим, что наше утверждение верно и для случая $|\alpha| = n + 1$, откуда следует его истинность для любого натурального числа n . Наконец, замечаем, что если $\theta_N^*(q_0, \alpha)$ содержит некоторое финальное состояние q_f , то $q_f \in \theta_D^*(\{q_0\}, \alpha)$ и, значит, $\theta_D^*(\{q_0\}, \alpha) \in F_D$, а если $q_f \in \theta_D^*(\{q_0\}, \alpha)$, то $q_f \in \theta_N^*(q_0, \alpha)$. Следовательно, $L(M_N) = L(M_D)$. **Теорема доказана.**

Таким образом, оба способа задания языков – с помощью конечных автоматов или регулярными выражениями – равносильны.

2.6. Минимизация конечных автоматов

Любой ДКА определяет единственный язык, а именно тот, который он допускает. Обратное не верно: для каждого автоматного языка существует бесконечное множество конечных автоматов, допускающих этот язык. Автоматы из этого множества могут существенно отличаться друг от друга количеством состояний, оставаясь эквивалентными. Для практических целей предпочтение обычно отдают тем из них, которые содержат наименьшее число состояний. Существует целый ряд методов нахождения таких автоматов. Они осуществляют так называемую *минимизацию конечных автоматов*, т. е. в классе эквивалентных автоматов находят автомат с минимальным числом состояний.

Пример 14.

Нетрудно проверить, что представленные на рис. 13 автоматы (а) и (б) эквивалентны. В этом можно убедиться с помощью индукции, непосредственно выписывая строки, допускаемые тем и другим автоматом.

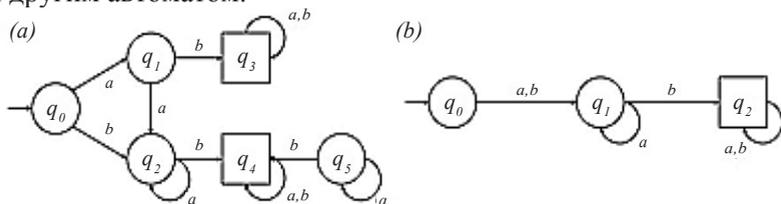


Рис. 13. Эквивалентные ДКА с разным числом состояний

Заметим, что состояние q_5 не играет никакой роли в распознавании строк, так как никогда не может быть достигнуто

из начального состояния q_0 . Подобные состояния, называемые недостижимыми, заведомо могут быть удалены из диаграммы ДКА вместе с входящими в них и выходящими из них дугами без ущерба для допускаемого автоматом языка. Но даже после удаления вершины q_5 число состояний может быть сокращено до трех так, как это, например, показано на рис. 13 (б).

Пусть $M = (Q, \Sigma, \theta, q_0, F)$ – ДКА.

Определение 10.

Два состояния p и q детерминированного конечного автомата называются *неразличимыми*, если для любой строки $\alpha \in \Sigma^*$ верно одно из двух:

$$\theta^*(q, \alpha) \in F \text{ и } \theta^*(p, \alpha) \in F$$

или

$$\theta^*(q, \alpha) \notin F \text{ и } \theta^*(p, \alpha) \notin F.$$

Состояния p и q называются *различимыми*, если существует строка $\alpha \in \Sigma^*$ такая, что одно из состояний

$$\theta^*(q, \alpha), \theta^*(p, \alpha)$$

принадлежит множеству F , а другое – нет.

Ясно, что любые два состояния автомата M либо различимы, либо неразличимы. Отношение неразличимости состояний является эквивалентностью на множестве Q , так как оно рефлексивно, симметрично и транзитивно.

Метод редукции (т. е. уменьшения) числа состояний ДКА M , который мы собираемся рассмотреть, основан на нахождении множеств неразличимых состояний автомата M . Эти множества представляют собой классы эквивалентности по отношению неразличимости, которые образуют разбиение множества Q на непересекающиеся подмножества. Эти классы эквивалентности и будут состояниями нового редуцированного автомата.

Вначале дадим описание алгоритма нахождения пар различимых состояний автомата M .

Процедура DIS:

Шаг 1. Удаляем все недостижимые состояния автомата. Эти состояния определяются простым просмотром всех путей (без циклов), ведущих из начальной вершины диаграммы пере-

ходов. Любая вершина, не принадлежащая ни одному из этих путей, является недостижимой.

Шаг 2. Рассмотрим все пары состояний (p, q) . Если $p \in F$ и $q \notin F$ или наоборот, то пара (p, q) помещается во множество D .

Шаг 3. Повторяем следующий цикл до тех пор, пока не перестанут добавляться в D новые пары:

- для всех пар (p, q) и всех $a \in \Sigma$ вычисляем $\theta(p, a) = p_a$ и $\theta(q, a) = q_a$;
- если пара (p_a, q_a) была уже помещена в D , то пару (p, q) тоже помещаем в D .

Теорема 3.

Процедура *DIS* сходится и помещает в D все различимые состояния автомата M .

Доказательство.

Очевидно, алгоритм сходится, так как множество всех пар, которые надо просмотреть, конечно, а на каждом витке цикла *Шага 3* в множество D помещается хотя бы одна пара. Нетрудно заметить также, что пары, помещенные в множество D , различимы. Остается показать, что процедура *DIS* помещает в D все различимые пары (p, q) .

Будем говорить, что состояния q_i и p_j различимы строкой длины n ($n > 0$), если существуют переходы

$$\theta(q_i, a) = q_k(1)$$

и

$$\theta(p_j, a) = q_l(2)$$

для некоторого $a \in \Sigma$, где q_k и q_l – состояния, различимые строкой длины $n - 1$. Для $n = 0$ состояния p и q различимы строкой длины 0, если одно из них принадлежит F , а другое – нет.

Покажем, что после n витков цикла на *Шаге 3* все пары состояний, различимые строкой длины n или **меньшей**, будут помещены во множество D . *Шаг 2* служит базой индукции: на этом шаге все пары, различимые пустой строкой, помещаются в D .

Допустим, что наше утверждение верно для всех длин $k = 0, 1, \dots, n - 1$. Тогда на n -м витке цикла все пары, различимые строками, длины которых меньше n , будут уже содержаться в D .

Тогда, учитывая соотношения (1) и (2), замечаем, что все пары, различимые строками длины n , будут помещены в D . По индукции заключаем, что после завершения n -го витка цикла все пары (p, q) , различимые строками длины n или меньшей, будут в D .

Покажем, что после завершения работы алгоритма не останется ни одной различимой пары, не помещенной в D . Предположим, что алгоритм *DIS* остановился после $(n + 1)$ -го витка цикла *Шага 3*. Как мы показали, после **n -го витка цикла все пары, различимые строками длины n или меньшей, будут уже содержаться в D** . Раз цикл остановился на $(n + 1)$ -м витке, это означает, что на этом витке не было добавлено в D ни одной различимой пары. Следовательно, не существует пары, различимой строкой длины $n + 1$. Но тогда из соотношений (1) и (2) получаем, что не может быть пар, различимых строкой длины $n + 2, n + 3$ и так далее. Значит, после завершения работы цикла все различимые пары окажутся в D . ***Теорема доказана.***

Используя множество D , разобьем множество Q на непересекающиеся подмножества неразличимых состояний

$$\{q_p, q_p, \dots, q_k\}, \{q_p, q_m, \dots, q_n\}, \dots$$

так, чтобы каждое $q \in Q$ входило ровно в одно подмножество и чтобы любые два состояния из различных подмножеств были различимы. Так как свойство неразличимости является отношением эквивалентности на множестве Q , то такое разбиение существует. Возьмем построенные подмножества в качестве состояний нового автомата, построим этот автомат и покажем, что он минимальный. Построение минимального автомата производится процедурой *MIN*, которая использует процедуру *DIS*.

Процедура MIN:

По данному ДКА $M = (Q, \Sigma, \theta, q_0, F)$ строим ДКА $M' = (Q', \Sigma, \theta', q_0', F')$ следующим образом:

Шаг 1. Используя процедуру *DIS*, находим все пары различимых и, соответственно, все пары неразличимых состояний. По ним строим подмножества

$$\{q_p, q_p, \dots, q_k\}, \{q_p, q_m, \dots, q_n\}, \dots$$

как это было описано выше.

Шаг 2. Для каждого такого множества

$$\{q_i, q_j, \dots, q_k\}$$

неразличимых состояний образуем состояние автомата M' , обозначаемое меткой

$$\langle ij \dots k \rangle.$$

Шаг 3. Для каждого перехода ДКА M вида

$$\theta(q_r, a) = q_s$$

находим соответствующие подмножества, которым принадлежат q_r и q_s .

Если $q_r \in \{q_i, q_j, \dots, q_k\}$ и $q_s \in \{q_l, q_m, \dots, q_n\}$, то добавляем в Q' соотношение:

$$\theta(\langle ij \dots k \rangle, a) = \langle lm \dots n \rangle.$$

Шаг 4. Начальным состоянием q_0' объявляем такое состояние M' , метка которого содержит 0.

Шаг 5. В F' помещаем каждое состояние, метка которого $\langle \dots, i, \dots \rangle$ содержит i для некоторого $q_i \in F$.

Теорема 4.

Пусть M – произвольный ДКА, тогда применение к нему процедуры MIN дает ДКА M' такой, что

$$L(M) = L(M'),$$

причем автомат M' минимален, т. е. не существует другого ДКА с меньшим числом состояний, эквивалентного автомату M .

Доказательство.

Пусть M – произвольный детерминированный конечный автомат. Применим к нему процедуру MIN . В результате получаем некоторый ДКА, который обозначим M' . Доказать эквивалентность M и M' можно по индукции аналогично тому, как это было сделано для установления эквивалентности ДКА и НКА. Действительно, достаточно показать, что для любых $q_i \in Q$ и $\alpha \in \Sigma^* \theta^*(q_i, \alpha) = q_j$ тогда и только тогда, когда $\theta'^*(\langle \dots i \dots \rangle, \alpha)$ имеет метку вида $\langle \dots j \dots \rangle$. Это легкое упражнение мы оставляем читателю.

Сложнее показать, что M' минимален.

Пусть ДКА M' имеет состояния $\{p_0, p_1, p_2, \dots, p_m\}$, где p_0 – начальное состояние. Допустим, что существует другой ДКА M_1 ,

эквивалентный M' , но имеющий меньше состояний $\{r_0, r_1, \dots, r_s\}$, где $s < m$, а r_0 – начальное состояние M_1 , и пусть θ_1 – функция переходов автомата M_1 . Так как в автомате M' нет недостижимых состояний, то все состояния p_1, p_2, \dots, p_m достижимы из p_0 , т. е. существуют строки $\alpha_1, \alpha_2, \dots, \alpha_m$ такие, что

$$\theta^*(p_0, \alpha_i) = p_i, \quad i = 1, 2, \dots, m.$$

Но в силу того, что M_1 имеет меньше состояний, нежели M' , должны существовать, по меньшей мере, две такие различные строчки α_k и α_l , $1 \leq k \leq m$, $1 \leq l \leq m$, что

$$\theta_1^*(r_0, \alpha_k) = \theta_1^*(r_0, \alpha_l).$$

Так как p_k и p_l различимы, то должна существовать некоторая строка β такая, что $\theta^*(p_0, \alpha_k\beta) = \theta^*(p_k, \beta)$ является заключительным состоянием, а

$$\theta^*(p_0, \alpha_l\beta) = \theta^*(p_l, \beta)$$

не является заключительным состоянием (или наоборот). Это означает, что строка $\alpha_k\beta$ допускается автоматом M' , а $\alpha_l\beta$ – нет. Но в то же время имеем:

$$\begin{aligned} \theta_1^*(r_0, \alpha_k\beta) &= \theta_1^*(\theta_1^*(r_0, \alpha_k), \beta) = \\ &= \theta_1^*(\theta_1^*(r_0, \alpha_l), \beta) = \theta_1^*(r_0, \alpha_l\beta). \end{aligned}$$

А это означает, что строки $\alpha_k\beta$ и $\alpha_l\beta$ одновременно или допускаются автоматом M_1 , или отвергаются, что противоречит эквивалентности M' и M_1 . Следовательно, M_1 не существует. **Теорема доказана.**

Глава 3. Регулярные выражения и регулярные грамматики

3.1. Регулярные выражения

Одним из способов описания языков с использованием алгебраических конструкций является задание языков регулярными выражениями. Эти конструкции включают в себя строки символов в некотором фиксированном алфавите Σ , скобки и символы операций $+$, \cdot и $*$. Простейшим случаем является обозначение языка $\{a\}$ регулярным выражением a . Чуть сложнее выглядит регулярное выражение, обозначающее $\{a, b, c\}$:

$$a+b+c,$$

где символ $+$ используется для операции объединения множеств. Аналогичным образом будем использовать символы \cdot и $*$ для обозначения операций сцепления и итерации. Так, например, выражение $(a + b \cdot c)^*$ обозначает язык $(\{a\} \cup \{b\} \cdot \{c\})^* = \{\epsilon, a, bc, aa, abc, \dots\}$. Заметим, что здесь использована следующая иерархия операций: предполагается, что символ \cdot связывает операнды сильнее, чем $+$, а символ $*$ – сильнее, чем оба предыдущие.

Дадим строгое определение регулярных выражений. Это определение имеет рекурсивный характер, как и большинство подобных определений в алгебре и логике.

Начнем с определения одного класса множеств, называемых регулярными множествами. Это те множества, которые могут быть получены из простейших множеств строк в фиксированном алфавите Σ с помощью операций объединения языков, их сцепления и итерации.

Определение 11.

Пусть Σ – конечный алфавит. Регулярное множество в алфавите Σ определяется рекурсивно следующим образом:

- пустое множество \emptyset – регулярное;
- множество $\{\epsilon\}$ – регулярное;
- для любого $a \in \Sigma$ множество $\{a\}$ – регулярное;
- если P и Q – регулярные множества в алфавите Σ , то регулярными являются и множества $P \cup Q, P \cdot Q, P^*$;

– других регулярных множеств в алфавите Σ нет.

Итак, множество в алфавите Σ регулярно тогда и только тогда, когда оно либо \emptyset , либо $\{\varepsilon\}$, либо $\{a\}$, где $a \in \Sigma$, либо получено из этих множеств применением конечного числа операций объединения, сцепления и итерации.

Определение 12.

Регулярные выражения в алфавите Σ и обозначаемые ими регулярные множества в том же алфавите определяются рекурсивно следующим образом:

– $\mathbf{0}$ – регулярное выражение, обозначающее регулярное множество \emptyset ;

– ε – регулярное выражение, обозначающее регулярное множество $\{\varepsilon\}$;

– если $a \in \Sigma$, то \mathbf{a} – регулярное выражение, обозначающее регулярное множество $\{a\}$;

– если \mathbf{p} и \mathbf{q} – регулярные выражения, обозначающие регулярные множества $L(\mathbf{p})$ и $L(\mathbf{q})$, то $(\mathbf{p} + \mathbf{q})$ – регулярное выражение, обозначающее регулярное множество $L(\mathbf{p}) \cup L(\mathbf{q})$; $(\mathbf{p} \bullet \mathbf{q})$ – регулярное выражение, обозначающее множество $L(\mathbf{p}) \bullet L(\mathbf{q})$; $(\mathbf{p})^*$ – регулярное выражение, обозначающее множество $(L(\mathbf{p}))^*$;

– других регулярных выражений в алфавите Σ нет.

Учитывая наше соглашение относительно приоритетов операций $+$, \bullet и $*$, мы будем избегать употребления избыточных скобок в регулярных выражениях. Например, запись $\mathbf{a} + \mathbf{b} \bullet \mathbf{a}^*$ означает выражение

$$(\mathbf{a} + (\mathbf{b} \bullet (\mathbf{a}^*))).$$

Пример 15.

Для $\Sigma = \{a, b, c\}$ строка $(\mathbf{a} + \mathbf{b} \bullet \mathbf{c})^* \bullet (\mathbf{c} + \mathbf{0})$ является регулярным выражением, обозначающим множество $\{a, bc\}^* \bullet \{c\}$.

Пример 16.

Найти множество $L(\mathbf{a}^* \bullet (\mathbf{a} + \mathbf{b}))$.

По определению 12 имеем: $L(\mathbf{a}^* \bullet (\mathbf{a} + \mathbf{b})) = L(\mathbf{a}^*) \bullet L(\mathbf{a} + \mathbf{b}) = (L(\mathbf{a}^*)) \bullet (L(\mathbf{a}) \cup L(\mathbf{b})) = \{\varepsilon, a, aa, \dots\} \bullet \{a, b\} = \{a, aa, aaa, \dots, b, ab, aab, \dots\}$.

Пример 17.

Выражение $p = (a \cdot a)^* \cdot (b \cdot b)^* \cdot b$ обозначает множество $L(p) = \{a^{2n}b^{2m+1} \mid n \geq 0, m \geq 0\}$.

Пример 18.

В алфавите $\Sigma = \{a, b\}$ найти регулярное выражение p такое, что $L(p) = \{\alpha \in \Sigma^* \mid \alpha \text{ имеет, как минимум, два соседних символа } a\}$.

В этом случае любая строка $\alpha \in L(p)$ может быть представлена в виде $\alpha = \beta a a \gamma$, где β и γ – произвольные строки из Σ^* . Тогда, очевидно, можно записать:

$$p = (a + b)^* \cdot a \cdot a \cdot (a + b)^*.$$

Определение 13.

Регулярные выражения p и q эквивалентны, если $L(p) = L(q)$, т. е. если они обозначают одно и то же множество. Будем в этом случае писать $p = q$.

Ясно, что для каждого регулярного выражения можно построить регулярное множество, обозначаемое этим выражением. Понятно, что и для каждого регулярного множества можно найти, по крайней мере, одно регулярное выражение, обозначающее это множество. Но таких выражений для одного и того же регулярного множества существует бесконечно много. Действительно, если p – регулярное выражение в алфавите Σ , обозначающее множество $L(p)$, то регулярные выражения $p + 0$, $(p + 0) + 0$, $((p + 0) + 0) + 0$, ... будут обозначать одно и то же множество $L(p)$.

Лемма 5.

Пусть p, q, r – регулярные выражения. Тогда:

- 1) $p + q = q + p$
- 2) $\emptyset^* = \varepsilon$
- 3) $p + (q + r) = (p + q) + r$
- 4) $p \cdot (q \cdot r) = (p \cdot q) \cdot r$
- 5) $p \cdot (q + r) = p \cdot q + p \cdot r$
- 6) $(p + q) \cdot r = p \cdot r + q \cdot r$
- 7) $p \cdot \varepsilon = \varepsilon \cdot p = p$
- 8) $\emptyset \cdot p = p \cdot \emptyset = \emptyset$

$$9) p^* = p + p^*$$

$$10) (p^*)^* = p^*$$

$$11) p + p = p$$

$$12) p + \theta = p.$$

Доказательство.

Пусть p и q – регулярные выражения, которые обозначают множества $L(p)$ и $L(q)$ соответственно. Тогда $p + q$ обозначает $L(p) \cup L(q)$, а $q + p$ обозначает $L(q) \cup L(p)$. Но $L(p) \cup L(q) = L(q) \cup L(p)$ по свойству операции объединения множеств, следовательно, $p + q = q + p$. Доказательства остальных эквивалентностей проводятся по той же схеме и оставляются читателю.

3.2. Связь между регулярными выражениями и автоматными языками

Между регулярными выражениями и, соответственно, регулярными множествами, с одной стороны, и автоматными языками – с другой, существует тесная связь. На самом деле мы покажем, что класс автоматных языков и класс регулярных множеств – это одно и то же.

Вначале мы покажем, что если p – регулярное выражение, то $L(p)$ – автоматный язык, т. е. существует конечный автомат (детерминированный или недетерминированный), который допускает $L(p)$. Построение такого автомата (на самом деле недетерминированного) мы будем осуществлять в соответствии с рекурсивной схемой Определения 12 регулярного выражения.

Теорема 6.

Пусть p – регулярное выражение. Тогда существует недетерминированный конечный автомат $M(p)$, который допускает язык $L(p)$, что означает, что $L(p)$ является автоматным языком.

Доказательство.

Рассмотрим элементарные выражения θ , ε и a для $a \in \Sigma$. Соответствующие автоматы, распознающие обозначенные этими выражениями языки, представлены на рис. 14 в пунктах а), б) и с):

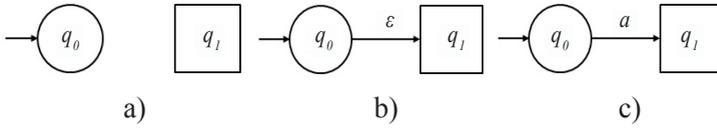


Рис. 14. НКА для элементарных выражений

Пусть теперь p и q – произвольные регулярные выражения, а $M(p)$ и $M(q)$ – конечные автоматы, допускающие множества, обозначаемые этими выражениями. Договоримся схематично изображать автомат $M(r)$, допускающий множество, обозначаемое регулярным выражением r , так, как это представлено на рис. 15.

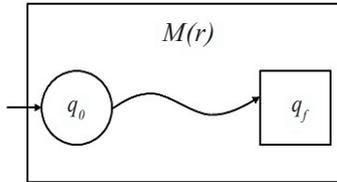


Рис. 15. Схематичное представление НКА, допускающего $L(r)$

На этой схеме левая вершина-кружок представляет собой начальное состояние в диаграмме переходов автомата $M(r)$, а правая вершина-квадрат – его заключительное состояние. Так как для каждого конечного автомата существует эквивалентный ему конечный автомат с одним заключительным состоянием, то данное представление не уменьшает общность рассуждений. Пользуясь таким схематическим представлением для НКА $M(p)$ и $M(q)$, нетрудно построить диаграммы переходов НКА для множеств $L(p + q) = L(p) \cup L(q)$ (см. рис. 16), для $L(p \cdot q) = L(p) \cdot L(q)$ (см. рис. 17) и для $L(p^*) = (L(p))^*$ (см. рис. 18).

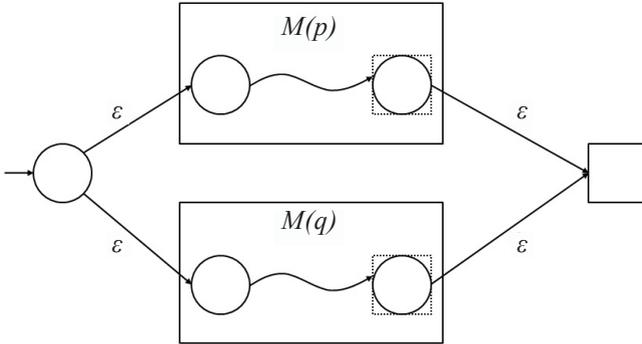


Рис. 16. НКА для $L(p + q)$

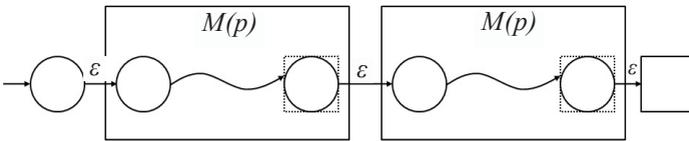


Рис. 17. НКА для $L(p \cdot q)$

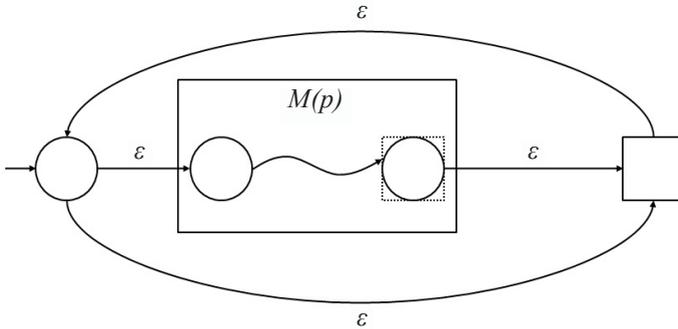


Рис. 18. НКА для $L(p^*)$

Как видно из рисунков, заключительные состояния автоматов $M(p)$ и $M(q)$ перестают быть таковыми при конструировании новых автоматов, имеющих свои собственные заключительные состояния, обозначаемые, как обычно, квадратиками.

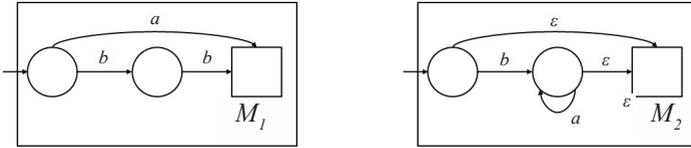
На основании индукции заключаем, что, поступая так систематически, мы можем построить НКА для любого регулярного выражения r , а значит, язык $L(r)$ – автоматный. Теорема доказана.

Пример 19.

Найти НКА, допускающий язык $L(p)$, где

$$p = (a + b \cdot b)^* \cdot (b \cdot a^* + \varepsilon).$$

Сначала строим автоматы M_1 и M_2 для выражений $(a + b \cdot b)$ и $(b \cdot a^* + \varepsilon)$ (см. рис. 19). Затем, соединяя их по теореме 6, получаем требуемый НКА (рис. 20).



а) НКА для $(a + b \cdot b)$

б) НКА для $(b \cdot a^* + \varepsilon)$

Рис. 19. Автоматы M_1 и M_2

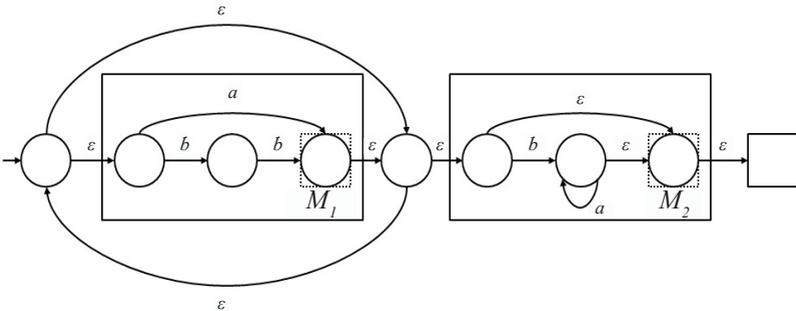


Рис. 20. НКА, допускающий язык $L((a + b \cdot b)^* \cdot (b \cdot a^* + \varepsilon))$

Ожидаемым, но отнюдь не очевидным является обратное утверждение. Оно известно под названием «Теорема Клини».

Теорема 7 (Теорема Клини).

Для любого автоматного языка L существует регулярное выражение p такое, что $L=L(p)$.

Доказательство.

Пусть L – автоматный язык, т. е. существует ДКА

$$M = (Q, \Sigma, \theta, q_0, F)$$

такой, что $L = L(M)$. Если $F = \{q_{f1}, q_{f2}, \dots, q_{ft}\}$, тогда из ДКА M можно получить t автоматов M_1, M_2, \dots, M_t , положив

$$M_i = (Q, \Sigma, \theta, q_0, \{q_{fi}\}), i = 1, \dots, t.$$

Нетрудно видеть, что

$$L = L(M_1) \cup L(M_2) \cup \dots \cup L(M_t).$$

Если мы сможем доказать, что для любого $L(M)$ существует регулярное выражение p_i , которое обозначает это множество, тогда для L искомое регулярное выражение будет следующим:

$$p = p_1 + p_2 + \dots + p_t$$

Таким образом, достаточно рассмотреть случай, когда ДКА M имеет одно-единственное заключительное состояние. Пусть

$$M = (Q, \Sigma, \theta, q_1, \{q_n\}),$$

где $Q = \{q_1, q_2, \dots, q_n\}$. Возьмем диаграмму переходов $D(M)$ автомата M и рассмотрим множество всех путей в $D(M)$, ведущих из вершины q_i в вершину q_j и проходящих через вершины, номера которых не превышают $s, s \leq n$. Если промежуточных вершин нет, то будем считать, что $s = 0$. Соответствующее множество строк, помечающих эти пути, обозначим через $L_{ij}^s, 1 \leq i \leq n, 1 \leq j \leq n, 1 \leq s \leq n$.

Покажем индукцией по s , что для любых i, j, s , удовлетворяющих указанным выше условиям, множеству L_{ij}^s соответствует некоторое регулярное выражение p_{ij}^s , которое обозначает это множество.

Пусть $s = 0$. Возможны два случая.

1) $i \neq j$, т. е. вершины q_i и q_j различны. Если дуга между ними отсутствует, то $L_{ij}^0 = \emptyset$ и $p_{ij}^0 = 0$. Если же дуга (q_i, q_j) существует, она может быть помечена любым символом из $\Sigma = \{a_1, a_2, \dots, a_m\}$, следовательно, множество L_{ij}^0 в этом случае представимо выражением вида

$$p_{ij}^0 = a_{k_1} + a_{k_2} + \dots + a_{k_r} + 0$$

где $a_{k_i} \in \Sigma, 1 \leq k_i \leq m$.

2) $i = j$, т. е. вершины q_i и q_j совпадают. В этом случае дуга представляет собой петлю в вершине q_i , помеченную некоторым символом из Σ . Если же дуга отсутствует, то это можно интерпре-

тировать как ε -переход из вершины q_i в q_j . Таким образом, можем положить здесь

$$p_{ij}^0 = ak_1 + ak_2 + \dots + ak_r + \varepsilon$$

Допустим, что для любого $s \leq k$ наше утверждение относительно множества L_{ij}^s верно. Пусть теперь $s = k + 1$. Тогда множество L_{ij}^{k+1} можно представить следующим образом:

$$L_{ij}^{k+1} = L_{ij}^k \cup L_{ik}^k + 1 \cdot (L_{k+1\ k+1}^k)^* \cdot L_{k+1j}^k.$$

Действительно, множество путей из вершины q_i в вершину q_j в этом случае состоит из всех путей, либо не проходящих через вершину q_{k+1} (это L_{ij}^k), либо проходящих через q_{k+1} (и в этом случае весь путь разбивается на 3 участка: от q_i до q_{k+1} , из q_{k+1} в q_{k+1} и от q_{k+1} до q_j). По предположению индукции существуют регулярные выражения

$$p_{ij}^k, p_{ik+1}^k, p_{k+1\ k+1}^k, p_{k+1j}^k,$$

обозначающие, соответственно, множества L_{ij}^k , L_{ik+1}^k , $L_{k+1\ k+1}^k$ и L_{k+1j}^k . Но тогда выражение

$$p_{ij}^{k+1} = p_{ij}^k + p_{ik+1}^k \cdot (p_{k+1\ k+1}^k)^* \cdot p_{k+1j}^k$$

будет искомым регулярным выражением для множества L_{ij}^{k+1} . По индукции заключаем, что наше утверждение верно, т. е. для любого $s \geq 0$ существует регулярное выражение p_{ij}^s , обозначающее множество L_{ij}^s . Для завершения доказательства теоремы нам теперь достаточно заметить, что

$$L(M) = L_{1n}^n,$$

и, следовательно, для автоматного языка $L(M)$ тоже существует регулярное выражение, которое его обозначает. **Теорема доказана.**

Следствие 8.

Класс автоматных языков совпадает с классом регулярных множеств.

Таким образом, оба способа задания языков – с помощью конечных автоматов или с помощью регулярных выражений – равносильны.

3.3. Регулярные грамматики

Определение 14.

Грамматика $G = (N, T, S, P)$ называется *праволинейной*, если все продукции имеют вид

$$A \rightarrow \alpha B$$

$$A \rightarrow \alpha, \text{ где } A, B \in N, \alpha \in T^*.$$

Аналогично грамматика называется *леволинейной*, если все продукции имеют вид

$$A \rightarrow B\alpha$$

$$A \rightarrow \alpha.$$

Грамматика называется *регулярной*, если она или праволинейная или леволинейная.

Язык назовем *регулярным*, если он порождается некоторой регулярной грамматикой.

Пример 20.

$$G_1 = (\{S\}, \{a, b\}, S, P)$$

$$S \rightarrow abS \mid a$$

Здесь грамматика G_1 – регулярная.

Пример 21.

$$G_2 = (\{S, A, B\}, \{a, b\}, S, P)$$

$$S \rightarrow A$$

$$A \rightarrow aB \mid \varepsilon$$

$$B \rightarrow Ab$$

Грамматика G_2 – линейная, но не регулярная.

Заметим, что при выводе сентенциальных форм в праволинейной грамматике эти формы будут иметь вид $ab \dots cD$. Пусть к этой сентенциальной форме применяется продукция

$$D \rightarrow dE,$$

то есть имеем в G :

$$ab \dots cD \Rightarrow ab \dots cdE.$$

Этому шагу в выводе можно сопоставить такт работы НКА M , который из состояния D , прочитав символ d , переходит в состояние E :

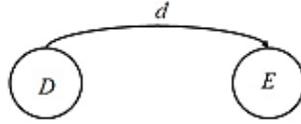


Рис. 21. НКА М

Тогда, очевидно, выводу в G соответствует путь в диаграмме переходов автомата M , и наоборот.

Теорема 9.

Пусть $G = (N, T, S, P)$ – праволинейная грамматика. Тогда $L(G)$ – автоматный язык.

Доказательство.

Положим $N = \{A_0, A_1, \dots\}$, $S = A_0$ и допустим, что продукции имеют вид

$$\begin{aligned} A_0 &\rightarrow \alpha_1 A_i, \\ A_i &\rightarrow \alpha_2 A_j, \\ &\dots \\ A_n &\rightarrow \alpha_r. \end{aligned}$$

Если $\alpha \in L(G)$, тогда, в силу вида продукций G , вывод должен иметь вид:

$$A_0 \Rightarrow \alpha_1 A_i \Rightarrow \alpha_1 \alpha_2 A_j \Rightarrow^* \alpha_1 \alpha_2 \dots \alpha_k A_n \Rightarrow \alpha_1 \alpha_2 \dots \alpha_k \alpha_1 = \alpha. \quad (3)$$

Соответствующий автомат будет воспроизводить вывод, поглощая по очереди каждую α_i . Начальное состояние автомата – A_0 , а для каждой переменной A_i он будет иметь нефинальное состояние, помеченное буквой A_i . Для каждой продукции вида

$$A_i \rightarrow a_1 a_2 \dots a_m A_j$$

автомат будет иметь переходы, связывающие вершины A_i и A_j , то есть функция переходов θ будет определена так, что

$$\theta^*(A_i, a_1 a_2 \dots a_m) = A_j.$$

Для каждой продукции вида

$$A_i \rightarrow a_1 a_2 \dots a_m$$

соответствующий переход

$$\theta^*(A_i, a_1 a_2 \dots a_m) = A_f,$$

где A_f – финальное состояние.

Промежуточные состояния могут быть любыми. Целиком автомат состоит из таких отдельных частей, то есть θ определена указанными соотношениями.

Допустим, что $\alpha \in L(G)$ и (3) имеет место. Тогда, по построению НКА, в диаграмме переходов есть путь из A_0 в A_i , помеченный α_1 , путь из A_i в A_j , помеченный α_2 и т. д., откуда ясно, что

$$A_f \in \theta^*(A_0, \alpha),$$

то есть α допускается автоматом M .

Обратно, пусть α допустима M . В силу построения M , допустить α – это пройти через последовательность состояний A_0, A_i, \dots до A_f , используя пути, помеченные $\alpha_1, \alpha_2, \dots$. Таким образом, α должна иметь вид

$$\alpha = \alpha_1 \alpha_2 \dots \alpha_k \alpha_1,$$

следовательно, возможен вывод

$$A_0 \Rightarrow \alpha_1 A_i \Rightarrow \alpha_1 \alpha_2 A_j \Rightarrow^* \alpha_1 \alpha_2 \dots \alpha_k A_n \Rightarrow \alpha_1 \alpha_2 \dots \alpha_k \alpha_1,$$

где $\alpha_1 \alpha_2 \dots \alpha_k \alpha_1 \in L(G)$. Теорема доказана.

Пример 22.

Построить автомат, допускающий язык, порожденный грамматикой $A0 \rightarrow aA1, A1 \rightarrow abA0 \mid b$.

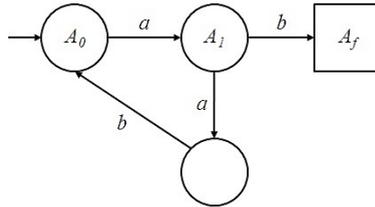


Рис. 22. Автомат, допускающий язык из примера 22

Теорема 10.

Если L – автоматный язык в алфавите Σ , тогда существует праволинейная грамматика $G = (N, T, S, P)$ такая, что

$$L = L(G).$$

Доказательство.

Пусть M – ДКА, допускающий L ,

$M = (Q, \Sigma, \theta, q_0, F)$.

Предположим, $Q = \{q_0, q_1, \dots, q_n\}$, $\Sigma = \{a_1, a_2, \dots, a_m\}$. Построим праволинейную грамматику $G = (N, \Sigma, S, P)$, где $N = \{q_0, q_1, \dots, q_n\}$, $S = q_0$.

Для каждого перехода в ДКА M

$$\theta(q_i, a_j) = q_k$$

мы помещаем в P продукцию $q_i \rightarrow a_j q_k$. В дополнение к этому, если $q_k \in F$, то мы добавляем в P еще продукцию $q_k \rightarrow \varepsilon$.

Покажем вначале, что так определенная грамматика G порождает любую строку из L . Пусть $\alpha \in L$ и $\alpha = a_i a_j \dots a_k a_l$. Для M принять α означает совершить переходы:

$$\theta(q_0, a_i) = q_p$$

$$\theta(q_p, a_j) = q_r$$

...

$$\theta(q_s, a_k) = q_t$$

$$\theta(q_t, a_l) = q_f, q_f \in F.$$

По построению, грамматика G будет иметь одну продукцию (как минимум) для каждого из этих соотношений. Поэтому можно построить в G вывод:

$$\begin{aligned} q_0 &\Rightarrow a_i q_p \Rightarrow a_i a_j q_r \Rightarrow \dots a_i a_j \dots a_k a_l \Rightarrow a_i a_j \dots a_k a_l q_f \Rightarrow \\ &\Rightarrow a_i a_j \dots a_k a_l \end{aligned} (**)$$

следовательно, $\alpha \in L(G)$.

Обратно, если $\alpha \in L(G)$, то ее вывод имеет вид (**), а это влечет соотношение $\theta^*(q_0, a_i a_j \dots a_k a_l) = q_f$, то есть $\alpha \in L(M)$. Теорема доказана.

Пример 23.

Построить праволинейную грамматику для $L(\mathbf{a \cdot a \cdot b^* \cdot a})$.

По выражению $\mathbf{a \cdot a \cdot b^* \cdot a}$ построим соответствующий НКА M :

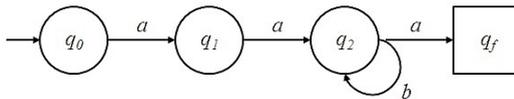


Рис. 23. НКА, соответствующий выражению $\mathbf{a \cdot a \cdot b^* \cdot a}$

В данном случае НКА совпадает с ДКА. Как показано выше, имеем соответствие между M и искомой грамматикой G :

M :

$$\theta(q_0, a) = q_1;$$

$$\theta(q_1, a) = q_2;$$

$$\theta(q_2, b) = q_2;$$

$$\theta(q_2, a) = q_f;$$

$$q_f \in F;$$

G :

$$q_0 \rightarrow aq_1$$

$$q_1 \rightarrow aq_2$$

$$q_2 \rightarrow bq_2$$

$$q_2 \rightarrow aq_f$$

$$q_f \rightarrow \varepsilon.$$

Тогда, к примеру, строка $aaba$ будет иметь вывод в G :

$$q_0 \Rightarrow aq_1 \Rightarrow aaq_2 \Rightarrow aabq_2 \Rightarrow aabaq_f \Rightarrow aaba.$$

Аналогично нетрудно доказать следующую теорему.

Теорема 11.

Язык L автоматный тогда и только тогда, когда существует левосторонняя грамматика G такая, что

$$L = L(G).$$

Объединяя, получаем:

Теорема 12.

Язык L является автоматным тогда и только тогда, когда существует регулярная грамматика G такая, что $L = L(G)$.

Таким образом, любой автоматный язык является регулярным и наоборот.

Глава 4. Свойства регулярных языков

Итак, мы установили совпадение трех классов: автоматных языков, регулярных множеств и регулярных языков, а это означает эквивалентность всех трех определений и позволяет использовать то или иное определение класса регулярных языков в зависимости от конкретной задачи. Но этот результат дает повод задать вопрос: существуют ли другие классы языков, не совпадающие с регулярными? И нельзя ли из регулярных языков с помощью обычных теоретико-множественных операций и некоторых других (таких, как сцепление, итерация) получить языки, не являющиеся регулярными? Последний вопрос относится к свойствам алгебраической замкнутости класса регулярных языков.

Следующий вопрос связан с существованием алгоритмов, позволяющих определять те или иные свойства языков, например, конечен язык или бесконечен. Мы убедимся, что в классе регулярных языков для целого ряда важных случаев удастся найти положительный ответ на вопрос о существовании алгоритма.

И наконец, последний вопрос: можно ли по данному языку определить, регулярен он или нет? Если язык регулярен, можно попытаться построить для него ДКА, регулярное выражение или регулярную грамматику. Но если язык не является регулярным, то требуется иной способ для ответа на вопрос. В частности, если известно некоторое свойство, которым в обязательном порядке обладают все регулярные языки, то можно попытаться показать, что данный язык не обладает этим свойством, и тогда этот язык, очевидно, не будет регулярным.

4.1 Замкнутость класса регулярных языков

Теорема 13.

Класс регулярных языков замкнут относительно операции объединения, пересечения, дополнения, сцепления и итерации, то есть если L_1 и L_2 – регулярные языки, то языки

$$L_1 \cup L_2, L_1 \cap L_2, \bar{L}_1, L_1 \cdot L_2, L_1^*$$

также будут регулярными.

Доказательство.

Пусть L_1 и L_2 – регулярные языки, тогда существуют регулярные выражения p_1 и p_2 такие, что $L_1 = L(p_1)$ и $L_2 = L(p_2)$. По определению,

$$p_1 + p_2, p_1 \cdot p_2 \text{ и } p_1^*$$

– регулярные выражения, обозначающие языки $L_1 \cup L_2$, $L_1 \cdot L_2$, L_1^* соответственно. Следовательно, эти языки тоже регулярны, то есть класс регулярных языков замкнут относительно объединения, конкатенации и итерации.

Для доказательства замкнутости относительно дополнения рассмотрим ДКА $M = (Q, \Sigma, \theta, q_0, F)$, который допускает язык L_1 . Тогда нетрудно видеть, что ДКА

$$\bar{M} = (Q, \Sigma, \theta, q_0, Q \setminus F)$$

допускает именно язык \bar{L}_1 , являющийся дополнением к L_1 в множестве Σ^* (то есть $\bar{L}_1 = \Sigma^* \setminus L_1$). Действительно, заметим, что в определении ДКА мы предполагали, что θ^* является всюду определенной функцией, то есть для любой строки $\alpha \in \Sigma^*$ $\theta^*(q_0, \alpha)$ определено. Следовательно, либо $\theta^*(q_0, \alpha) \in F$, то есть является заключительным состоянием автомата M и в этом случае $\alpha \in L_1$, или же $\theta^*(q_0, \alpha) \in \bar{F} = Q \setminus F$ и $\alpha \in \bar{L}_1$, откуда и получаем, что

$$\bar{L}_1 = L(\bar{M}).$$

Наконец, для доказательства замкнутости относительно пересечения воспользуемся известным соотношением:

$$L_1 \cap L_2 = \overline{\bar{L}_1 \cup \bar{L}_2}.$$

Так как \bar{L}_1 и \bar{L}_2 регулярны, то их объединение тоже регулярно, а следовательно, и его дополнение – регулярный язык. Теорема доказана.

Следствие 14.

Класс регулярных языков замкнут относительно операции разности двух множеств.

Действительно, это немедленно получается из равенства

$$L_1 \setminus L_2 = L_1 \cap \bar{L}_2$$

и предыдущей теоремы.

Определение 15.

Пусть Σ и Γ – алфавиты, тогда функция

$$h : \Sigma \rightarrow \Gamma^*$$

называется *гомоморфизмом*. Иными словами, гомоморфизм – это подстановка, в которой каждый символ одного алфавита заменяется некоторой строкой в другом алфавите.

Область определения функции h может быть расширена на множество строк Σ^* очевидным образом: если $\alpha = a_1 a_2 \dots a_n$ и $\alpha \in \Sigma^*$, то $h(\alpha) = h(a_1)h(a_2)\dots h(a_n)$.

Если L – язык в алфавите Σ , то его гомоморфный образ по отношению к функции h определяется так:

$$h(L) = \{h(\alpha) \mid \alpha \in L\}.$$

Пусть p – регулярное выражение для языка L , тогда регулярное выражение для языка $h(L)$ может быть получено простой заменой в p каждого символа на его гомоморфный образ.

Пример 24.

Пусть $\Sigma = \{0,1\}$ и $\Gamma = \{a, b, c\}$. Определим отображение h так:

$$h(0) = ab,$$

$$h(1) = bbc.$$

Тогда $h(010) = abbbcab$. Соответственно, гомоморфный образ языка $L = \{00,010\}$ – это язык

$$h(L) = \{abab, abbbcab\}.$$

Пример 25.

Возьмем $\Sigma = \{a, b\}$, $\Gamma = \{b, c, d\}$. Положим

$$h(a) = dbcc,$$

$$h(b) = bdc.$$

Предположим, что язык L обозначается выражением

$$p = (a + b^*) \cdot (a \cdot a)^*.$$

Тогда выражение

$p_1 = (d \cdot b \cdot c \cdot c + (b \cdot d \cdot c)^*) \cdot (d \cdot b \cdot c \cdot c \cdot d \cdot b \cdot c \cdot c)^*$
будет обозначать язык $h(L)$.

Теорема 15.

Пусть L – регулярный язык, а h – гомоморфизм, тогда $h(L)$ – регулярный язык. Следовательно, класс регулярных языков замкнут относительно гомоморфизмов.

Доказательство.

Пусть L – регулярный язык, обозначаемый некоторым регулярным выражением p . Находим $h(p)$, подставляя в p вместо каждого символа $a \in \Sigma$ его образ $h(a)$. Нетрудно видеть из определения регулярного выражения, что $h(p)$ – тоже регулярное выражение. Кроме того, легко заметить, что это выражение $h(p)$ обозначает именно язык $h(L)$. Действительно, достаточно показать, что если строка $\alpha \in L(p)$, то образ этой строки $h(\alpha)$ принадлежит $L(h(p))$, и наоборот, для каждой строки $\beta \in L(h(p))$ существует $\alpha \in L(p)$ такая, что $\beta = h(\alpha)$. Оставляя детали этого рассуждения читателю в качестве упражнения, мы завершаем тем самым доказательство. ***Теорема доказана.***

4.2. Алгоритмические проблемы регулярных языков

Одной из основных проблем, относящихся к разряду алгоритмических, является так называемая проблема вхождения (или проблема принадлежности): можно ли по данному языку L и данной строке α определить, входит строка α в L или нет? Под методом, с помощью которого предполагается находить ответ, понимается алгоритм, решающий проблему вхождения. Если для какого-то класса языков такого алгоритма не существует, то языки этого класса не имеют сколько-нибудь существенных приложений. Помимо проблемы вхождения, имеются и другие проблемы относительно языков, где речь идет о существовании некоторого алгоритма. Все такие проблемы будем называть алгоритмическими. Необходимо, однако, уточнить, что мы понимаем под выражением «задан язык». Дело в том, что существует много способов описания одного и того же семейства языков. Так, для регулярных языков имеется возможность неформального описания языка, представление его в теоретико-

множественной нотации, задание с помощью конечного автомата регулярного выражения и регулярной грамматики. Так как для конструирования алгоритмов требуется однозначность исходных данных, то мы остановимся на трех последних способах. Будем называть стандартным представлением регулярного языка задание его или конечным автоматом, или регулярным выражением, или регулярной грамматикой.

Теорема 16.

Пусть фиксирован алфавит Σ , тогда существует алгоритм, который для любого регулярного языка $L \subseteq \Sigma^*$ в стандартном представлении и любой строки $\alpha \in \Sigma^*$ определяет, входит α в L или нет.

Доказательство.

Вначале переходим к представлению языка L с помощью конечного автомата (если L был задан в иной форме, то есть регулярным выражением или регулярной грамматикой). Это можно сделать эффективно за конечное число шагов. Затем строим диаграмму переходов автомата (также за конечное число шагов) и, перебирая все пути, ведущие из начальной вершины и имеющие длину $|\alpha|$, определяем, есть ли среди них путь, помеченный α и оканчивающийся в одной из заключительных вершин (это тоже осуществимо за конечное число шагов). Если такой путь найдется, то делаем вывод, что $\alpha \in L$, а если нет, то $\alpha \notin L$.

Теорема доказана.

Описанный алгоритм универсален в том смысле, что он дает ответ на вопрос « $\alpha \in L?$ » для любой пары (L, α) . Это отражает массовый характер алгоритмической проблемы, что является чрезвычайно важным обстоятельством. Именно оно придает значение любому алгоритму, так как освобождает от необходимости в каждом частном случае изобретать каждый раз новый метод решения проблемы.

Другими важными алгоритмическими проблемами являются проблемы пустоты языков, равенства двух языков и вопрос о том, конечен или бесконечен данный язык. Для регулярных языков все эти проблемы разрешимы.

Теорема 17.

Существуют алгоритмы, которые для любого регулярного языка, заданного стандартным представлением, определяют, является ли он пустым, конечным или бесконечным.

Доказательство.

Как и в предыдущей теореме, строим диаграмму переходов ДКА, допускающего данный язык. Если существует путь без циклов из начальной вершины в одну из финальных вершин, то язык непустой, в противном случае – язык пустой. Для определения, является язык конечным или нет, выписываем все вершины, которые являются основаниями циклов, и проверяем, находится ли хоть одна из них на пути от начальной вершины к какой-нибудь финальной. Если такой путь существует, то язык бесконечен; в противном случае язык конечен. Для завершения доказательства достаточно заметить, что такая проверка осуществляется за конечное число шагов, то есть алгоритм всегда сходится.

Теорема доказана.

Теорема 18.

Существует алгоритм, который по стандартному представлению любых двух регулярных языков L_1 и L_2 определяет, равны ли они.

Доказательство.

Пусть даны два регулярных языка L_1 и L_2 . Построим новый язык

$$L_3 = (L_1 \cap \bar{L}_1) \cup (\bar{L}_1 \cap L_2).$$

Очевидно, L_3 – тоже регулярный язык. Кроме того, нетрудно заметить, что $L_1 = L_2$ тогда и только тогда, когда $L_3 = \emptyset$. Но по теореме 18 существует алгоритм, который для любого языка выясняет, пуст ли он. Применив его в данном случае к языку L_3 , получаем ответ на вопрос, равны ли L_1 и L_2 . ***Теорема доказана.***

В заключение заметим, что не для всех семейств языков удается столь же легко находить решение алгоритмических проблем, как для класса регулярных языков, а в ряде случаев таких решений не существует в принципе, а соответствующие проблемы называются алгоритмически неразрешимыми.

4.3. Лемма о расширении регулярных языков

Следующий результат, известный как Лемма о расширении регулярных языков (Pumping Lemma), основан на том простом замечании, что в диаграмме переходов с n вершинами любой путь длины, большей или равной n , должен содержать цикл, то есть повторяющуюся вершину.

Теорема 19.

Пусть L – бесконечный регулярный язык. Тогда существует целое число $m > 0$ такое, что для любой строки $\alpha \in L$, длина которой $|\alpha| \geq m$, возможно представление

$$\alpha = \beta\gamma\delta, \text{ где } |\gamma| \neq 0 \text{ и } |\beta\gamma| \leq m,$$

такое, что для любого $i = 0, 1, \dots$ строка

$$\alpha_i = \beta\gamma^i\delta$$

также принадлежит L .

Доказательство.

Если язык L регулярный, тогда существует ДКА M , который распознает его. Пусть

$$q_0, q_1, \dots, q_n$$

– множество всех состояний автомата M . Возьмем любую строку $\alpha \in L$ такую, что

$$|\alpha| = k \geq m = n + 1.$$

Так как по предположению L бесконечен, то такая достаточно длинная строка всегда найдется. Рассмотрим все те состояния автомата M , через которые он проходит, когда прочитывает строку α ; пусть это будут состояния

$$q_0, q_r, q_p, \dots, q_f$$

Так как эта последовательность имеет $k + 1$ вершину и

$$k + 1 > n + 1,$$

то, как минимум, одна вершина должна повториться, по крайней мере, на n -м шаге. Таким образом, последовательность должна иметь следующий вид:

$$q_0, q_r, q_p, \dots, q_p, \dots, q_r, \dots, q_f$$

Причем, очевидно, можно считать, что последовательность $q_0, q_p, q_p, \dots, q_p, \dots, q_r$ уже не содержит других повторяющихся элементов, кроме последнего q_r .

Но тогда должны существовать подстроки β, γ, δ строки α такие, что

$$\theta^*(q_0, \beta) = q_r,$$

$$\theta^*(q_r, \gamma) = q_r,$$

$$\theta^*(q_r, \delta) = q_r,$$

причем $|\beta\gamma| \leq n + 1 = m$ и $|\gamma| \geq 1$. Отсюда непосредственно получаем, что

$$\theta^*(q_0, \beta\delta) = q_r,$$

а также

$$\theta^*(q_0, \beta\gamma 2\delta) = q_r,$$

$$\theta^*(q_0, \beta\gamma 3\delta) = q_r$$

и так далее, что и завершает доказательство. **Теорема доказана.**

Заметим, что Лемма о расширении применима только к бесконечным языкам, так как любая достаточно длинная строка языка сразу порождает бесконечное множество строк этого же языка.

Впрочем, это не является недостатком, так как все конечные языки заведомо регулярны, а Лемма о расширении главным образом применяется для доказательства того, что какой-то данный язык не является регулярным. Это доказательство всегда проводится по принципу «от противного», чтобы затем получить противоречие. Но, как и любое другое *необходимое* условие, Лемма о расширении не может использоваться для доказательства регулярности какого-то языка.

Пример 26.

Показать, используя Лемму о расширении, что язык

$$L = \{a_n b_n \mid n \geq 0\}$$

не является регулярным.

Допустим противное, то есть что язык L – регулярный. Тогда для L справедлива Лемма о расширении, а значит, и должно существовать такое m , что если $|\alpha| = n > m$, где $\alpha \in L$, то α можно представить в виде $\beta\gamma\delta$.

Для строки γ возможны три случая:

- 1) $\gamma = a^k$,
- 2) $\gamma = b^t$
- 3) $\gamma = a^k b^t$.

В каждом из этих случаев мы получаем:

- 1) $\alpha_i = a^{n-k} (a^k)^i b^n$,
- 2) $\alpha_i = a^n (b^t)^i b^{n-t}$,
- 3) $\alpha_i = a^{n-k} (a^k b^t)^i b^{n-t}$.

Полагая $i = 2$, убеждаемся, что ни в одном из случаев α_2 не принадлежит L , что противоречит утверждению Леммы о расширении. Следовательно, наше первоначальное предположение о регулярности языка L неверно, то есть язык $\{a_n b_n \mid n \geq 0\}$ не является регулярным.

Пример 27.

Показать, что язык

$$L = \{a_n b_m \mid n \neq m, n \geq 0, m \geq 0\}$$

не является регулярным.

Можно и здесь попытаться прямо использовать Лемму о расширении, но мы дадим более изящное решение, опирающееся на предыдущий пример. Допустим, что L – регулярный язык. Тогда регулярным должен быть и язык

$$L_1 = \bar{L}_1 \cap L(a^* \cdot b^*).$$

Но $L_1 = \{a_n b_n \mid n \geq 0\}$ и, как уже было показано, не является регулярным. Следовательно, и L не может быть регулярным.

Глава 5. Контекстно-свободные языки

Как мы уже показали, не все языки регулярны. Так, например, применив лемму о расширении регулярных языков, мы убедились, что язык $L = \{a_n b_n \mid n \geq 0\}$ не является регулярным. В то же время легко распознать в нем так называемую скобочную структуру, входящую во многие языки программирования. Эта структура соответствует простому правилу: число открывающих скобок в программе должно совпадать с числом закрывающих скобок. В терминах языка L строки $(())$, $((()))$ принадлежат L , а строка $((())$ – нет. Для того чтобы включить в рассматриваемый класс этот и многие другие языки, введем понятие контекстно-свободного языка и контекстно-свободной грамматики. Для этого класса языков мы рассмотрим проблему грамматического разбора, или синтаксического анализа, заключающуюся в том, чтобы по данной строке определить, выводима ли она в данной грамматике. Изучение этой проблемы очень важно для разработки языков программирования и построения компиляторов, так как лексический анализ – это необходимая фаза трансляции языка, а все известные нам языки программирования являются контекстно-свободными.

5.1. Контекстно-свободные грамматики

В регулярной грамматике класс продукций очень ограничен: в левой части любой продукции $A \rightarrow \alpha$ фигурирует один нетерминальный символ A , а правая часть α имеет специальный вид. Можно попытаться ослабить это последнее ограничение, разрешив α быть любой строкой терминальных и нетерминальных символов (в том числе, и пустой). Это расширение и приводит нас к контекстно-свободной грамматике.

Определение 16.

Грамматика $G = (N, T, S, P)$ называется контекстно-свободной (или КС-грамматикой), если любая ее продукция имеет вид $A \rightarrow \alpha$, где $A \in N$, $\alpha \in (N \cup T)^*$.

Язык L называется контекстно-свободным (или КС-языком), если существует контекстно-свободная грамматика G такая, что

$$L = L(G).$$

Так как, очевидно, любая регулярная грамматика контекстно-свободна, то и любой регулярный язык является КС-языком.

С другой стороны, язык $L = \{a^n b^n \mid n \geq 0\}$ не регулярный, но, как это видно из примера 5, является контекстно-свободным.

Следовательно, класс регулярных языков есть собственное подмножество класса КС-языков.

Свое название контекстно-свободные грамматики получили за следующее свойство: замена нетерминального символа в левой части продукции на правую часть в процессе вывода может осуществляться в любой момент, когда этот нетерминальный символ появляется в сентенциальной форме, и независимо от его окружения (то есть контекста). Это является следствием того, что левая часть продукции представляет собою один-единственный нетерминальный символ.

Пример 28.

Грамматика $G = (\{S\}, \{a, b\}, S, P)$, где P есть множество продукций

$$S \rightarrow aSa$$

$$S \rightarrow bSb$$

$$S \rightarrow \varepsilon,$$

является контекстно-свободной. Типичным выводом в этой грамматике является следующий:

$$S \Rightarrow aSa \Rightarrow abSba \Rightarrow abbSbba \Rightarrow abbbbba.$$

Отсюда нетрудно видеть, что $L(G) = \{\alpha\alpha^{-1} \mid \alpha \in \{a, b\}^*\}$. Этот язык контекстно-свободный, но, как было показано ранее, не регулярный.

В контекстно-свободных грамматиках вывод может включать сентенциальные формы, содержащие более одного нетерминального символа. В этом случае появляется возможность выбора, какой из этих символов заменять раньше другого. В качестве примера рассмотрим КС-грамматику

$$G = (\{E, T, F\}, \{a, +, *, (\cdot)\}, E, P)$$

с продукциями

- 1) $E \rightarrow E + T$, 2) $E \rightarrow T$,
- 3) $T \rightarrow T^* F$, 4) $T \rightarrow F$,
- 5) $F \rightarrow (E)$, 6) $F \rightarrow a$.

Рассмотрим два вывода в G строки $a + a$:

$$\begin{array}{cccccccc}
 & 1 & & 2 & & 4 & & 6 & & 4 & & 6 \\
 1) & E \Rightarrow & E + T \Rightarrow & T + T \Rightarrow & F + T \Rightarrow & a + T \Rightarrow & a + F \Rightarrow & a + a; \\
 & 1 & & 4 & & 6 & & 2 & & 4 & & 6 \\
 2) & E \Rightarrow & E + T \Rightarrow & E + F \Rightarrow & E + a \Rightarrow & T + a \Rightarrow & F + a \Rightarrow & a + a.
 \end{array}$$

В этих двух выводах над стрелками указаны номера примененных на соответствующем шаге продукций из P . Можно заметить, что в выводах 1) и 2) использованы одни и те же продукции, но в разном порядке: в выводе 1) на каждом шаге заменяется самый левый нетерминальный символ, а в выводе 2) – самый правый.

Определение 17.

Вывод в КС-грамматике называется *левосторонним*, если на каждом шаге замещается самый левый нетерминальный символ. Если замещается всегда самый правый символ, то вывод называется *правосторонним*.

Таким образом, в рассмотренном ранее примере вывод 1) является левосторонним, а вывод 2) – правосторонним.

Другой способ представления вывода вне зависимости от порядка применения продукций – представление его в виде *дерева вывода*. Это упорядоченное дерево, вершины которого помечены левыми частями продукций, а непосредственные потомки вершины в совокупности представляют собою правую часть соответствующей продукции. Так, если на некотором шаге вывода в грамматике $G = (N, T, S, P)$ была применена продукция $X \rightarrow Y_1 Y_2 \dots Y_m$, а затем продукция $Y_2 \rightarrow Z_1 Z_2 \dots Z_n$, где

$$Y_i, Z_j \in N \cup T, 1 \leq i \leq m, 1 \leq j \leq n, Y_2 \in N,$$

то этой части вывода будет соответствовать фрагмент дерева вывода, представленный на рис. 24.

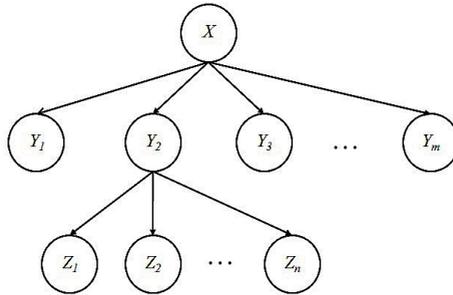


Рис. 24. Фрагмент дерева вывода

Корень дерева вывода помечается начальным символом грамматики, а листья – терминальными символами. В целом, дерево вывода показывает, как в процессе вывода происходит замещение нетерминальных символов.

Определение 18.

Пусть $G = (N, T, S, P)$ – контекстно-свободная грамматика. Дерево вывода в грамматике G – это упорядоченное дерево, удовлетворяющее следующим условиям:

- 1) корень помечен начальным символом S ;
- 2) каждый лист помечен символом из множества $T \cup \{\epsilon\}$;
- 3) каждая вершина, не являющаяся листом, помечена символом из N ;
- 4) если вершина имеет метку $A \in N$, а ее потомки помечены (слева направо) символами X_1, X_2, \dots, X_n , тогда продукция $A \rightarrow X_1 X_2 \dots X_n$ содержится в P ;
- 5) лист, помеченный ϵ , является единственной дочерней вершиной своей родительской вершины.

Если в этом определении исключить условие 1, а условие 2 заменить на условие 2') каждый лист помечен символом из множества

$$N \cup T \cup \{\epsilon\},$$

то получим определение так называемого частичного дерева вывода.

Кроной дерева вывода назовем строку, которая получится, если выпisać слева направо все метки листьев (опуская при этом символы ϵ):

Пример 29.

Рассмотрим грамматику G с продукциями

$$S \rightarrow aSbS \mid bSaS \mid \epsilon.$$

Тогда следующим двум выводам в G строки $abab$:

а) $S \Rightarrow aSbS \Rightarrow abSaSbS \Rightarrow *ab\epsilon a\epsilon b\epsilon$

б) $S \Rightarrow aSbS \Rightarrow a\epsilon bS \Rightarrow a\epsilon b aSbS \Rightarrow *a\epsilon b a\epsilon b\epsilon$

будут соответствовать деревья вывода, изображенные на рис. 25. Дерево вывода представляет собой очень наглядное и легкое для понимания описание вывода. Как и для диаграмм переходов конечных автоматов, такая наглядность оказывается очень полезной при доказательстве различных свойств КС-грамматик. Но вначале установим строгую связь между выводами и деревьями выводов.

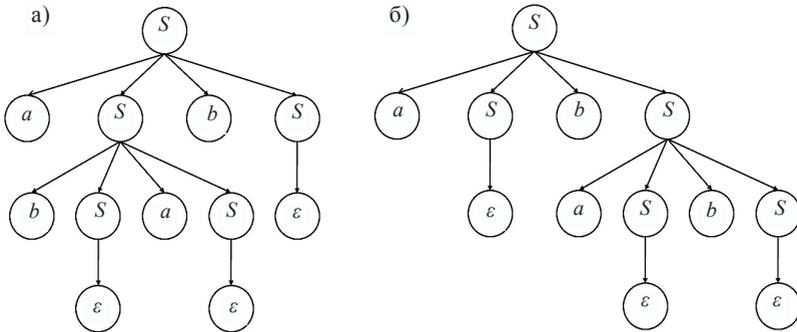


Рис. 25. Деревья выводов

Теорема 20.

Пусть $G = (N, T, S, P)$ – КС-грамматика. Тогда для любой строки $\alpha \in L(G)$ существует дерево вывода, крона которого совпадает с α . И обратно, крона любого дерева вывода в G принадлежит $L(G)$. Кроме того, крона любого частичного дерева вывода в G , корень которого помечен S , является сентенциальной формой в G .

Доказательство.

Покажем вначале, что каждой сентенциальной форме в G соответствует частичное дерево вывода. Используем для этого индукцию по длине вывода. Пусть длина вывода равна 1. Так как вывод $S \Rightarrow \alpha$ влечет наличие продукции $S \rightarrow \alpha$ в G , то требуемое утверждение немедленно следует из Определения 18.

Предположим теперь, что для каждой сентенциальной формы, выводимой за n шагов, существует соответствующее дерево вывода. Пусть произвольная сентенциальная форма ϕ выводима за $(n + 1)$ шаг, тогда вывод можно разбить на две части:

вывод длины n : $S \Rightarrow^* \alpha A \beta$, где $\alpha, \beta \in (N \cup T)^*$, $A \in N$;

и вывод длины 1: $\alpha A \beta \Rightarrow \alpha X_1 X_2 \dots X_n \beta = \phi$, где $X_i \in N \cup T$.

Так как по предположению индукции существует частичное дерево вывода с кроной $\alpha A \beta$ и так как грамматика содержит продукцию

$$A \rightarrow X_1 X_2 \dots X_n,$$

то, наращивая это дерево в вершине A с помощью указанной продукции, мы получим частичное дерево вывода с кроной $\alpha X_1 X_2 \dots X_n \beta = \phi$, что и требовалось.

Нетрудно показать и обратное, то есть что каждое частичное дерево вывода в G имеет в качестве кроны некоторую сентенциальную форму в той же грамматике. Предлагается сделать это самостоятельно в качестве несложного упражнения.

Так как любое дерево вывода является в то же время и частичным деревом вывода, листья которого помечены терминальными символами, то из предыдущих рассуждений следует, что каждая строка в $L(G)$ является кроной некоторого дерева вывода в G , а крона любого дерева вывода в G принадлежит $L(G)$. Теорема доказана.

Итак, для каждой строки $\alpha \in L(G)$ существует соответствующее дерево вывода, которое показывает, какие продукции участвуют в выводе этой строки, но без указания их строгого порядка применения. Но можно представить себе, что при построении дерева вывода наращивается самая левая вершина, помеченная нетерминальным символом. Нетрудно заметить, что такой порядок построения соответствует левостороннему выводу, когда

замещается самый левый нетерминальный символ, входящий в сентенциальную форму. Аналогичные рассуждения показывают существование у строки α и правостороннего вывода.

Следствие 21.

В КС-грамматике G любая строка $\alpha \in L(G)$ имеет левосторонний и правосторонний вывод.

5.2. Грамматический разбор

В предыдущих главах мы интересовались в основном вопросом о том, какое множество строк можно породить с помощью данной грамматики G . В практических приложениях приходится рассматривать и аналитический аспект: как по данной строке α , состоящей из терминальных символов грамматики, определить, входит ли эта строка в язык $L(G)$ или нет, а если входит, то как найти ее вывод в G . Ответ на первый вопрос дает алгоритм, распознающий принадлежность строки α языку $L(G)$. Процесс построения вывода α может быть описан в виде последовательности продукций, с помощью которых строка α выводится в G ; этот процесс называется грамматическим разбором. Для данной строки α из $L(G)$ можно провести грамматический разбор следующим образом: строим все возможные (например, левосторонние) выводы в G и смотрим, какие из них порождают α . Начинаем с просмотра всех продукций вида

$$S \rightarrow \phi$$

и находим все строки ϕ , которые могут быть получены из S за один шаг. Если ни одна из них не совпадает с α , то переходим к следующему этапу: применяем все продукции из G , применимые к ϕ , к самому левому нетерминальному символу в строке ϕ . Это дает нам множество всех сентенциальных форм, некоторые из которых могут вести нас к α . Далее, на каждом промежуточном этапе мы каждый раз выбираем самые левые переменные в полученных на предыдущем этапе строках и применяем к ним все возможные продукции. Может случиться, что некоторые сентенциальные формы не ведут к α , тогда они могут быть исключены

из рассмотрения. Вообще говоря, на n -м этапе мы получим множество всех сентенциальных форм, которые могут быть выведены в G применением n продукций. Если $\alpha \in L(G)$, то эта строка α должна иметь левосторонний вывод конечной длины. Следовательно, применяя описанную процедуру к α , мы на некотором шаге n получим левосторонний вывод для α . Будем называть эту процедуру грамматического разбора методом полного перебора, который относится к группе методов, имеющих общее название – «нисходящий разбор», который соответствует построению дерева вывода сверху вниз, начиная с корня.

Пример 30.

Рассмотрим грамматику $S \rightarrow SS \mid aSb \mid bSa \mid \varepsilon$ и строку $\alpha = aabb$. Однократное применение продукций грамматики дает следующие выводы:

$$S \Rightarrow SS,$$

$$S \Rightarrow aSb,$$

$$S \Rightarrow bSa,$$

$$S \Rightarrow \varepsilon.$$

Имея в виду построение вывода строки α , мы можем исключить из дальнейшего рассмотрения последние два вывода как заведомо не ведущие к цели. Рассмотрим теперь выводы длины 2 для первых двух случаев:

$$1) S \Rightarrow SS \Rightarrow SSS,$$

$$S \Rightarrow SS \Rightarrow aSbS,$$

$$S \Rightarrow SS \Rightarrow bSaS,$$

$$S \Rightarrow SS \Rightarrow S.$$

$$2) S \Rightarrow aSb \Rightarrow aSSb,$$

$$S \Rightarrow aSb \Rightarrow aaSbb,$$

$$S \Rightarrow aSb \Rightarrow abSab,$$

$$S \Rightarrow aSb \Rightarrow ab.$$

Как и на предыдущем шаге, некоторые выводы могут быть исключены из дальнейшего рассмотрения (третий вывод в группе 1 и два последних – в группе 2).

На следующем шаге строим продолжения оставшихся выводов из групп 1 и 2, применяя к каждой из заключительных сен-

тенциальных форм последовательно все применимые продукции грамматики. В результате получим выводы длины 3, среди которых находим вывод

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb.$$

Это означает, что данная строка α принадлежит языку, порождаемому рассматриваемой грамматикой.

Грамматический разбор методом полного перебора имеет серьезные недостатки. Во-первых, этот алгоритм недостаточно эффективен по времени работы и по объему используемой памяти, а во-вторых, он может расходиться на строке, не принадлежащей порождаемому языку. Так, в только что рассмотренном примере, если мы вместо строки α возьмем строку $\beta = abb$ и запустим описанный выше алгоритм, то он будет без конца порождать все новые и новые сентенциальные формы, безуспешно пытаясь построить несуществующий вывод строки β в данной грамматике.

Заметим, что если грамматика не содержит продукций вида $A \rightarrow \varepsilon$ и $A \rightarrow B$, то применение любой продукции такой грамматики либо увеличивает длину сентенциальной формы, либо добавляет в нее терминальный символ. А это дает нам эффективный критерий остановки алгоритма: как только длина сентенциальной формы становится больше длины данной строки, то работа алгоритма на данной ветке прекращается. Очевидно, что через конечное число шагов работа алгоритма на всех ветвях прекратится, даже если данная строка не принадлежит рассматриваемому языку. В дальнейшем будет показано, что указанное ограничение на вид продукций грамматики не сужает класс языков, к которым может применяться рассмотренный алгоритм. Так, например, грамматика

$$S \Rightarrow SS \mid aSb \mid bSa \mid ab \mid ba$$

удовлетворяет указанным требованиям и порождает тот же язык, что и в примере 29, но не содержащий пустой строки. При этом для любой строки $\alpha \in \{a, b\}^+$ грамматический разбор методом полного перебора всегда заканчивается не более чем за $2n$ шагов, где n равно длине строки α . В результате мы получаем либо вывод строки α , либо информацию о том, что α не при-

надлежит языку, порождаемому данной грамматикой. Резюмируем сказанное в виде следующей теоремы для КС-языков.

Теорема 22.

Пусть $G = (N, T, S, P)$ – контекстно-свободная грамматика, не содержащая продукций вида $A \rightarrow \varepsilon$ или $A \rightarrow B$, где $A, B \in N$. Тогда грамматический разбор методом полного перебора может быть трансформирован в алгоритм, который для любой строки $\alpha \in T^*$ либо строит ее вывод, либо сообщает, что вывод для α не существует.

Доказательство.

Рассмотрим $\alpha \in T^*$ и любой вывод в G :

$$S \Rightarrow \phi_1 \Rightarrow \phi_2 \Rightarrow \dots \Rightarrow \phi_n,$$

где ϕ_i – сентенциальные формы. Для каждой сентенциальной формы введем две числовые характеристики: ее длину и количество терминальных символов в ней. Очевидно, при переходе от ϕ_i к ϕ_{i+1} возрастает, по крайней мере, одна из этих характеристик. Если нас интересует вывод строки α , то ни длина ϕ_i , ни число терминальных символов не могут превышать α . Следовательно, сам вывод не может иметь более 2α шагов. Просмотрев все такие выводы, мы либо найдем среди них вывод α , либо заключим, что $\alpha \notin L(G)$.

Что касается неэффективности метода полного перебора при грамматическом разборе, то эта проблема намного сложнее. На первом шаге мы можем породить $|P|$ сентенциальных форм (если каждая из продукций применима к S), на втором шаге к каждой из полученных сентенциальных форм мы снова можем применить $|P|$ продукций, то есть получаем $|P|^2$ форм, и так далее. В целом, применение полного перебора может дать $|P|^{|\alpha|}$ сентенциальных форм. Если под строкой α понимать текст программы в некотором языке программирования, то легко понять, что полученная экспоненциальная оценка сложности делает невозможным практическое применение данного метода. Существуют более эффективные алгоритмы грамматического разбора, но они не столь просты для понимания и их изучение выходит за рамки нашего курса. Такие алгоритмы рассматриваются

в курсе, посвященном методам построения компиляторов. Мы здесь лишь отметим, что практически применимыми являются те методы грамматического разбора, для которых время работы пропорционально длине рассматриваемой строки. В общем случае нам не известны такие алгоритмы, применимые ко всем КС-языкам, но они существуют для целого ряда ограниченных, но практически важных специальных классов КС-языков.

Пример 31.

Рассмотрим такой тип контекстно-свободной грамматики, у которой все продукции имеют вид

$$A \rightarrow a\alpha,$$

где $A \in N$, $a \in T$, $\alpha \in N^*$, причем каждая пара символов (A, a) может входить не более чем в одну продукцию. Покажем, что для любой строки α , выводимой в этой грамматике, грамматический разбор может быть произведен не более чем за α шагов. Возьмем произвольную строку

$$\alpha = a_1 a_2 \dots a_n$$

и применим к ней метод полного перебора для построения ее вывода. Так как существует единственная пара (A_1, a_1) , входящая в продукцию, применимую на первом шаге, то единственным образом получаем:

$$S \Rightarrow a_1 A_1 A_2 \dots A_m.$$

Затем, заменяя переменную A_1 и учитывая, что существует лишь единственная продукция, соответствующая паре (A_1, a_2) , получаем:

$$S \Rightarrow *a_1 a_2 B_1 \dots B_k A_2 \dots A_m.$$

Ясно, что на каждом шаге в сентенциальную форму добавляется ровно один терминальный символ, следовательно, через α шагов алгоритм остановится.

Рассмотренная грамматика называется простой грамматикой (или S -грамматикой). Несмотря на кажущуюся простоту S -грамматик, в них могут быть выражены многие распространенные конструкции языков программирования.

5.3. Неоднозначность грамматик и языков

Из предыдущих рассуждений ясно, что для произвольной строки $\alpha \in L(G)$ грамматический разбор методом полного перебора дает нам некоторое дерево вывода для α . Если существует несколько таких деревьев для одной и той же строки, то говорят, что имеет место *неоднозначность*.

Определение 23.

Контекстно-свободная грамматика G называется *неоднозначной*, если существует некоторая строка $\alpha \in L(G)$, которая имеет в G , по крайней мере, два различных дерева вывода. В противном случае КС-грамматика называется *однозначной*.

Нетрудно заметить, что неоднозначность влечет существование двух или более различных левосторонних (или правосторонних) выводов.

Пример 32.

Покажем, что грамматика

$$S \rightarrow aSb \ SS \ \varepsilon$$

неоднозначна. Для этого возьмем строку

$$\alpha = aabb$$

и построим для нее два вывода в этой грамматике:

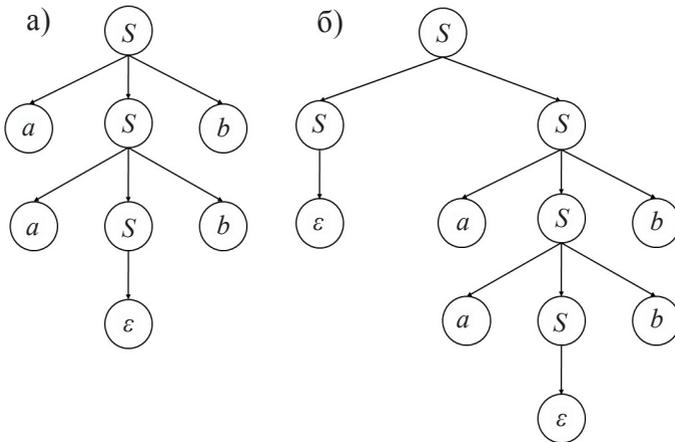


Рис. 26. Два вывода строки $aabb$ в грамматике S

В естественных языках неоднозначность является одним из присущих им свойств, к которому относятся вполне терпимо и которое в ряде случаев эффективно используют. Что же касается языков программирования, где для каждого выражения должна быть единственная интерпретация, с неоднозначностью следует по мере возможностей бороться. Часто, чтобы избавиться от нее, бывает достаточно переписать грамматику в эквивалентной, но уже однозначной форме.

Пример 33.

Рассмотрим грамматику $G = (N, T, S, P)$, где $N = \{E, I\}$, $T = \{a, b, c, +, *, (,)\}$ и P – множество следующих продукций:

$$\begin{aligned} E &\rightarrow I, \\ E &\rightarrow E + E, \\ E &\rightarrow E * E, \\ E &\rightarrow (E), \\ I &\rightarrow a b c. \end{aligned}$$

Нетрудно видеть, что эта грамматика порождает собственное подмножество арифметических выражений в таких языках программирования, как, например, Паскаль или Фортран. Покажем, что эта грамматика неоднозначна. Возьмем строку $a + b * c$, которая принадлежит языку $L(G)$, и построим для нее два дерева вывода a) и b), рис. 27.

Для того чтобы устранить неоднозначность, в подобных случаях иногда вводят дополнительные правила, устанавливающие относительные приоритеты арифметических операций. Так, обычно считают, что операция умножения $*$ связывает операнды сильнее, чем сложение $+$. Тогда на рис. 27. дерево вывода a) должно будет рассматриваться как правильное, соответствующее тому, что произведение $b * c$ является подформулой, значение которой должно быть вычислено раньше, чем будет произведено сложение.

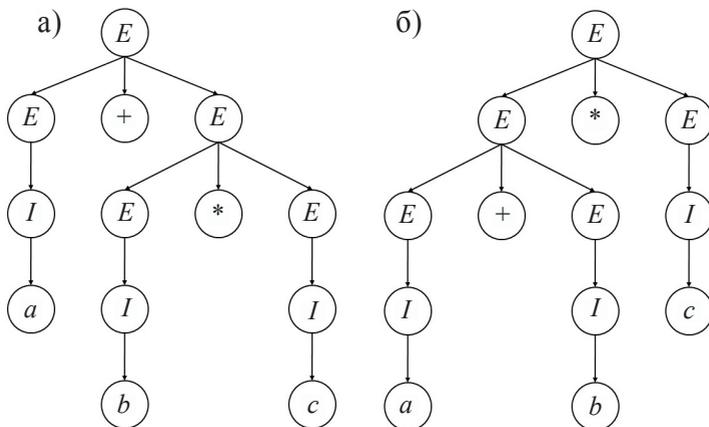


Рис. 27. Два дерева вывода для $a + b * c$

Однако эти правила не соответствуют нашим грамматическим правилам, имеющим строго определенную форму продукций. Поэтому попробуем так видоизменить данную грамматику, чтобы она стала однозначной. Для этого в качестве N возьмем следующее множество переменных: $\{E, M, F, I\}$ и новый набор продукций:

$$\begin{aligned}
 E &\rightarrow M, \\
 M &\rightarrow F, \\
 F &\rightarrow I, \\
 E &\rightarrow E + M, \\
 M &\rightarrow M * F, \\
 F &\rightarrow (E), \\
 I &\rightarrow a b c.
 \end{aligned}$$

В этой грамматике строка $a + b * c$ будет иметь уже единственное дерево вывода, представленное на рис. 28. Более того, можно показать, что и для любой другой строки, выводимой в этой грамматике, соответствующее дерево вывода единственно. А это означает, что преобразованная грамматика однозначна. Если же учесть, что она эквивалентна исходной грамматике (в чем нетрудно убедиться), то можно считать, что мы достигли цели и избавились от неоднозначности.

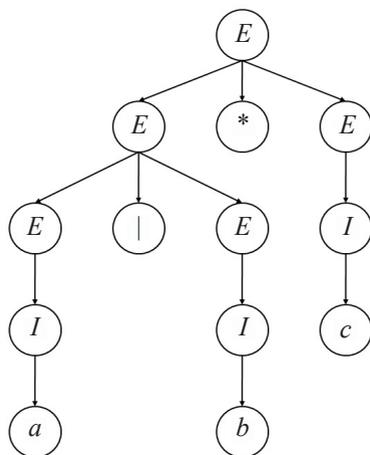


Рис. 28. Единственное дерево вывода для $a + b * c$

К сожалению, не всегда так легко удается решить проблему неоднозначности. Даже вопрос, однозначна грамматика или нет, часто оказывается очень трудным, впрочем, как и вопрос, эквивалентны ли две грамматики. В действительности, ни для первого, ни для второго вопросов не существует универсальных алгоритмов, которые всегда давали бы ответы.

В рассмотренном примере нам удалось избавиться от неоднозначности, выбрав другую форму грамматики для данного языка. Но, к сожалению, оказывается, что это не всегда возможно, так как иногда неоднозначность присуща самому языку.

Определение 24.

Контекстно-свободный язык L называется *однозначным*, если для него существует однозначная грамматика. Если же любая грамматика, порождающая L , неоднозначна, то такой язык называется *существенно неоднозначным*.

Проблема существенной неоднозначности языков является очень трудной, и ее обсуждение выходит за рамки нашего курса. Поэтому ограничимся лишь одним примером.

Пример 34.

Язык $L = \{a^n b^n c^m \mid n, m \geq 0\} \cup \{a^n b^m c^m \mid n, m \geq 0\}$ является существенно неоднозначным контекстно-свободным языком.

Покажем, что язык L – контекстно-свободный.

Нетрудно проверить, что язык $L = \{a^n b^n c^m \mid n, m \geq 0\}$ порождается грамматикой G_1 :

$$S_1 \rightarrow aS_1 b C \mid \varepsilon,$$

$$C \rightarrow cC \mid \varepsilon,$$

а язык $L_2 = \{a^n b^m c^m \mid n, m \geq 0\}$ порождается грамматикой G_2 :

$$S_2 \rightarrow aS_2 \mid B,$$

$$B \rightarrow bBc \mid \varepsilon.$$

Тогда, очевидно, язык $L = L_1 \cup L_2$ порождается грамматикой, множество продукций которой состоит из всех продукций грамматик G_1 и G_2 и двух дополнительных продукций $S \rightarrow S_1 \mid S_2$.

Мы не будем приводить здесь технически сложное доказательство существенной неоднозначности языка L , а отметим только, что эта неоднозначность связана с невозможностью выразить с помощью одного и того же множества продукций два «противоречивых» требования: в языке L_1 количества символов a и b в строке должны совпадать между собой и почти всегда отличаться от количества символов c , тогда как в языке L_2 количества символов b и c в строке совпадают между собой и почти всегда отличаются от количества символов a .

Глава 6. Преобразования КС-грамматик и нормальные формы

Определение контекстно-свободной грамматики не содержит никаких ограничений на вид правой части продукций. Оказывается, такая свобода является излишней, и можно, не умаляя общности, отказаться от некоторых случаев, которые затрудняют синтаксический анализ. Так, например, можно без изменения порождаемого языка исключить из грамматики продукции вида $A \rightarrow \varepsilon$ и $A \rightarrow B$. В связи с этим возникает необходимость изучения проблемы преобразования КС-грамматики к эквивалентному виду, удовлетворяющему определенным ограничениям на вид продукций. Наибольший интерес представляют для нас так называемые *нормальные формы* КС-грамматик с весьма сильными ограничениями на вид продукций, которые, однако, достаточно широки для того, чтобы любую КС-грамматику можно было преобразовать к эквивалентной КС-грамматике в нормальной форме. В этой главе мы рассмотрим две нормальные формы КС-грамматик: *нормальную форму Хомского* и *нормальную форму Грейбах*, имеющих исключительно полезные теоретические и практические приложения.

6.1. Методы преобразования грамматик

Прежде всего сделаем одно важное замечание по поводу пустой строки ε . Как уже говорилось, наличие в грамматике пустой строки сильно усложняет рассуждения относительно такой грамматики, поэтому будем предполагать, что рассматриваемые нами КС-грамматики не содержат ε -продукций, то есть продукций вида $A \rightarrow \varepsilon$. Это не сузит наши возможности в изучении свойств КС-грамматик и КС-языков. Действительно, пусть L – контекстно-свободный язык и пусть

$$G = (N, T, S, P)$$

Это КС-грамматика, порождающая язык $L \setminus \{\varepsilon\}$. Тогда, добавив к N новый нетерминальный символ S' в качестве начального символа и расширив P двумя новыми продукциями

$$S' \rightarrow S \mid \varepsilon,$$

мы получим новую грамматику G' , порождающую язык L . Поэтому любое нетривиальное заключение по поводу свойств языка $L \setminus \{\varepsilon\}$ может быть перенесено и на язык L . Кроме того, существует метод получения из любой КС-грамматики G некоторой другой КС-грамматики G_ε такой, что

$$L(G_\varepsilon) = L(G) \setminus \{\varepsilon\}.$$

Следовательно, для всех приложений можно ограничиться случаем, когда контекстно-свободный язык не содержит ε , что мы и будем предполагать в дальнейшем.

Изучение преобразований КС-грамматик начнем с так называемых правил подстановки.

Определение 25.

Продукции вида $A \rightarrow \alpha$, где A – нетерминальный символ, а строка α отлична от ε , будем называть A -продукциями.

Теорема 26.

Пусть $G = (N, T, S, P)$ – КС-грамматика. Предположим, что P содержит продукцию вида

$$A \rightarrow \alpha_1 B \alpha \tag{1}$$

Где A и B – различные символы из N . Допустим, что

$$B \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n \tag{2}$$

множество всех B -продукций в P . Пусть $G' = (N, T, S, P')$ – грамматика, в которой P' получено удалением из P продукции (1) и добавлением новых продукций

$$A \rightarrow \alpha_1 \beta_1 \alpha_2 \mid \alpha_1 \beta_2 \alpha_2 \mid \dots \mid \alpha_1 \beta_n \alpha_2.$$

Тогда $L(G') = L(G)$.

Доказательство.

Пусть $\phi \in L(G)$, т. е.

$$S \Rightarrow_G^* \phi. \tag{3}$$

Если в этом выводе продукция (1) не участвует, тогда, очевидно,

$$S \Rightarrow_G^* \phi.$$

Допустим, что продукция (1) существенным образом используется в выводе (3). Тогда, не уменьшая общности, можно пред-

положить, что символ B замещается сразу на следующем шаге вывода (3) по одному из правил (2), т. е.

$$S \Rightarrow_G^* \gamma_1 A \gamma_2 \Rightarrow_G \gamma_1 \alpha_1 B \alpha_2 \gamma_2 \Rightarrow_G \gamma_1 \alpha_1 \beta_j \alpha_2 \gamma_2.$$

Так как в выводе $S \Rightarrow_G^* \gamma_1 A \gamma_2$ продукция (1) не используется, то мы, очевидно, можем записать:

$$S \Rightarrow_{G'}^* \gamma_1 A \gamma_2 \Rightarrow_{G'}^* \gamma_1 \alpha_1 \beta_j \alpha_2 \gamma_2.$$

Таким образом, если сентенциальная форма выводится в G , то она выводима и в G' . Если далее в выводе (3) продукция (1) снова используется, опять повторяем то же рассуждение. Окончательно получаем, что

$$S \Rightarrow_{G'}^* \phi.$$

Значит, из того, что $\phi \in L(G)$, следует $\phi \in L(G')$. Аналогичными рассуждениями можно легко показать, что если $\phi \in L(G')$, то $\phi \in L(G)$, что и завершает доказательство. **Теорема доказана.**

Пример 35.

Рассмотрим КС-грамматику

$$G = (\{B, S\}, \{a, b\}, S, P)$$

с продукциями

$$\begin{aligned} S &\rightarrow a \mid aaS \mid abBa \\ B &\rightarrow abbS \mid b. \end{aligned}$$

Производя подстановку вместо символа B в правой части продукции $S \rightarrow abBc$ на правые части B -продукций, получаем новую грамматику G' с множеством продукций

$$S \rightarrow a \mid aaS \mid ababbSc \mid abbc.$$

При этом G' эквивалентна G . В частности, строка $aaabbc$ выводима как в G :

$$S \Rightarrow aaS \Rightarrow aaabBc \Rightarrow aaabbc,$$

так и в G' :

$$S \Rightarrow aaS \Rightarrow aaabbc.$$

В теореме 26 мы рассмотрели случай замены продукции $A \rightarrow \alpha_1 B \alpha_2$, когда $A \neq B$. В следующей теореме рассматривается случай $A = B$.

Определение 27.

Продукции вида $A \rightarrow A\alpha$, где $A \in N$, $\alpha \in (N \cup T)^*$, назовем леворекурсивными продукциями.

Теорема 28.

Пусть дана КС-грамматика $G = (N, T, S, P)$. Допустим, что

$$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \dots \mid A\alpha_n, \quad (4)$$

где $A \in N$, $\alpha_i \in (N \cup T)^*$, $i = 1, 2, 3, \dots, n$, – все ее леворекурсивные A -продукции, а

$$A \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_m, \quad (5)$$

где $\beta_i \in (N \cup T)^*$, $i = 1, 2, 3, \dots, m$, – все остальные ее A -продукции. Рассмотрим грамматику

$$G' = (N \cup \{Z_A\}, T, S, P')$$

где $Z_A \notin N$, а P' получено заменой всех продукций вида (4) и (5) на следующую совокупность продукций:

$$\begin{aligned} A &\rightarrow \beta_i \mid \beta_i Z_A, \quad i = 1, 2, \dots, m, \\ Z_A &\rightarrow \alpha_j \mid \alpha_j Z_A, \quad j = 1, 2, \dots, n. \end{aligned}$$

Тогда $L(G) = L(G')$.

Доказательство.

Рассмотрим вывод в грамматике G , использующий нетерминальный символ A в последовательности шагов, на каждом из которых применяется продукция вида (4):

$$A \Rightarrow A\alpha_i \Rightarrow A\alpha_i \alpha_i \Rightarrow^*_G A\alpha_k \dots \alpha_j \alpha_i.$$

Для получения в итоге строки терминальных символов нам необходимо применить хотя бы одну из продукций (5), что дает

$$A \Rightarrow^*_G \beta_r \alpha_k \dots \alpha_j \alpha_r, \quad 1 \leq r \leq m.$$

Но этот вывод можно заменить выводом в G' с тем же результатом:

$$A \Rightarrow \beta_r Z_A \Rightarrow \beta_r \alpha_k Z_A \Rightarrow^* \beta_r \alpha_k \dots \alpha_j Z_A \Rightarrow \beta_r \alpha_k \dots \alpha_j \alpha_r.$$

Следовательно, для любой строки ϕ такой, что $A\phi$, мы имеем: $A\phi$. Нетрудно показать и обратное, то есть если имеет место вывод $A\phi$, то возможен и вывод $A\phi$.

Итак, мы получили способ избавиться от леворекурсивных продукций в КС-грамматиках. Эти продукции являются частным случаем левой рекурсии; в общем случае грамматика G называется леворекурсивной, если существует нетерминал A , для которого возможен вывод AAa . Это свойство чаще всего является нежелательным, и Теорема 28 позволяет систематически исключать леворекурсивные продукции, которые вначале могут быть полезны при построении грамматики.

Пример 36.

Пусть G_0 – грамматика с продукциями:

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T \times F \mid F$$

$$F \rightarrow (E) \mid a.$$

Применяя к ней процедуру из Теоремы 28, получим эквивалентную ей грамматику со следующим множеством продукций:

$$E \rightarrow T \mid TZ_E$$

$$Z_E \rightarrow + T \mid + TZ_E$$

$$T \rightarrow F \mid FZ_T$$

$$Z_T \rightarrow \times F \mid \times FZ_T$$

$$F \rightarrow (E) \mid a.$$

Следующий тип продукций, которые мы хотели бы научиться исключать из грамматики, – это так называемые бесполезные продукции, то есть те, которые никогда не участвуют в выводе никакой строки терминалов. Например, в грамматике с продукциями

$$S \rightarrow aSb \mid \varepsilon \mid A$$

$$A \rightarrow aA$$

последняя продукция является бесполезной, так как символ A никогда не может быть преобразован в строку терминалов. Если при выводе из S символ A окажется входящим в сентенциальную форму, то она никогда не сможет быть трансформирована в сентенцию. Значит, удаление продукции $A \rightarrow aA$ не окажет влияния на язык, порождаемый грамматикой, а лишь приведет к упрощению множества продукций.

Определение 29.

В КС-грамматике $G = (N, T, S, P)$ нетерминал $A \in N$ называется *полезным*, если существует хотя бы одна строка $\phi \in L(G)$ такая, что

$$S \Rightarrow^* \alpha A \beta \Rightarrow^* \phi, \alpha, \beta \in (N \cup T)^*.$$

Нетерминал, не являющийся полезным в грамматике G , называется *бесполезным*.

Продукция в грамматике G является *бесполезной*, если в нее входит хотя бы один бесполезный нетерминал.

Нетерминальный символ может быть бесполезным по двум причинам: или он недостижим из начального символа грамматики, или из него не выводима строка терминалов. Рассмотренный выше пример относится ко второму случаю. Возьмем другую грамматику с начальным символом S и следующими продукциями:

$$S \rightarrow A$$

$$A \rightarrow aA \mid \varepsilon$$

$$B \rightarrow bA.$$

Здесь как символ B , так и продукция $B \rightarrow bA$ являются бесполезными, и хотя из B можно вывести терминальную строку, но не существует ни одного вывода вида

$$S \Rightarrow^* \alpha B \beta.$$

Для исключения нетерминалов, недостижимых из начального символа грамматики, полезен *граф зависимости* нетерминалов. Он наглядно показывает сложные связи между нетерминалами по выводимости и находит много применений.

Определение 30.

Для КС-грамматики *граф зависимости* – это граф, вершины которого помечены нетерминалами и в котором дуга между двумя вершинами A и B существует тогда и только тогда, когда в грамматике есть продукция вида $A \rightarrow \alpha B \beta$.

Нетрудно заметить, что нетерминал X полезен только тогда, когда в графе зависимости существует путь из вершины S в вершину X .

Пример 37.

Пусть дана КС-грамматика $G = (N, T, S, P)$, где $N = \{S, A, B, C\}$, $T = \{a, b\}$, а P состоит из продукций

$$S \rightarrow aS \mid A \mid C$$

$$A \rightarrow a$$

$$B \rightarrow aa$$

$$C \rightarrow aCb.$$

Требуется удалить из G все бесполезные символы и продукции.

Вначале определим множество нетерминалов, которые могут породить терминальные строки. Так как $A \rightarrow a$ и $B \rightarrow aa$, то A и B принадлежат этому множеству. То же самое справедливо для S , так как $S \Rightarrow A \Rightarrow a$. А вот нетерминал C не войдет в это множество, то есть он является бесполезным символом так же, как и те продукции, которые его содержат. Исключая нетерминал C вместе с соответствующими продукциями, получаем грамматику $G' = (N', T, S, P')$, где $N' = \{S, A, B\}$, а P' включает продукции

$$S \rightarrow aS \mid A$$

$$A \rightarrow a$$

$$B \rightarrow aa.$$

Теперь наша задача – удалить из G' недостижимые нетерминалы, то есть символы из N' , недостижимые с помощью вывода из S . Воспользуемся для этого графом зависимости для N' , изображенным на рис. 29:

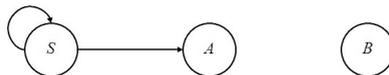


Рис. 29. Граф зависимости

На основании вида этого графа делаем вывод, что терминал B недостижим, а, следовательно, и бесполезен. Удаляя его и соответствующую продукцию, окончательно получаем грамматику $G'' = (\{S, A\}, \{a\}, S, P'')$, где P'' состоит из трех продукций:

$$S \rightarrow aS \mid A$$

$$A \rightarrow a.$$

Очевидно, что при этом $L(G'') = L(G)$.

Обобщим наши рассуждения в следующей теореме.

Теорема 31.

Для любой КС-грамматики $G = (N, T, S, P)$ существует эквивалентная грамматика, не содержащая ни бесполезных нетерминальных символов, ни бесполезных продукций.

Доказательство.

Построение проведем в два этапа. Сначала построим грамматику $G' = (N', T, S, P')$ такую, что N' содержит только такие нетерминалы A , для которых возможен вывод $A\phi$, $\phi \in T^*$. Это построение проведем с помощью следующего алгоритма:

Шаг 1. Полагаем N' равным \emptyset .

Шаг 2. Выполнять цикл до тех пор, пока больше не будет добавлен новый символ в N' : для каждого $A \in N$, для которого в P есть продукция вида

$$A \rightarrow \alpha_1 \alpha_2 \dots \alpha_n, \text{ где } \alpha_i \in T^* \cup N', i = 1, 2, \dots, n,$$

добавить A в N' .

Шаг 3. Поместить в P' все продукции из P , образованные из символов (кроме \rightarrow), входящих в множество

$$N' \cup T \cup \{\varepsilon\}.$$

Нетрудно видеть, что этот алгоритм, начав работать на G , через конечное число шагов останавливается. Ясно также, что если $A \in N'$, то в G' возможен вывод $A\phi$, $\phi \in T^*$. Надо лишь показать, что каждый символ A , для которого $A\phi = ab\dots$, будет добавлен к N' прежде, чем процедура закончит работу. Чтобы убедиться в этом, рассмотрим такой нетерминал A и соответствующее этому выводу частичное дерево вывода (рис. 30).

На уровне k присутствуют лишь вершины, помеченные терминалами, поэтому каждый символ A_i на уровне $k-1$ будет помещен в N' на первом витке цикла *Шага 2* алгоритма. Любой нетерминал на уровне $k-2$ будет добавлен к N' на втором витке цикла *Шага 2*. На третьем витке все переменные уровня $k-3$ будут добавлены к N' и т. д. Алгоритм не остановится до тех пор, пока в дереве есть еще нетерминалы, не вошедшие в N' . Следовательно, рано или поздно A будет добавлен к N' .

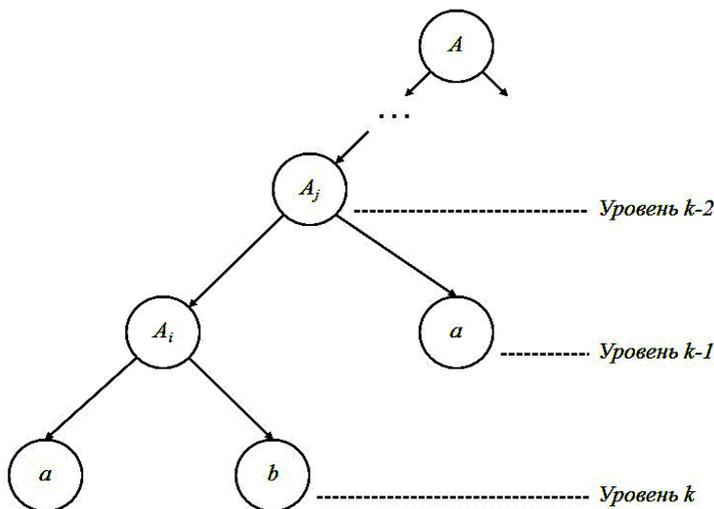


Рис. 30. Частичное дерево вывода

Теперь переходим ко второму этапу построения грамматики \hat{G} . Построим граф зависимости для \hat{G} и с помощью него найдем все недостижимые нетерминальные символы. Исключим их из G' вместе с содержащими эти символы продукциями. Исключим также те терминалы, которые не входят в какую-нибудь полезную продукцию. В результате получим грамматику $\hat{G} = (\hat{N}, \hat{T}, S, \hat{P})$. По построению, \hat{G} не содержит бесполезных символов или продукций.

Пусть для произвольной строки $\phi \in L(G)$ существует вывод в G :

$$S \Rightarrow_G^* \alpha A \beta \Rightarrow_G^* \phi.$$

Так как по построению грамматика \hat{G} сохранила A и все связанные с ним полезные продукции, то у нас есть все необходимое, чтобы построить соответствующий вывод в :

$$S \Rightarrow_{\hat{G}}^* \alpha A \beta \Rightarrow_{\hat{G}}^* \phi.$$

Грамматика \hat{G} получилась из G удалением части продукций, значит, $\hat{P} \subseteq P$ и $L(\hat{G}) \subseteq L(G)$.

В итоге получаем равенство:

$$L(\hat{G}) = L(G),$$

то есть грамматики G и \hat{G} эквивалентны. **Теорема доказана.**

Следующий тип продукций, которые в ряде случаев нежелательны, – это продукции с пустой строкой в правой части.

Определение 32.

Продукция КС-грамматики, имеющая вид

$$A \rightarrow \varepsilon,$$

называется ε -продукцией. Нетерминал A , для которого в грамматике возможен вывод $A \Rightarrow^* \varepsilon$, называется *обнуляемым*.

Заметим, что грамматика может породить язык, не содержащий ε , имея при этом ε -продукции или обнуляемые нетерминалы. В этом случае ε -продукции могут быть исключены из грамматики.

Теорема 33.

Пусть КС-грамматика G порождает язык $L(G)$, не содержащий ε . Тогда существует эквивалентная ей грамматика G' , не содержащая ε -продукций.

Доказательство.

Вначале построим множество N_0 всех обнуляемых нетерминалов в G , используя следующую процедуру:

Шаг 1. Для всех продукций $A \rightarrow \varepsilon$ помещаем A в N_0 .

Шаг 2. Повторяем следующий цикл до тех пор, пока возможно добавление нетерминалов в N_0 : для всех продукций $B \rightarrow A_1 A_2 \dots A_n$, где $A_1, A_2, \dots, A_n \in N_0$, поместить B в N_0 .

После окончания формирования множества N_0 можем перейти к построению P' . Рассмотрим все продукции из P вида $A \rightarrow X_1 X_2 \dots X_m$, $m \geq 1$, для которых $X_i \in N \cup T$. Для каждой такой продукции из P мы помещаем в P' как саму эту продукцию, так и все производные от нее продукции, получающиеся стиранием любого количества обнуляемых символов в правой части исходной продукции, за одним исключением: если все X_i обнуляемые, то продукция $A \rightarrow \varepsilon$ в P' не добавляется. Эквивалентность

получившейся при этом грамматики G' исходной грамматике G почти очевидна, и мы оставляем доказательство этого в качестве упражнения читателю. **Теорема доказана.**

Пример 38.

Найти КС-грамматику без ε -продукций, эквивалентную грамматике с продукциями:

$$S \rightarrow ABaC,$$

$$A \rightarrow BC,$$

$$B \rightarrow b \mid \varepsilon,$$

$$C \rightarrow D \mid \varepsilon,$$

$$D \rightarrow d.$$

Находим множество обнуляемых нетерминалов: A, B, C . Затем видоизменяем совокупность продукций; получаем:

$$S \rightarrow ABaC \mid BaC \mid AaC \mid ABa \mid aC \mid Aa \mid Ba \mid a,$$

$$A \rightarrow B \mid C \mid BC,$$

$$B \rightarrow b,$$

$$C \rightarrow D,$$

$$D \rightarrow d.$$

Последний класс «нежелательных» продукций, который мы рассмотрим, – это продукты, у которых левая и правая части представляют собой изолированные нетерминальные символы.

Определение 34.

Продукция КС-грамматики, имеющая вид $A \rightarrow B$, где A, B – нетерминальные символы, называется *цепной продукцией*.

Теорема 35.

Любая КС-грамматика $G = (N, T, S, P)$ без ε -продукций может быть преобразована к эквивалентной грамматике $G' = (N', T', S, P')$, не имеющей цепных продукций.

Доказательство.

Очевидно, любая цепная продукция вида $A \rightarrow A$ может быть удалена из грамматики без всякого ущерба для порождаемого языка, поэтому будем рассматривать продукты вида $A \rightarrow B$, где A и B различные нетерминалы.

Прежде всего, для каждого $A \in N$ найдем множество

$$Ded(A) = \{B \mid A \Rightarrow_G^* B\}.$$

Это можно сделать с помощью графа зависимости для G : он будет содержать дугу (C, D) , если в грамматике есть продукция $C \rightarrow D$. Тогда $A \Rightarrow_G^* B$ имеет место всякий раз, когда существует путь из A в B .

Построение новой грамматики G' начинаем с помещения в P' всех продукций из P , не являющихся цепными. Пусть, далее, $B \in Ded(A)$. Тогда добавляем в P' новые продукции

$$A \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n,$$

где $B \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$ – множество всех B -продукций в P' (не в $P!$). Заметим, что, так как продукции $B \rightarrow \beta_i$ берутся из P' , ни одна из строк β_i не может быть единственным нетерминалом, следовательно, ни одной цепной продукции в P' добавлено не будет. Как это было сделано в теореме 26, нетрудно показать, что получившаяся грамматика G' эквивалентна G .

Пример 39.

Исключить все цепные продукции из грамматики

$$S \rightarrow Aa \mid B$$

$$B \rightarrow A \mid bb$$

$$A \rightarrow a \mid bc \mid B.$$

Строим граф зависимости (рис. 31):

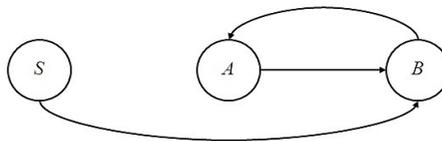


Рис. 31. Граф зависимости

Видим, что $Ded(S) = \{A, B\}$, $Ded(A) = \{B\}$, $Ded(B) = A$. Исключаем из грамматики все цепные продукции $S \rightarrow B$, $B \rightarrow A$ и $A \rightarrow B$ и добавляем к оставшимся новые продукции; получаем:

$$S \rightarrow Aa$$

$$A \rightarrow a \mid bc$$

$$B \rightarrow bb$$

«старые» продукции

$$S \rightarrow a \mid bc \mid bb$$
$$A \rightarrow bb \quad \text{«новые» продукции}$$
$$B \rightarrow a \mid bc$$

Объединяя полученные результаты, можно показать, что КС-грамматики могут быть «очищены» от бесполезных продукций, ε -продукций и цепных продукций.

Теорема 36.

Для любого КС-языка L , не содержащего ε , существует КС-грамматика, порождающая L и не содержащая ни бесполезных продукций, ни ε -продукций, ни цепных продукций.

Доказательство.

Теоремы 31, 33 и 35, каждая по отдельности, позволяют избавляться от продукций соответствующих типов. Существует, однако, опасность, что при удалении одного типа продукций могут возникнуть продукции другого, ранее удаленного, типа. Заметим, что процедура удаления из грамматики цепных продукций не порождает ε -продукций, а процедура удаления бесполезных продукций не создает ни ε -продукций, ни цепных продукций. Следовательно, можно удалить все нежелательные продукции в следующем порядке:

1. Удаляем ε -продукции.
2. Удаляем цепные продукции.
3. Удаляем бесполезные продукции.

Рассмотренные методы преобразования КС-грамматик будут использованы нами в следующем разделе для приведения этих грамматик к нормальным формам.

6.2. Нормальные формы КС-грамматик

Существует много различных нормальных форм контекстно-свободных грамматик, имеющих разные области применения и широту использования. Мы здесь остановимся на двух из них как на наиболее важных и полезных для наших целей. Одна из этих двух форм имеет жесткое ограничение на длину правой части продукций грамматики: допускается использование не более двух символов. Она представляет собой *нормальную форму Хомского*.

Определение 37.

КС-грамматика $G = (N, T, S, P)$ имеет *нормальную форму Хомского*, если любая ее продукция имеет вид либо

$$A \rightarrow BC,$$

либо $A \rightarrow a$, где $A, B, C \in N$, $a \in T$.

Теорема 38.

Любая КС-грамматика $G = (N, T, S, P)$ такая, что $\varepsilon \notin L(G)$, может быть преобразована в эквивалентную ей грамматику $G_1 = (N_1, T, S, P_1)$, имеющую нормальную форму Хомского.

Доказательство.

В силу теоремы 36 мы можем, не нарушая общности, считать, что G не содержит ни ε -продукций, ни цепных продукций. Построение грамматики G_1 осуществим за два шага.

Шаг 1. Построим грамматику

$$G' = (N', T, S, P').$$

Рассмотрим все продукции из P вида:

$$A \rightarrow X_1 X_2 \dots X_n, \quad (6)$$

где каждый символ X_i либо терминал из T , либо нетерминал из N . Если $n = 1$, тогда X_1 должен быть терминалом, так как в P нет цепных продукций. В этом случае помещаем эту продукцию в P' . Если $n \geq 2$, то вводим новые нетерминальные символы Z_a

для каждого $X = a$, где $a \in T$. Каждой продукции из P , имеющей вид (6), ставим в соответствие продукцию:

$$A \rightarrow B_1 B_2 \dots B_n,$$

где $B_i = X_i$, если $X_i \in N$, и $B_i = Z_a$, если $X_i = a$ и $a \in T$.

Для каждого нового нетерминального символа Z_a мы формируем продукцию

$$Z_a \rightarrow a$$

и помещаем ее в P' . На этом шаге удаляются все терминалы из продукций, правая часть которых имеет длину больше 1, а вместо них на их места вводятся новые нетерминалы. В результате после завершения данного шага получаем грамматику $G' = (N', T, S, P')$, все продукции которой имеют вид либо

$$A \rightarrow a, a \in T, \tag{7}$$

либо

$$A \rightarrow B_1 B_2 \dots B_n, \tag{8}$$

где $B_i \in N, i = 1, 2, \dots, n$.

По теореме 26 заключаем, что

$$L(G') = L(G).$$

Шаг 2. На втором шаге мы вводим вспомогательные нетерминалы для того, чтобы сократить длину правых частей до 2, если она превышает эту границу.

Сперва помещаем все продукции вида (7) в P_1 , а также помещаем в P_1 все продукции вида (8), правая часть которых имеет длину 2 (то есть когда $n = 2$). Если $n > 2$, вводим новые нетерминалы, ... и добавляем в P' новые продукции:

$$A \rightarrow B_1 B'_1$$

$$B'_1 \rightarrow B_2 B'_2$$

...

$$B'_{n-2} \rightarrow B_{n-1} B_n.$$

Очевидно, полученная в результате этого шага грамматика $G_1 = (N_1, T, S, P_1)$ будет иметь нормальную форму Хомского,

причем, как это следует из теоремы 26, $L(G') = L(G_1)$. Окончательно имеем:

$$L(G) = L(G_1),$$

что и требовалось доказать. *Теорема доказана.*

Пример 40.

Преобразовать грамматику G с продукциями

$$S \rightarrow AVa,$$

$$A \rightarrow aab,$$

$$B \rightarrow Ac$$

к нормальной форме Хомского.

Прежде всего убеждаемся, что G не содержит ни цепных, ни ϵ -продукций. На шаге 1 вводим новые нетерминалы Z_a, Z_b, Z_c и изменяем множество продукций:

$$S \rightarrow ABZ_a,$$

$$A \rightarrow Z_a Z_a Z_b,$$

$$B \rightarrow AZ_c,$$

$$Z_a \rightarrow a,$$

$$Z_b \rightarrow b,$$

$$Z_c \rightarrow c.$$

На втором шаге вводим новые нетерминалы и редуцируем правые части продукций (первой и второй); в результате получаем грамматику в *нормальной форме Хомского с множеством продукций*:

$$S \rightarrow AV' \quad V' \rightarrow BZ_a$$

$$A \rightarrow Z_a \rightarrow Z_a Z_b$$

$$B \rightarrow AZ_c Z_b \rightarrow b$$

$$Z_a \rightarrow aZ_c \rightarrow c.$$

Другой полезной грамматической формой является *нормальная форма Грейбах*. В этом случае ограничение накладывается не на длину правых частей, а на порядок, в котором терминалы и нетерминалы входят в правую часть продукции. Эта форма имеет много теоретических и практических применений, однако процедура преобразования к ней сложнее, чем в случае нормальной формы Хомского, а эквивалентность исходной грамматике не столь очевидна.

Определение 39.

КС-грамматика $G = (N, T, S, P)$ имеет нормальную форму Грейбах, если все ее продукции имеют вид

$$A \rightarrow a\alpha, \quad (9)$$

где $A \in N, a \in T, \alpha \in N^*$.

Заметим, что, в силу определения, грамматика в нормальной форме Грейбах не может быть леворекурсивной.

Теорема 40.

Для любой КС-грамматики $G = (N, T, S, P)$ такой, что $\varepsilon \notin L(G)$, существует эквивалентная нормальная форма Грейбах

$$G_2 = (N_2, T, S, P_2).$$

Доказательство.

Алгоритм преобразования данной грамматики G в нормальную форму Грейбах состоит из нескольких шагов.

Шаг 1. Преобразуем грамматику G к нормальной форме Хомского G_1 .

Шаг 2. Упорядочим и перенумеруем все нетерминалы:

$$A_1, A_2, \dots, A_n$$

так, чтобы $A_1 = S$.

Шаг 3. Используя теоремы 26 и 28, преобразуем грамматику G_1 таким образом, чтобы все продукции были следующих форм:

$$\begin{aligned} A_i &\rightarrow A_j \alpha_j, j > i, \\ Z_i &\rightarrow A_j \alpha_j, j \leq n, \\ A_i &\rightarrow a \alpha_i, \end{aligned} \quad (10)$$

где $a \in T, \alpha_i \in N$, и Z_i – нетерминалы, появившиеся при применении теоремы 28 для устранения левой рекурсии.

Покажем, что это всегда можно сделать. Возьмем продукции из G_1 , левая часть которых A_i . Все такие продукции удовлетворяют условиям (10), за исключением продукций вида

$$A_i \rightarrow A_1 \alpha.$$

Применим теорему 28 к этим продукциям и введем нетерминал Z_1 , исключив эти леворекурсивные продукции.

Теперь рассмотрим продукции вида

$$A_2 \rightarrow A_1 \alpha.$$

Если в правой части, применив теорему 26, заменить A_1 , то получим или допустимую продукцию, удовлетворяющую условиям (10), или продукцию вида

$$A_2 \rightarrow A_2 \alpha.$$

Применяя теорему 28, исключаем этот случай. Для нетерминала A_3 сначала исключаем по теореме 26 случай $A_3 \rightarrow A_1 \alpha$, потом случай $A_3 \rightarrow A_2 \alpha$. А затем по теореме 28 удаляем леворекурсивные A_3 -продукции. И так далее, пока не получим лишь те продукции, которые удовлетворяют условиям (10).

Шаг 4. После шага 3 все A_n -продукции будут иметь вид:

$$A_n \rightarrow a \alpha_n,$$

а значит, будут удовлетворять условиям (9).

Подставим правые части этих продукций в продукции $A_{n-1} \rightarrow A_n \alpha$, получим:

$$A_{n-1} \rightarrow a \alpha_n \alpha,$$

то есть продукции типа (9), затем подставим их правые части вместо нетерминала A_{n-1} в A_{n-2} -продукции и т. д. В конце концов все продукции приобретут требуемый вид (9), а грамматика G_1 трансформируется тем самым в грамматику $G_2 = (N_2, T, S, P_2)$, имеющую нормальную форму Грейбах. Так как все использованные при этом преобразования сохраняли эквивалентность грамматик, то, очевидно, имеем:

$$L(G) = L(G_2).$$

Пример 41.

Найти нормальную форму Грейбах для грамматики

$$A_1 \rightarrow A_2 A_2 \mid a$$

$$A_2 \rightarrow A_1 A_2 \mid b.$$

Данная грамматика уже имеет нормальную форму Хомского, а нетерминалы перенумерованы нужным образом. Поэтому можем начать сразу с шага 3, описанного в теореме 40 алгоритма. Продукции

$$A_2 \rightarrow b \text{ и } A_1 \rightarrow A_2 A_2 \mid a$$

уже в требуемой по условиям (10) форме. Чтобы привести продукцию

$$A_2 \rightarrow A_1 A_2$$

к нормальной форме, используем теорему 26; подставим вместо A_1 строки $A_2 A_2$, a и получим все A_2 -продукции:

$$A_2 \rightarrow A_2 A_2 A_2 \mid a A_2 \mid b.$$

Теперь используем теорему 28 для удаления левой рекурсии. Вводя нетерминал Z и полагая в теореме 28 $A = A_2$, $\alpha_1 = A_2 A_2$, $\beta_1 = a A_2$, $\beta_2 = b$, получим:

$$\begin{aligned} A_2 &\rightarrow a A_2 \mid a A_2 Z \mid b \mid b Z \\ Z &\rightarrow A_2 A_2 \mid A_2 A_2 Z. \end{aligned}$$

На последнем шаге подставляем вместо A_2 в других продукциях правые части A_2 -продукций и получаем грамматику в нормальной форме Грейбах:

$$\begin{aligned} A_1 &\rightarrow a A_2 A_2 \mid a A_2 Z A_2 \mid b A_2 \mid b Z A_2 \mid a; \\ Z &\rightarrow a A_2 A_2 \mid a A_2 Z A_2 \mid b A_2 \mid b Z A_2 \mid a A_2 A_2 Z \mid a A_2 Z A_2 Z \mid b A_2 Z \mid b Z A_2 Z; \\ A_2 &\rightarrow a A_2 \mid a A_2 Z \mid b \mid b Z. \end{aligned}$$

Глава 7. Магазинные автоматы

Описание контекстно-свободных языков посредством контекстно-свободных грамматик хорошо согласуется с использованием нотации Бэкуса-Наура при определении языков программирования. Возникает вопрос, существует ли класс автоматов, соответствующий классу КС-языков. Как мы уже видели, конечные автоматы не могут распознавать КС-языки. Интуитивно можно предположить, что это происходит оттого, что конечные автоматы имеют только строго ограниченную – конечную – память, тогда как распознавание контекстно-свободного языка может потребовать запоминания неограниченного объема информации. Например, когда считывается строка из языка $L = \{a^n b^n \mid n \geq 0\}$, мы должны не только проверить, что все символы a предшествуют первому символу b , но также должны запоминать количество символов a . Так как n не ограничено, то этот подсчет невозможно сделать, располагая лишь конечной памятью. Для этого требуется машина, которая не содержит такого ограничения. Более того, для некоторых языков, таких, как $\{\alpha\alpha^{-1}\}$, требуется нечто большее, нежели просто способность к неограниченному счету, а именно способность запоминать и сравнивать последовательность символов в обратном порядке. Это наводит на мысль использовать в качестве устройства памяти магазин (или стек). Так мы приходим к классу машин, называемых *магазинными автоматами*.

В данной главе мы исследуем связь между магазинными автоматами и контекстно-свободными языками. Вначале покажем, что в недетерминированном варианте класс таких автоматов допускает в точности семейство КС-языков. Но на этом аналогия с детерминированными и недетерминированными версиями конечных автоматов заканчивается: класс детерминированных магазинных автоматов определяет совершенно другой класс языков, отличных от семейства КС-языков, – так называемых *детерминированных КС-языков*, образующих собственное подмножество множества всех КС-языков. Этот класс весьма интересен с точки зрения разработки эффективных методов трансляции языков программирования.

7.1. Недетерминированные магазинные автоматы

Схематически магазинный автомат можно представить так, как это изображено на рис. 32.



Рис. 32. Магазинный автомат

На каждом шаге устройство управления считывает один символ с входной ленты и одновременно изменяет содержимое магазина посредством обычных стековых операций. Каждый следующий шаг, совершаемый устройством управления, определяется данным прочитанным символом с входной ленты, верхним символом стека и внутренним состоянием автомата. Результатом такого шага (или такта работы) является новое состояние устройства управления и замена содержимого вершины стека. Как уже было сказано в главе 2, мы ограничиваемся рассмотрением лишь автоматов-распознавателей. В этом случае формальное определение выглядит следующим образом.

Определение 41.

Недетерминированный магазинный автомат (НМА) определяется как семерка $M = (Q, \Sigma, V, \theta, q_0, z_0, F)$, где

$Q = \{q_0, q_1, \dots, q_n\}$ – конечное множество внутренних состояний автомата;

Σ – конечное множество символов, называемых входным алфавитом;

V – конечное множество символов, называемое стековым алфавитом;

θ – функция переходов автомата, определяемая отображением $\theta: Q \times (\Sigma \cup \{\varepsilon\}) \times V \rightarrow \{S \mid S \text{ – конечное подмножество множества } Q \times V^*\}$;

q_0 – начальное состояние автомата, $q_0 \in Q$;

z_0 – начальный символ стека, $z_0 \in V$;

F – множество заключительных (или финальных) состояний автомата, $F \subseteq Q$.

Функция θ зависит от трех аргументов (q, a, z) , из которых обозначает текущее состояние автомата в данный момент, a – обозреваемый входной символ и z – символ в вершине стека. Значением $\theta(q, a, z)$ является конечное множество пар (q_i, β_i) , где q_i – следующее состояние автомата, а β_i – строка, помещаемая в вершину стека вместо символа a . Заметим, что вторым аргументом у функции θ может быть символ пустой строки ε , означающий, что возможен такт работы автомата, на котором ни один из входных символов не читается. Такой такт называется ε -переходом. В то же время следует отметить, что третий аргумент не может быть пустым, то есть переход невозможен, если стек пуст.

Пример 42.

Предположим, что множество правил перехода НМА содержит правило

$$\theta(q_1, a, b) = \{(q_2, cd), (q_3, \varepsilon)\}.$$

Если в некоторый момент времени автомат, находясь в состоянии q_1 , читает входной символ a и при этом имеет в вершине стека символ b , тогда возможно одно из двух:

– автомат переходит в состояние q_2 , а строка cd заменяет символ в вершине стека (при этом в вершине стека оказывается символ c);

– автомат переходит в состояние q_3 , при этом символ b удаляется из вершины стека.

Пример 43.

Рассмотрим НМА, у которого

$$Q = \{q_0, q_1, q_2, q_3\},$$

$$\Sigma = \{a, b\},$$

$$V = \{0, 1\},$$

$$z_0 = 0,$$

$$F = \{q_3\},$$

а функция θ определяется соотношениями:

$$\theta(q_0, a, 0) = \{(q_1, 10), (q_3, \varepsilon)\},$$

$$\theta(q_0, \varepsilon, 0) = \{(q_3, \varepsilon)\},$$

$$\theta(q_1, a, 1) = \{(q_1, 11)\},$$

$$\theta(q_1, b, 1) = \{(q_2, \varepsilon)\},$$

$$\theta(q_2, b, 1) = \{(q_2, \varepsilon)\},$$

$$\theta(q_2, \varepsilon, 0) = \{(q_3, \varepsilon)\}.$$

Заметим, что функция θ не всюду определена. Так, например, для $(q_0, b, 0)$ соотношение, указывающее значение θ , отсутствует. В этом случае мы интерпретируем значение θ как пустое множество, из которого нет ни одной дуги, означающей переход в другое состояние.

Основными соотношениями для θ являются следующие два: первое – $\theta(q_1, a, 1) = \{(q_1, 11)\}$, которое добавляет 1 в стек, когда прочитан очередной символ a , и второе – $\theta(q_2, b, 1) = \{(q_2, \varepsilon)\}$, которое удаляет 1 из стека, когда во входной строке встречается b .

Эти два шага подсчитывают число символов a и сравнивают с числом символов b . Автомат находится в состоянии q_1 до тех пор, пока не встретит на ленте первый символ b , и тогда переходит в состояние q_2 . Это гарантирует, что ни одно b не встретится раньше последнего a . Анализируя остальные соотношения для θ ,

убеждаемся, что автомат переходит в состояние q_3 тогда и только тогда, когда входная строка имеет вид

$$a^n b^n \text{ или } a,$$

то есть принадлежит языку $L = \{ a^n b^n \mid n \geq 0 \} \cup \{ a \}$.

По аналогии с конечными автоматами мы можем сказать, что НМА допускает язык L .

Пусть q_1 – текущее состояние НМА, $a\alpha$ – непрочитанная часть входной строки, $b\beta$ – содержимое стека с символом b в его вершине. Тогда упорядоченная тройка

$$(q_1, a\alpha, b\beta)$$

называется конфигурацией магазинного автомата. Переход из одной конфигурации в другую с помощью одного такта работы автомата обозначается символом \vdash :

$$(q_1, a\alpha, b\beta) \vdash (q_2, \alpha, \gamma\beta)$$

возможно тогда и только тогда, когда

$$(q_2, \gamma) \in \theta(q_1, a, b).$$

Переходы, включающие произвольное число тактов, будут обозначаться \vdash^* (или \vdash^+ для случая, когда хотя бы один шаг действительно имеет место). Чтобы идентифицировать переход, совершенный данным автоматом M , будем использовать символ \vdash_M .

Определение 42.

Пусть $M = (Q, \Sigma, V, \theta, q_0, z_0, F)$ – недетерминированный магазинный автомат. Тогда язык, *допускаемый автоматом* M , – это множество

$$L(M) = \{ \alpha \mid \alpha \in \Sigma^* \text{ и } (q_0, \alpha, z_0) \vdash_M^* (q_f, \varepsilon, \beta), q_f \in F, \beta \in V^* \}.$$

Таким образом, язык, допускаемый НМА M , это множество строк в алфавите Σ , которые переводят автомат M из начального состояния в финальное состояние после прочтения входной строки (при этом содержимое стека не играет никакой роли и может быть любым).

Пример 44.

Построить НМА для языка

$$L = \{ \alpha \mid \alpha \in \{ a, b \}^* \text{ и } n_a(\alpha) = n_b(\alpha) \},$$

где $n_a(\alpha)$ и $n_b(\alpha)$ обозначают число символов a и b в строке α .

Нетрудно проверить, что НМА

$$M = (\{q_0, q_f\}, \{a, b\}, \{0, 1, z\}, \theta, q_0, z, \{q_f\}),$$

где θ определяется соотношениями:

$$\begin{aligned}\theta(q_0, \varepsilon, z) &= \{(q_f, z)\}, \\ \theta(q_0, a, z) &= \{(q_0, 0z)\}, \\ \theta(q_0, b, z) &= \{(q_0, 1z)\}, \\ \theta(q_0, a, 0) &= \{(q_0, 00)\}, \\ \theta(q_0, b, 0) &= \{(q_0, \varepsilon)\}, \\ \theta(q_0, a, 1) &= \{(q_0, \varepsilon)\}, \\ \theta(q_0, b, 1) &= \{(q_0, 11)\},\end{aligned}$$

действительно допускает язык L . В частности, при чтении строки $baab$ НМА M совершает следующие такты:

$$\begin{aligned}(q_0, baab, z) \vdash (q_0, aab, 1z) \vdash (q_0, ab, z) \vdash (q_0, b, 0z) \vdash \\ \vdash (q_0, \varepsilon, z) \vdash (q_f, \varepsilon, z),\end{aligned}$$

следовательно, эта строка допускается автоматом M .

Пример 45.

Построить НМА, допускающий язык

$$L = \{\alpha\alpha^{-1} \mid \alpha \in \{a, b\}^+\}.$$

Для построения недетерминированного магазинного автомата, распознающего данный язык, воспользуемся тем фактом, что символы извлекаются из стека в порядке, обратном тому, в котором они туда заносились. Во время чтения первой половины входной строки мы помещаем последовательно читаемые символы в стек, а при чтении второй половины строки мы сравниваем текущий входной символ с тем, который находится в вершине стека, продолжая это сравнение до окончания входной строки. Так как символы извлекаются из стека в порядке, обратном тому, в котором они заносились в стек, то полностью сравнение будет успешным тогда и только тогда, когда входная строка имеет вид $\alpha\alpha^{-1}$ для некоторой строки $\alpha \in \{a, b\}^+$.

Очевидной трудностью здесь является то, что мы не знаем, где середина входной строки, то есть где кончается α и начинается α^{-1} . Но недетерминированный характер автомата приходит нам на помощь: НМА правильно указывает, где середина, и сигнализирует своими состояниями об этом. Итак, строим НМА

$$M = (Q, \Sigma, V, \theta, q_0, z_0, F),$$

где $Q = \{q_0, q_1, q_2\}$, $\Sigma = \{a, b\}$, $V = \{a, b, z\}$, $F = \{q_2\}$.

Соотношения, определяющие функцию θ , можно разбить на несколько групп:

а) команды, записывающие α в стек:

$$\theta(q_0, a, a) = \{(q_0, aa)\},$$

$$\theta(q_0, b, a) = \{(q_0, ba)\},$$

$$\theta(q_0, a, b) = \{(q_0, ab)\},$$

$$\theta(q_0, b, b) = \{(q_0, bb)\},$$

$$\theta(q_0, a, z) = \{(q_0, az)\},$$

$$\theta(q_0, b, z) = \{(q_0, bz)\};$$

б) команды, определяющие середину строки, что сигнализируется переходом из состояния q_0 в q_1 :

$$\theta(q_0, \varepsilon, a) = \{(q_1, a)\},$$

$$\theta(q_0, \varepsilon, b) = \{(q_1, b)\},$$

в) команды, сравнивающие α^{-1} с содержимым стека:

$$\theta(q_1, a, a) = \{(q_1, \varepsilon)\},$$

$$\theta(q_1, b, b) = \{(q_1, \varepsilon)\},$$

$$\theta(q_1, \varepsilon, z) = \{(q_2, z)\}.$$

Рассмотрим последовательность тактов при распознавании этим автоматом строки $abba$:

$$(q_0, abba, z) \vdash (q_0, bba, az) \vdash (q_0, ba, baz) \vdash$$

$$\vdash (q_1, ba, baz) \vdash (q_1, a, az) \vdash (q_1, \varepsilon, z) \vdash (q_2, \varepsilon, z).$$

Недетерминированный выбор при определении середины входной строки возникает на третьем такте. На этом шаге НМА имеет конфигурацию (q_0, ba, baz) и две возможности для следующего шага: один из них использует команду

$$\theta(q_0, b, b) = \{(q_0, bb)\}$$

и порождает такт

$$(q_0, ba, baz) \vdash (q_0, a, bbaz),$$

а другой – порождается использованной выше командой

$$\theta(q_0, \varepsilon, b) = \{(q_1, b)\},$$

и только он ведет к распознаванию входной строки.

7.2. Магази́нные автоматы и КС-языки

В этом разделе мы установим непосредственную связь между магазинными автоматами и контекстно-свободными языками, а именно мы покажем, что для каждого КС-языка существует НМА, который допускает его, и, наоборот, любой язык, допускаемый недетерминированным магазинным автоматом, является контекстно-свободным.

Итак, покажем вначале, что для каждого КС-языка существует распознающий его НМА. Основная идея построения такого НМА заключается в том, чтобы он мог каким-то способом воспроизводить левосторонний вывод любой строки данного языка. Для упрощения построения такого автомата будем предполагать, что язык порождается КС-грамматикой в нормальной форме Грейбах. Магази́нный автомат будет производить вывод, запоминая переменные правой части сентенциальной формы в стеке, тогда как левая часть, целиком состоящая из терминалов, должна быть идентична прочитанной части входной строки. Начинаем с занесения в стек начального символа, после чего для моделирования применения некоторой продукции $A \rightarrow a\alpha$ нам нужно иметь переменную A в вершине стека, а терминал a – в качестве читаемого входного символа. Затем переменная из вершины стека удаляется и заменяется на строку нетерминальных символов α . Нетрудно понять, каковы должны быть при этом команды, определяющие функцию θ .

Пример 46.

Построить автомат, допускающий язык, порожденный грамматикой с продукциями

$$S \rightarrow aSbb \mid a.$$

Преобразуем грамматику к нормальной форме Грейбах:

$$S \rightarrow aSA \mid a,$$

$$A \rightarrow bB,$$

$$B \rightarrow b.$$

Соответствующий автомат будет иметь три состояния $\{q_0, q_1, q_2\}$, q_0 – начальное, q_2 – финальное состояние. Вначале заносим в стек начальный символ грамматики S командой

$$\theta(q_0, \varepsilon, z) = \{(q_1, Sz)\}.$$

Продукция $S \rightarrow aSA$ моделируется автоматом посредством извлечения S из стека и заменой на SA , когда читается входной символ a . Аналогично продукция $S \rightarrow a$ соответствует извлечению символа S из стека в результате чтения символа a из входной строки. Таким образом, обе эти продукции представляются в НМА соотношением

$$\theta(q_1, a, S) = \{(q_1, SA), (q_1, \varepsilon)\}.$$

Аналогичным образом остальным продукциям соответствуют команды:

$$\begin{aligned}\theta(q_1, b, A) &= \{(q_1, B)\}, \\ \theta(q_1, b, B) &= \{(q_1, \varepsilon)\}.\end{aligned}$$

Появление в вершине стека начального символа стека z означает завершение вывода, и НМА переходит в заключительное состояние:

$$\theta(q_1, \varepsilon, z) = \{(q_2, \varepsilon)\}.$$

Теорема 43.

Для любого контекстно-свободного языка L , не содержащего ε , существует НМА M такой, что

$$L = L(M).$$

Доказательство.

Если L – ε -свободный КС-язык, то существует КС-грамматика в нормальной форме Грейбах $G = (N, T, S, P)$, порождающая L . Построим магазинный автомат, моделирующий левосторонний вывод в этой грамматике. Положим

$$M = (\{q_0, q_1, q_f\}, T, N \cup \{z_0\}, \theta, q_0, z_0, \{q_f\}),$$

где $z_0 \notin N$. Таким образом, видим, что входной алфавит автомата совпадает с множеством терминалов грамматике G , стековый алфавит содержит множество всех нетерминалов из G .

Функция переходов θ будет содержать соотношение

$$\theta(q_0, \varepsilon, z_0) = \{(q_1, Sz_0)\}; \quad (11)$$

таким образом, после первого шага автомата M стек будет содержать стартовый символ S вывода. (Стековый начальный символ z_0 будет служить нам для сигнализации окончания вывода.)

Кроме того, множество правил перехода будет обладать следующим свойством: для любой продукции $A \rightarrow aa$ из P

$$(q_1, \alpha) \in \theta(q_1, a, A). \quad (12)$$

По этому соотношению НМА M читает входной символ a , удаляет переменную A из стека и заменяет ее на α . Это и дает нам переходы, которые позволяют НМА моделировать выводы в G . Наконец, добавим соотношение

$$\theta(q_1, \varepsilon, z_0) = \{(q_p, z_0)\}, \quad (13)$$

которое переводит M в заключительное состояние.

Чтобы показать, что M допускает любую строку $\alpha \in L(G)$, рассмотрим частичный левосторонний вывод

$$S \Rightarrow^* a_1 a_2 \dots a_n A_1 A_2 \dots A_m \Rightarrow a_1 a_2 \dots a_n b B_1 B_2 \dots B_k A_2 \dots A_m.$$

Так как M моделирует этот вывод, то после прочтения строки $a_1 a_2 \dots a_n$ стек должен содержать $A_1 A_2 \dots A_m$. Для построения следующего шага вывода в G должна иметься продукция

$$A_1 \rightarrow b B_1 \dots B_k.$$

Но конструкция M такова, что M должен иметь правило перехода, по которому

$$(q_1, B_1 \dots B_k) \in \theta(q_1, b, A_1),$$

так что стек после прочтения части $a_1 a_2 \dots a_n b$ входа должен теперь содержать

$$B_1 \dots B_k A_2 \dots A_m.$$

Индукцией по длине вывода убеждаемся, что если

$$S \Rightarrow^* \alpha,$$

то

$$(q_1, \alpha, Sz_0) \vdash^* (q_1, \varepsilon, z_0).$$

С учетом (11) и (13), получаем:

$$(q_0, \alpha, z_0) \vdash (q_1, \alpha, Sz_0) \vdash^* (q_1, \varepsilon, z_0) \vdash (q_p, \varepsilon, z_0),$$

следовательно,

$$L(G) \subseteq L(M).$$

Покажем теперь, что $L(M) \subseteq L(G)$.

Пусть $\alpha \in L(M)$, тогда, по определению,

$$(q_0, \alpha, z_0) \vdash^* (q_p, \varepsilon, \varphi).$$

Нетрудно видеть, что для перехода из q_0 в q_1 существует лишь единственный путь, как и для перехода из q_1 в q_p . Поэтому верно соотношение

$$(q_1, \alpha, Sz_0) \vdash^* (q_1, \varepsilon, z_0).$$

Возьмем $\alpha = a_1 a_2 a_3 \dots a_n$. Тогда в последовательности

$$(q_1, a_1 a_2 a_3 \dots a_n, Sz_0) \vdash^* (q_1, \varepsilon, z_0) \quad (14)$$

первый шаг, очевидно, должен совершаться по правилу типа (12), что дает

$$(q_1, a_1 a_2 a_3 \dots a_n, Sz_0) \vdash (q_1, a_2 a_3 \dots a_n, \varphi_1 z_0).$$

Но это означает, что грамматика G должна иметь продукцию вида

$$S \rightarrow a_1 \varphi_1,$$

что позволяет записать

$$S \Rightarrow a_1 \varphi_1.$$

Далее, полагая $\varphi_1 = A\varphi_2$, получим

$$(q_1, a_2 a_3 \dots a_n, A\varphi_2 z_0) \vdash (q_1, a_3 \dots a_n, \varphi_3 \varphi_2 z_0),$$

откуда заключаем, что грамматика G должна содержать продукцию $A \rightarrow a_2 \varphi_3$, значит,

$$S \Rightarrow^* a_1 a_2 \varphi_3 \varphi_2.$$

Отсюда несложно видеть, что на каждом шаге содержимое стека (исключая z_0) совпадает с еще не замещенной терминалами частью сентенциальной формы. С учетом (14) получаем

$$S \Rightarrow^* a_1 a_2 \dots a_n.$$

Следовательно, $L(M) \subseteq L(G)$, и *теорема доказана*.

Пример 47.

По грамматике

$$S \rightarrow aA,$$

$$A \rightarrow aABC|bB|a,$$

$$B \rightarrow b,$$

$$C \rightarrow c$$

построить НМА, допускающий язык $L(G)$.

Так как грамматика уже в нормальной форме Грейбах, переходим сразу к построению НМА:

$$\theta(q_0, \varepsilon, z_0) = \{(q_1, Sz_0)\},$$

$$\theta(q_1, \varepsilon, z_0) = \{(q_1, z_0)\},$$

$$\theta(q_1, a, S) = \{(q_1, A)\},$$

$$\theta(q_1, a, A) = \{(q_1, ABC), (q_1, \varepsilon)\},$$

$$\theta(q_1, b, A) = \{(q_1, B)\},$$

$$\theta(q_1, b, B) = \{(q_1, \varepsilon)\},$$

$$\theta(q_1, c, C) = \{(q_1, \varepsilon)\}.$$

Рассмотрим шаги автомата на входной строке $aaabc$:

$$\begin{aligned} &(q_0, aaabc, z_0) \vdash (q_1, aaabc, Sz_0) \vdash (q_1, aabc, Az_0) \vdash \\ &\vdash (q_1, abc, ABCz_0) \vdash (q_1, bc, BCz_0) \vdash (q_1, c, Cz_0) \vdash \\ &\vdash (q_1, \varepsilon, z_0) \vdash (q_f, \varepsilon, z_0). \end{aligned}$$

Эта последовательность шагов соответствует выводу:

$$S \Rightarrow aA \Rightarrow aaABC \Rightarrow aaaBC \Rightarrow aaabC \Rightarrow aaabc.$$

Теперь нашей задачей будет доказательство обратного по отношению к теореме 34 утверждения. Для этого надо по данному НМА построить грамматику G , моделирующую такты автомата. Это означает, что содержимое стека должно изображаться той частью сентенциальной формы, которая состоит из нетерминальных символов, в то время как прочитанная часть входной строки должна представлять собой терминальный префикс сентенциальной формы.

Для того чтобы избавить рассуждения от излишних деталей, мы потребуем, чтобы НМА удовлетворял следующим условиям:

1. НМА имеет единственное финальное состояние, которое достигается тогда и только тогда, когда стек полностью пуст;
2. Все переходы должны иметь вид:

$$\theta(q_r, a, A) = \{k_1, k_2, \dots, k_n\},$$

где для любого $r = 1, 2, \dots, n$

$$k_r = (q_j, \varepsilon) \tag{15}$$

или

$$k_r = (q_j, BC). \tag{16}$$

Таким образом, каждый такт автомата увеличивает или уменьшает содержимое стека ровно на 1 символ. Нетрудно убедиться, что для любого НМА существует эквивалентный ему другой НМА, удовлетворяющий этим двум условиям. Проверку этого оставляем читателю в качестве несложного упражнения.

Считая условия 1 и 2 вспомогательными, построим КС-грамматику для языка, допускаемого таким НМА.

Как уже говорилось, мы хотим, чтобы сентенциальная форма отражала содержимое стека. Кроме того, конфигурация автомата содержит символ внутреннего состояния, который также надо отразить в сентенциальной форме. Для этого нетерминальные

символы в грамматике G будем представлять в форме $(q_i A q_j)$, интерпретируя это следующим образом:

$$(q_i A q_j) \alpha$$

тогда и только тогда, когда НМА стирает A в стеке и в результате чтения входа α переходит из состояния q_i в состояние q_j . «Стирание» означает, что A удаляется из стека и на его место ничего не записывается, то есть в вершине стека оказывается символ, непосредственно находившийся под самым верхним символом A .

Используя эту интерпретацию, нетрудно видеть, что продукции грамматики с необходимостью должны соответствовать одному из двух типов переходов. Так как (15) влечет немедленное стирание A (в стеке), то грамматика будет иметь соответствующую продукцию

$$(q_i A q_j) \rightarrow a.$$

Переходы типа (16) порождают продукцию вида

$$(q_i A q_k) \rightarrow a(q_j B q_l)(q_t C q_k),$$

где q_k и q_l могут принимать любые значения из Q . Это объясняется тем, что для стирания A мы сначала заменяем его на BC в результате чтения a и переходим из состояния q_i в q_j , а затем последовательно переходим из q_j в q_l , стирая B , а потом – из q_t в q_k , стирая из стека C .

Наконец, в качестве начального символа грамматики возьмем $(q_0 z_0 q_f)$, где q_f – единственное заключительное состояние нашего НМА.

Теорема 44.

Если язык L допускается некоторым НМА M , тогда L является контекстно-свободным.

Доказательство.

Предположим, язык L допускается недетерминированным магазинным автоматом

$$M = (Q, \Sigma, V, \theta, q_0, z_0, \{q_f\}),$$

удовлетворяющим условиям 1 и 2, указанным выше. Построим для L грамматику $G = (N, T, S, P)$, где $T = \Sigma$, а N состоит из элементов вида $(q_i A q_j)$. Используем описанную ранее конструкцию

и покажем, что полученная грамматика такова, что для всех $q_i, q_j \in Q; A \in V; \varphi \in V^*; \alpha, \beta \in \Sigma^*$ соотношение

$$(q_i, \alpha\beta, A\varphi) \vdash^* (q_j, \beta, \varphi), \quad (17)$$

влечет

$$(q_i A q_j) \alpha,$$

и наоборот.

Итак, сначала нам надо показать следующее: если НМА таков, что символ A и его последователи могут быть удалены из стека в результате чтения строки α и перехода из состояния q_i в q_j , тогда α может быть выведена из нетерминала $(q_i A q_j)$ в грамматике G . Это нетрудно сделать, потому что грамматика G как раз и была построена так, чтобы это имело место. Достаточно провести простое рассуждение индукцией по числу тактов работы НМА, чтобы убедиться в этом.

Для доказательства обратного утверждения рассмотрим отдельный шаг вывода в G следующего вида:

$$(q_i A q_k) \Rightarrow a(q_j B q_i) (q_i C q_k).$$

Используя соответствующий переход для НМА

$$\theta(q_i, a, A) = \{(q_j, BC), \dots\}, \quad (18)$$

видим, что символ A в результате чтения a может быть удален из стека, а BC занесена в него с изменением состояния НМА с q_i на q_j .

Аналогично, если

$$(q_i A q_j) \Rightarrow a, \quad (19)$$

то должен существовать соответствующий переход

$$\theta(q_i, a, A) = \{(q_j, \varepsilon)\}, \quad (20)$$

по которому A может быть удален из стека.

Отсюда видно, что сентенциальные формы, выводимые из $(q_i A q_j)$, определяют последовательность возможных конфигураций НМА, которая соответствует (17). Заметим, что шаг

$$(q_i A q_j) \Rightarrow a(q_j B q_i) (q_i C q_k)$$

возможен для некоторых $(q_j B q_i)$, $(q_i C q_k)$, для которых нет соответствующих переходов вида (18) или (20). Но в этом случае, по меньшей мере, одна из этих переменных будет

несущественной и не будет оказывать влияние на язык, порождаемый грамматикой G . Но для всех сентенциальных форм, порождающих терминальную строку, приведенные рассуждения справедливы.

Если применить полученный результат к последовательности

$$(q_0, \alpha, z_0) \vdash^* (q_f, \varepsilon, \varepsilon),$$

то замечаем, что это возможно тогда и только тогда, когда

$$(q_0 z_0 q_f) \Rightarrow^* \alpha.$$

Следовательно, $L(M) = L(G)$, что и требовалось доказать.

Теорема доказана.

Пример 48.

Рассмотрим НМА со следующими переходами:

$$\theta(q_0, a, z) = \{(q_0, Az)\},$$

$$\theta(q_1, a, A) = \{(q_0, A)\},$$

$$\theta(q_0, b, A) = \{(q_1, \varepsilon)\},$$

$$\theta(q_1, \varepsilon, z) = \{(q_2, \varepsilon)\},$$

где q_0 обозначает, как обычно, начальное состояние, а q_2 – заключительное. Построим соответствующую грамматику G . Видим, что НМА удовлетворяет условию 1, но не удовлетворяет условию 2. Чтобы выполнялось условие 2, введем новое дополнительное состояние q_3 и промежуточный шаг, на котором мы вначале удаляем A из стека, а затем вновь помещаем его туда на следующем шаге. Получаем, таким образом, новую совокупность переходов:

$$\theta(q_0, a, z) = \{(q_0, Az)\},$$

$$\theta(q_3, \varepsilon, z) = \{(q_0, Az)\},$$

$$\theta(q_0, a, A) = \{(q_3, \varepsilon)\},$$

$$\theta(q_0, b, A) = \{(q_1, \varepsilon)\},$$

$$\theta(q_1, \varepsilon, z) = \{(q_2, \varepsilon)\}.$$

Последние три перехода имеют вид (15), поэтому им соответствуют продукции:

$$(q_0 A q_3) \rightarrow a,$$

$$(q_0 A q_1) \rightarrow b,$$

$$(q_1 z q_2) \rightarrow \varepsilon,$$

а двум первым переходам сопоставим продукции:

$$\begin{aligned}
& (q_0zq_0) \rightarrow a(q_0Aq_0)(q_0zq_0) \mid a(q_0Aq_1)(q_1zq_0) \mid a(q_0Aq_2)(q_2zq_0) \mid \\
& a(q_0Aq_3)(q_3zq_0), \\
& (q_0zq_1) \rightarrow a(q_0Aq_0)(q_0zq_1) \mid a(q_0Aq_1)(q_1zq_1) \mid a(q_0Aq_2)(q_2zq_1) \mid \\
& a(q_0Aq_3)(q_3zq_1), \\
& (q_0zq_2) \rightarrow a(q_0Aq_0)(q_0zq_2) \mid a(q_0Aq_1)(q_1zq_2) \mid a(q_0Aq_2)(q_2zq_2) \mid \\
& a(q_0Aq_3)(q_3zq_2), \\
& (q_0zq_3) \rightarrow a(q_0Aq_0)(q_0zq_3) \mid a(q_0Aq_1)(q_1zq_3) \mid a(q_0Aq_2)(q_2zq_3) \mid \\
& a(q_0Aq_3)(q_3zq_3), \\
& (q_3zq_0) \rightarrow (q_0Aq_0)(q_0zq_0) \mid (q_0Aq_1)(q_1zq_0) \mid (q_0Aq_2)(q_2zq_0) \mid \\
& (q_0Aq_3)(q_3zq_0), \\
& (q_3zq_1) \rightarrow (q_0Aq_0)(q_0zq_1) \mid (q_0Aq_1)(q_1zq_1) \mid (q_0Aq_2)(q_2zq_1) \mid \\
& (q_0Aq_3)(q_3zq_1), \\
& (q_3zq_2) \rightarrow (q_0Aq_0)(q_0zq_2) \mid (q_0Aq_1)(q_1zq_2) \mid (q_0Aq_2)(q_2zq_2) \mid \\
& (q_0Aq_3)(q_3zq_2), \\
& (q_3zq_3) \rightarrow (q_0Aq_0)(q_0zq_3) \mid (q_0Aq_1)(q_1zq_3) \mid (q_0Aq_2)(q_2zq_3) \mid \\
& (q_0Aq_3)(q_3zq_3).
\end{aligned}$$

Начальным символом грамматики будет (q_0, z, q_2) .

Возьмем строку aab , которая допускается данным НМА в соответствии с последовательностью конфигураций:

$$\begin{aligned}
& (q_0, aab, z) \vdash (q_0, ab, Az) \vdash (q_3, b, z) \vdash (q_0, b, Az) \\
& \vdash (q_1, \varepsilon, z) \vdash (q_2, \varepsilon, \varepsilon).
\end{aligned}$$

Соответствующий вывод в G будет иметь следующий вид:

$$\begin{aligned}
& (q_0zq_2) \Rightarrow a(q_0Aq_3)(q_3zq_2) \Rightarrow aa(q_3zq_2) \Rightarrow aa(q_0Aq_1) \\
& (q_1zq_2) \Rightarrow \Rightarrow aab(q_1zq_2) \Rightarrow aab.
\end{aligned}$$

7.3. Детерминированные магазинные автоматы и детерминированные КС-языки

Если НМА таков, что каждый его шаг определен однозначно (то есть отсутствует выбор при переходе из одной конфигурации в другую), то такой автомат называется детерминированным магазинным автоматом-распознавателем (ДМА). Дадим определение ДМА, основываясь на определении 28.

Определение 45.

Магазинный автомат

$$M = (Q, \Sigma, V, \theta, q_0, z_0, F)$$

называется детерминированным, если он является НМА в соответствии с определением 28 и удовлетворяет следующим ограничениям:

для любых $q \in Q$, $a \in \Sigma \cup \{\varepsilon\}$ и $b \in V$

1) $|\theta(q, a, b)| \leq 1$, то есть $\theta(q, a, b)$ содержит не более одного элемента,

2) если $\theta(q, \varepsilon, b) \neq \emptyset$, то для любого $a \in \Sigma$
 $\theta(q, a, b) = \emptyset$.

Первое из этих условий означает, что для любого входного символа и любого символа в вершине стека не может существовать более одного шага, а второе условие говорит о том, что когда в некоторой конфигурации возможен ε -переход, то никакой другой переход, сопровождающийся чтением входного символа, уже невозможен.

Определение 46.

Язык L называется детерминированным контекстно-свободным языком (ДКС-языком), если существует ДМА M такой, что $L = L(M)$.

Пример 49.

Покажем, что язык

$$L = \{a^n b^n \mid n \geq 0\}$$

является детерминированным контекстно-свободным языком.

Нетрудно убедиться, что

$$M = (\{q_0, q_1, q_2\}, \{a, b\}, \{0, 1\}, \theta, q_0, 0, \{q_0\}),$$

где θ определена соотношениями

$$\theta(q_0, a, 0) = \{(q_1, 10)\},$$

$$\theta(q_1, a, 1) = \{(q_1, 11)\},$$

$$\theta(q_1, b, 1) = \{(q_2, \varepsilon)\},$$

$$\theta(q_2, b, 1) = \{(q_2, \varepsilon)\},$$

$$\theta(q_2, \varepsilon, 0) = \{(q_0, \varepsilon)\},$$

является ДМА и допускает язык L .

В отличие от конечных автоматов, детерминированные и недетерминированные магазинные автоматы не эквивалентны. Существуют контекстно-свободные языки, которые не являются детерминированными.

Пример 50.

Возьмем два языка:

$$L_1 = \{a^n b^n \mid n \geq 0\}$$

и

$$L_2 = \{a^n b^{2n} \mid n \geq 0\}.$$

Как ранее было показано, язык L_1 контекстно-свободный. Несложной модификацией грамматики языка L_1 можно получить КС-грамматику для L_2 , значит, язык L_2 – тоже КС-язык.

Покажем, что язык

$$L = L_1 \cup L_2$$

является контекстно-свободным. Это можно сделать разными способами. Пусть, скажем,

$$G_1 = (N_1, T, S_1, P_1)$$

и

$$G_2 = (N_2, T, S_2, P_2)$$

– две КС-грамматики, порождающие языки L_1 и L_2 , соответственно.

Предположим, что $N_1 \cap N_2 = \emptyset$ и $S \notin N_1 \cup N_2$. Построим грамматику G , являющуюся комбинацией грамматик G_1 и G_2 :

$$G = (N_1 \cup N_2 \cup \{S\}, T, S, P),$$

где

$$P = P_1 \cup P_2 \cup \{S \rightarrow S_1, S \rightarrow S_2\}.$$

Нетрудно показать, что G порождает язык L , а следовательно, L – контекстно-свободный язык. Покажем, что L не является детерминированным КС-языком. Рассуждение построим следующим образом: предположив, что L – детерминированный КС-язык, придём к заключению, что тогда язык

$$L' = L \cup \{a^n b^n c^n \mid n \geq 0\}$$

должен быть контекстно-свободным, а это не так!

Для доказательства того, что L' будет КС-языком, если L – детерминированный КС-язык, построим НМА M' , распознающий L' , основываясь на ДМА M , распознающем L .

Идея построения этого НМА заключается в том, чтобы добавить к устройству управления автомата М аналогичный блок, в котором переходы, осуществляемые при чтении входного символа b , заменены на аналогичные переходы для символа c . Этот новый блок устройства управления вступает в работу после того, как М заканчивает чтение $a^n b^n$. Так как этот второй блок реагирует на c^n совершенно так же, как первый (основной) блок на b^n , то процесс распознавания $a^n b^{2n}$ превращается теперь в процесс распознавания строки $a^n b^n c^n$. На рис. 33 эта конструкция изображена схематически.

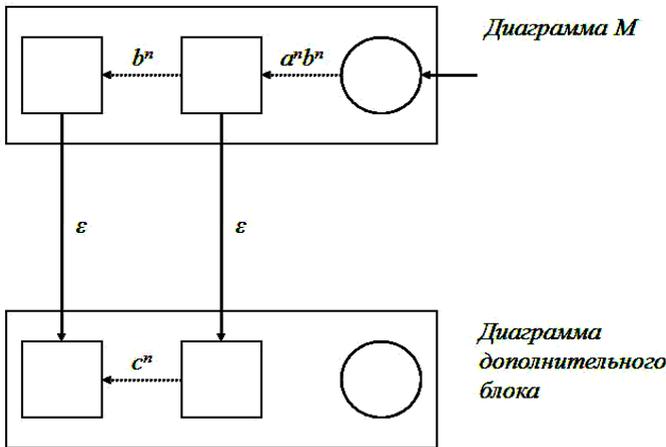


Рис. 33. Диаграмма переходов M'

Итак, пусть $M = (Q, \Sigma, V, \theta, q_0, z_0, F)$, где $Q = \{q_0, q_1, \dots, q_n\}$.

Рассмотрим $M' = (Q', \Sigma, V, \theta \cup \theta', q_0, z_0, F')$, где

$$Q' = Q \cup \{q'_0, q'_1, \dots, q'_n\}, F' = F \cup \{q'_i \mid q_i \in F\},$$

а функция θ' построена из θ добавлением соотношений

$$\theta'(q_j, \varepsilon, x) = \{(q'_j, x)\}$$

для всех $q_j \in F, x \in V$ и

$$\theta'(q'_i, c, x) = \{(q'_i, \varphi)\}$$

для всех $\theta(q_i, b, x) = \{(q_i, \varphi)\}$ и $q_i \in Q, x \in V, \varphi \in V^*$.

Для M распознать строку $a^n b^n$ означает выполнение соотношения

$$(q_0, a^n b^n, z_0)(q_i, \varepsilon, \varphi), \text{ где } q_i \in F.$$

Так как M – детерминированный, должно также выполняться соотношение

$$(q_0, a^n b^{2n}, z_0)(q_j, b^n, \varphi),$$

откуда следует, что для распознавания строки $a^n b^{2n}$ необходимо выполнение соотношения

$$(q_i, b^n, \varphi)(q_j, \varepsilon, \psi),$$

где q_j – некоторое заключительное состояние из F . Но тогда, по построению автомата M' , мы имеем:

$$(q'_i, c^n, \varphi)(q'_j, \varepsilon, \psi),$$

то есть M' допускает строку $a^n b^n c^n$. Нетрудно видеть, что никаких других строк, кроме тех, которые содержатся в L' , НМА M' не допускает. Следовательно, $L' = L(M')$, а это значит, что L' – контекстно-свободный язык. Но это не так!

В следующей главе, применяя Лемму о расширении, мы покажем (пример 51), что L' не является КС-языком. Поэтому наше исходное предположение о том, что L – детерминированный КС-язык, неверно.

Тем самым мы показали, что класс детерминированных КС-языков является собственным подмножеством класса всех КС-языков.

Глава 8. Свойства контекстно-свободных языков

Класс контекстно-свободных языков занимает центральное положение в иерархии формальных языков. С одной стороны, этот класс включает в себя важные, но ограниченные семейства языков, такие как регулярные языки и детерминированные контекстно-свободные языки. С другой стороны, существуют более широкие, нежели класс КС-языков, семейства языков, включающие в себя этот класс. Чтобы изучить соотношения между различными семействами языков, мы исследуем характеристические свойства различных классов языков, в частности, замкнутость относительно тех или иных операций, разрешимость ряда алгоритмических проблем и структурные результаты, к которым относится Лемма о расширении.

8.1. Лемма о расширении

В главе 4 мы доказали Лемму о расширении регулярных языков, которая является эффективным средством для доказательства того, что некоторый язык не регулярен. Подобные леммы о расширении известны и для других классов языков.

Теорема 47. (Лемма о расширении для КС-языков)

Пусть L – бесконечный КС-язык. Тогда существует некоторое целое число $m > 0$ такое, что для любой строки $\alpha \in L$ из того, что $|\alpha| \geq m$, следует, что строка α может быть представлена в следующем виде:

$$\alpha = \beta\gamma\delta\phi\psi, \quad (21)$$

$$\text{где } |\gamma\delta\phi| \leq m, \quad (22)$$

$$|\gamma\phi| \geq 1, \quad (23)$$

и так, что для любого $i = 0, 1, 2, \dots$

$$\beta\gamma^i\delta\phi^i\psi \in L. \quad (24)$$

Доказательство.

Рассмотрим ε -свободный язык L , порождаемый грамматикой G без ε -продукций и без цепных продукций. Так как длины строк в правой части продукций ограничены в совокупности некоторым числом $k > 0$, то длина вывода любой строки $\alpha \in L$ должна

быть не меньше, а так как язык L бесконечен, то, следовательно, в G существуют сколь угодно длинные выводы.

Рассмотрим левосторонний вывод достаточно длинной строки из L . Так как число переменных в грамматике G ограничено, а длина вывода – нет, то, по крайней мере, одна переменная, скажем A , должна встретиться дважды в позиции самого левого нетерминального символа сентенциальной формы. Таким образом, в G должен существовать вывод

$$S \Rightarrow^* \beta A \varphi_1 \Rightarrow^* \beta \gamma A \varphi_2 \varphi_1 \Rightarrow^* \beta \gamma \delta \varphi_2 \varphi_1, \quad (25)$$

где $\beta, \gamma, \delta \in T^*$; $\varphi_1, \varphi_2 \in (N \cup T)^*$.

Будем считать, что A является «самой внутренней» повторяющейся левой переменной в данном выводе, подразумевая под этим тот факт, что в выводе

$$A \Rightarrow^* \delta$$

уже нет повторяющихся самых левых переменных. Легко видеть, что если в выводе имеются несколько повторяющихся «левых» переменных, то всегда среди них можно выбрать одну, удовлетворяющую указанному свойству. В силу нашего предположения о том, что G не содержит ни ε -продукций, ни цепных продукций, ясно, что каждый шаг вывода или порождает новый терминальный символ в сентенциальной форме, или увеличивает ее длину. Отсюда следует, что

$$|\gamma \varphi_2| \geq 1.$$

Анализируя структуру (25), замечаем, что выводы

$$A \Rightarrow^* \gamma A \varphi_2 \text{ и } A \Rightarrow^* \delta$$

возможны в G . А это, в свою очередь, показывает, что в G возможны также выводы

$$S \Rightarrow^* \beta \delta \varphi_1$$

и

$$S \Rightarrow^* \beta \gamma^i \delta \varphi_2^i \varphi_1, \quad i = 1, 2, \dots \quad (26)$$

Вывод (25) заканчивается тогда, когда сентенциальная форма превращается в строку терминалов, а это означает, что существуют выводы

$$\begin{aligned} \varphi_2 &\rightarrow \varphi, \\ \varphi_1 &\rightarrow \psi, \end{aligned}$$

где $\varphi, \psi \in T^*$.

Поэтому из того, что $\alpha = \beta\gamma\delta\phi\psi$, и из (26) получаем, что $\beta\gamma^i\delta\phi^i\psi \in L$.

Так как $|\gamma\phi_2| \geq 1$, то отсюда следует выполнение условия (23). Покажем, что (22) также имеет место. Выберем такое вхождение A в (25), для которого в выводе $A \Rightarrow^*\delta$ отсутствуют повторяющиеся самые левые переменные. Следовательно, $|\delta|$ может быть ограничена константой независимо от строки α , откуда заключаем, что (22) может быть выполнено. Теорема доказана.

Эта Лемма о расширении полезна для доказательства того, что тот или иной язык не принадлежит классу контекстно-свободных языков. В ней сформулировано необходимое условие принадлежности этому классу, в силу чего эта лемма чаще всего используется в негативном смысле: если для некоторого языка утверждение леммы неверно, то это означает, что этот язык не контекстно-свободный.

Пример 51.

Показать, что язык

$$L = \{a^n b^n c^n \mid n \geq 0\}$$

не является контекстно-свободным.

Допустим, что язык L контекстно-свободный, и покажем, что это предположение приводит к противоречию. По теореме 47 для КС-языка должно существовать натуральное число m с указанными в теореме свойствами. Возьмем строку

$$a^m b^m c^m \in L.$$

Рассмотрим возможность выбора в ней подстроки $\gamma\delta\phi$. Очевидно, она не может целиком входить ни в подстроку a^m , ни в подстроку b^m , ни в c^m , так как итерация γ^i и ϕ^i сразу нарушает необходимую структуру строки из L . По тем же причинам невозможно выбрать подстроку $\gamma\delta\phi$ так, чтобы она содержала одинаковое количество символов a и b . Значит, остается единственный способ выбора такой подстроки, когда $\gamma\phi$ содержит одинаковое число символов a , b и c . Но это возможно, лишь когда b^m целиком входит в $\gamma\delta\phi$, откуда следует, что

$$|\gamma\delta\phi| > m,$$

а это противоречит условию (22). Итак, оказывается, что выбрать подстроку $\gamma\delta\phi$ в строке $a^m b^m c^m$ невозможно, следовательно, язык L не контекстно-свободный.

Пример 52.

Рассмотрим язык

$$L = \{\alpha\alpha \mid \alpha \in \{a, b\}^*\}.$$

Покажем, что L не является контекстно-свободным.

Рассмотрим строку

$$a^m b^m a^m b^m \in L.$$

Нетрудно видеть, что при любом представлении этой строки в виде

$$\beta\gamma\delta\varphi\psi,$$

как только мы начинаем итерировать γ и φ , так сразу нарушаем структуру исходной строки и видим тем самым, что существует i такое, что

$$\beta\gamma^i\delta\varphi^i\psi \notin L.$$

На основании этого заключаем, что L не может быть КС-языком.

**8.2. Свойства замкнутости
класса контекстно-свободных языков**

Теорема 48.

Класс контекстно-свободных языков замкнут относительно объединения, сцепления и итерации языков.

Доказательство.

Пусть L_1 и L_2 – КС-языки, порождаемые КС-грамматиками

$$G_1 = (N_1, T_1, S_1, P_1)$$

и

$$G_2 = (N_2, T_2, S_2, P_2)$$

соответственно. Не уменьшая общности, можно предположить, что множества N_1 и N_2 не пересекаются.

Рассмотрим язык $L(G_3)$, порожденный грамматикой

$$G_3 = (N_1 \cup N_2 \cup \{S_3\}, T_1 \cup T_2, S_3, P_3),$$

где $S_3 \notin N_1 \cup N_2$, $P_3 = P_1 \cup P_2 \cup \{S_3 \rightarrow S_1, S_3 \rightarrow S_2\}$. Очевидно, G_3 – КС-грамматика, а, значит, $L(G_3)$ – КС-язык. В то же время легко видеть, что

$$L(G_3) = L_1 \cup L_2.$$

Действительно, пусть $\alpha \in L_1$. Тогда в G_3 возможен вывод

$$S_3 \Rightarrow S_1 \Rightarrow^* \alpha,$$

то есть $\alpha \in L(G_3)$. Для случая, когда $\alpha \in L_2$, рассуждение аналогично.

Обратно, если $\alpha \in L(G_3)$, то первым шагом в выводе $S_3 \Rightarrow^* \alpha$ должен быть либо

$$S_3 \Rightarrow S_1,$$

либо

$$S_3 \Rightarrow S_2.$$

Предположим, что имеет место шаг

$$S_3 \Rightarrow S_1.$$

Так как сентенциальная форма, выводимая из S_1 , содержит переменные только из N_1 и при этом N_1 и N_2 не пересекаются, то вывод

$$S_1 \Rightarrow^* \alpha$$

может использовать лишь продукции из P_1 . Следовательно, $\alpha \in L_1$. Аналогичные рассуждения справедливы для второго случая $S_3 \Rightarrow S_2$. Таким образом, из того, что $\alpha \in L(G_3)$, следует, что

$$\alpha \in L_1 \cup L_2.$$

Далее рассмотрим сцепление двух языков. Положим

$$G_4 = (N_1 \cup N_2 \cup \{S_4\}, T_1 \cup T_2, S_4, P_4).$$

Здесь опять S_4 – новый начальный символ и

$$P_4 = P_1 \cup P_2 \cup \{S_4 \rightarrow S_1 S_2\}.$$

Тогда легко видеть, что

$$L(G_4) = L_1 \cdot L_2.$$

Наконец, рассмотрим итерацию L_1^* языка L_1 . Положим

$$G_5 = (N_1 \cup \{S_5\}, T_1, S_5, P_5),$$

где S_5 – новая переменная, а

$$P_5 = P_1 \cup \{S_5 \rightarrow S_1 S_5, S_5 \rightarrow \varepsilon\}.$$

Тогда

$$L(G_5) = L_1^*.$$

Теорема доказана.

Теорема 49.

Класс КС-языков не замкнут относительно пересечения и дополнения.

Доказательство.

Рассмотрим два языка

$$L_1 = \{a^n b^n c^m \mid n \geq 0, m \geq 0\}$$

и

$$L_2 = \{a^n b^m c^m \mid n \geq 0, m \geq 0\}.$$

Оба эти языка контекстно-свободные. Действительно, L_1 может быть задан, например, КС-грамматикой

$$S \rightarrow S_1 S_2,$$

$$S_1 \rightarrow a S_1' b \mid \varepsilon,$$

$$S_2 \rightarrow c S_2' \mid \varepsilon,$$

или можно заметить, что L_1 есть сцепление двух КС-языков:

$$\{a^n b^n \mid n \geq 0\} \text{ и } \{c^m \mid m \geq 0\}$$

и, следовательно, этот язык контекстно-свободный. Аналогичные рассуждения справедливы для L_2 . Но

$$L_1 \cap L_2 = \{a^n b^n c^n \mid n \geq 0\},$$

который, как было показано ранее, не является КС-языком. Значит, класс КС-языков не замкнут относительно пересечения.

Предположим теперь, что дополнение к любому КС-языку есть тоже КС-язык. Тогда для указанных выше L_1 и L_2 их дополнения $\overline{L_1}$ и $\overline{L_2}$ тоже должны быть КС-языками так же, как и $\overline{L_1} \cup \overline{L_2}$. Но

$$\overline{\overline{L_1} \cup \overline{L_2}} = L_1 \cap L_2,$$

который, как только что было сказано, не является КС-языком. Следовательно, наше предположение неверно, а это значит, что класс КС-языков не замкнут относительно операции дополнения. Теорема доказана.

Интересно, что, в то время как пересечение двух КС-языков может не быть КС-языком, пересечение КС-языка с регулярным языком всегда дает КС-язык.

Теорема 50.

Пусть L_1 – контекстно-свободный язык, а L_2 – регулярный язык. Тогда язык

$$L_1 \cap L_2$$

является контекстно-свободным.

Доказательство.

Пусть НМА

$$M_1 = (Q, \Sigma, V, \theta_1, q_0, z_0, F_1)$$

допускает язык L_1 , а ДКА

$$M_2 = (P, \Sigma, \theta_2, p_0, F_2)$$

допускает язык L_2 . Построим магазинный автомат

$$M' = (Q', \Sigma, V, \theta', q'_0, z_0, F'),$$

который моделирует параллельное функционирование автоматов M_1 и M_2 : как только очередной символ читается из входной строки, так M' одновременно совершает шаги, производимые M_1 и M_2 . Для этого мы полагаем

$$\begin{aligned} Q' &= Q \times P, \\ q'_0 &= (q_0, p_0), \\ F' &= F_1 \times F_2 \end{aligned}$$

и определяем θ' таким образом, чтобы соотношение

$$((q_k, p_l), \alpha) \in \theta'((q_i, p_j), a, b)$$

имело место тогда и только тогда, когда

$$(q_k, \alpha) \in \theta_1(q_i, a, b)$$

и

$$\theta_2(p_l, a) = p_j.$$

В дополнение к этому мы также требуем, чтобы в случае, когда $\alpha = \varepsilon$, выполнялось равенство $p_j = p_l$. Иными словами, состояния автомата M' обозначаются парами (q_i, p_j) , представляющими те соответствующие состояния, в которых M_1 и M_2 могут оказаться после прочтения некоторой входной строки. Простым рассуждением по индукции можно показать, что соотношение

$$((q_0, p_0), \alpha, z_0) ((q_f, p_f), \beta),$$

где $q_f \in F_1, p_f \in F_2$, выполняется тогда и только тогда, когда

$$(q_0, \alpha, z_0) (q_f, \beta)$$

и

$$\theta_2^*(p_0, \alpha) = p_f.$$

Это означает, что строка допускается автоматом M' тогда и только тогда, когда она допускается автоматами M_1 и M_2 одновременно, то есть что она принадлежит пересечению $L(M_1) \cap L(M_2)$, что и требовалось доказать. **Теорема доказана.**

Перефразируя формулировку теоремы 35, будем говорить, что класс КС-языков замкнут относительно регулярного пересечения (понимая под этим пересечение с регулярным языком).

Пример 53.

Показать, что язык

$$L = \{a^n b^n | n \geq 0, n \neq 100\}$$

является контекстно-свободным.

Заметим, что

$$L = \{a^n b^n | n \geq 0\} \cap L_1,$$

где $L_1 = \{\alpha | \alpha \in \{a, b\}^* \setminus \{a^{100} b^{100}\}\}$, то есть L_1 состоит из всех строк в алфавите $\{a, b\}$, за исключением строки $a^{100} b^{100}$, а значит, является регулярным. Так как язык

$$\{a^n b^n | n \geq 0\}$$

контекстно-свободный, то, в силу теоремы 35, язык L тоже контекстно-свободный.

Пример 54.

Показать, что язык

$$L = \{\alpha | \alpha \in \{a, b, c\}^* \text{ и } n_a(\alpha) = n_b(\alpha) = n_c(\alpha)\}$$

не контекстно-свободный.

Здесь $n_a(\alpha)$ означает число вхождений символа a в строку α . Предположим противное, то есть что L – КС-язык. Тогда язык

$$L \cap L(a^* \cdot b^* \cdot c^*) = \{a^n b^n c^n | n \geq 0\}$$

также должен бы быть контекстно-свободным, так как $L(a^* \cdot b^* \cdot c^*)$ регулярен. Но ранее было показано, что это не так, следовательно, L не является КС-языком.

8.3. Некоторые алгоритмические проблемы для КС-языков

Самой существенной проблемой для любого класса языков, используемых на практике, является проблема вхождения (или проблема принадлежности): $\alpha \in L$? При обсуждении этой и других алгоритмических проблем для класса КС-языков будем предполагать, что языки заданы своими грамматиками.

Теорема 51.

Существует алгоритм, который для любой КС-грамматики и для любой строки α определяет, принадлежит строка α языку $L(G)$ или нет. Иными словами, проблема принадлежности для КС-языков алгоритмически разрешима.

Доказательство.

Очевидно, можно сразу предположить, что грамматика G имеет нормальную форму Хомского (так как любая КС-грамматика за конечное число шагов может быть приведена к ней с сохранением языка). Пусть

$$G = (N, T, S, P)$$

и пусть дана произвольная строка

$$\alpha = a_1 a_2 \dots a_n.$$

Определим подстроки строки α

$$\alpha_{ij} = a_i \dots a_j, \quad i \leq j,$$

и подмножества

$$N_{ij} = \{A \mid A \in N, A \Rightarrow_G^* \alpha_{ij}\},$$

где $1 \leq i \leq n, 1 \leq j \leq n$. Ясно, что $\alpha \in L(G)$ тогда и только тогда, когда

$$S \in N_{1n}.$$

Для вычисления N_{ij} заметим, что $A \in N_{ii}$ тогда и только тогда, когда G содержит продукцию $A \rightarrow a_i$. Поэтому N_{ii} может быть вычислено для всех $i = 1, 2, \dots, n$ простым просмотром α и всех продукций грамматики G . Далее заметим, что для $j > i$ из A выводима α_{ij} тогда и только тогда, когда в G существует продукция $A \rightarrow BC$ такая, что $B \Rightarrow_G^* \alpha_{ik}$ и $C \Rightarrow_G^* \alpha_{k+1,j}$ для некоторого k в интервале $i \leq k < j$. Иначе говоря,

$$N_{ij} = \bigcup_{k \in \{i, i+1, \dots, j-1\}} \{A \mid A \rightarrow BC \text{ и } B \in N_{ik}, C \in N_{k+1,j}\}$$

Упорядочивая пары индексов (i, j) в этом выражении, видим, что множества N_{ij} могут быть вычислены в следующей последовательности:

вычисляем сначала

$$N_{11}, N_{22}, \dots, N_{nn};$$

затем вычисляем

$$N_{12}, N_{23}, \dots, N_{n-1, n};$$

потом вычисляем

$$N_{13}, N_{24}, \dots, N_{n-2, n};$$

и так далее.

Ясно, что через конечное число шагов все N_{ij} будут вычислены, в том числе и N_{1n} , после чего останется убедиться, принадлежит S этому множеству или нет. **Теорема доказана.**

Пример 55.

Определить, принадлежит ли строка $\alpha = aabbb$ языку, порожденному КС-грамматикой G

$$\begin{aligned} S &\rightarrow AB, \\ A &\rightarrow BB|a, \\ B &\rightarrow AB|b. \end{aligned}$$

Применим описанную в теореме 36 процедуру. Заметим, что данная грамматика уже в нормальной форме Хомского, поэтому сразу вычисляем множества N_{ij} , где $1 \leq i \leq 5$, $1 \leq j \leq 5$:

$$\begin{aligned} N_{11} &= \{A\}, N_{22} = \{A\}, N_{33} = \{B\}, N_{44} = \{B\}, N_{55} = \{B\}, \\ N_{12} &= \emptyset, N_{23} = \{S, B\}, N_{34} = \{A\}, N_{45} = \{A\}, \\ N_{13} &= \{S, B\}, N_{24} = \{A\}, N_{35} = \{S, B\}, \\ N_{14} &= \{A\}, N_{25} = \{S, B\}, N_{15} = \{S, B\}. \end{aligned}$$

Видим, что $S \in N_{15}$, значит, $\alpha \in L(G)$.

Теорема 52.

Существует алгоритм, который по любой КС-грамматике $G = (N, T, S, P)$

определяет, пуст язык $L(G)$ или нет.

Доказательство.

Предположим, что $\varepsilon \notin L(G)$. Это можно сделать, так как приведением грамматики G к ε -свободному виду всегда можно эффективно определить, входит ли ε в $L(G)$, и если $\varepsilon \in L(G)$, то язык не пуст. Итак, пусть $\varepsilon \notin L(G)$. Используем алгоритм удаления из G бесполезных символов и продукций. Если при этом S окажется в множестве бесполезных переменных, тогда язык $L(G)$ пуст, а если нет, то $L(G)$ содержит по меньшей мере один элемент. **Теорема доказана.**

Теорема 53.

Существует алгоритм, который по любой КС-грамматике $G = (N, T, S, P)$

определяет, бесконечен язык $L(G)$ или нет.

Доказательство.

Предположим, что G не содержит ни ε -продукций, ни цепных продукций, ни бесполезных символов. Допустим, что грамматика обладает повторяющимися переменными в том смысле, что существует некоторый нетерминал $A \in N$, для которого существует вывод

$$A \Rightarrow^* \alpha A \beta.$$

Так как G по предположению не имеет ε -продукций и цепных продукций, то α и β не могут быть одновременно пусты. Так как A не является ни ε -порождающим, ни бесполезным символом, то имеем:

$$S \varphi A \psi \gamma \text{ и } A \delta.$$

Но тогда

$$S \varphi A \psi \varphi \alpha^i A \beta^i \psi \varphi \alpha^i \delta \beta^i \psi,$$

для любого $i = 1, 2, \dots$, следовательно, $L(G)$ бесконечен. Если ни одна из переменных не повторяется в выводах грамматики G , тогда длина любого вывода в G ограничена числом $|N|$, а значит, множество всех выводов в G конечно. Следовательно, конечен и язык $L(G)$.

Итак, для построения алгоритма, проверяющего, бесконечен ли язык $L(G)$, достаточно иметь алгоритм проверки наличия в G повторяющихся переменных. Это может быть сделано посредством построения графов зависимостей для переменных таким образом, что дуга (A, B) графа соответствует продукции $A \rightarrow \alpha B \beta$. Тогда любая переменная, которая является основанием некоторого цикла, будет повторяющейся переменной в G . Следовательно, G имеет повторяющиеся переменные тогда и только тогда, когда граф зависимостей содержит цикл. В силу конечности графа эта проверка может быть проведена за конечное число шагов, а тем самым, мы получаем алгоритм проверки конечности языка $L(G)$. Теорема доказана.

В заключение заметим, что далеко не все алгоритмические проблемы для контекстно-свободных языков разрешимы. Так, например, не существует алгоритма для определения, эквивалентны ли две КС-грамматики.

Глава 9. Классификация и преобразования грамматик

В следующих главах на примере модельного языка приводятся некоторые важные для теории трансляции сведения. Содержание этих глав основано на материале учебного пособия [9].

9.1. Классификация грамматик и языков по Хомскому

Обозначим через Σ^* множество, содержащее все строки в алфавите Σ , включая пустую строку ε . Например, если $\Sigma = \{0,1\}$, то $\Sigma^* = \{\varepsilon, 0, 1, 00, 11, 01, 10, 000, 001, 011, \dots\}$.

Через Σ^+ обозначим множество, содержащее все строки в алфавите Σ , исключая пустую строку ε . Следовательно,

$$\Sigma^* = \Sigma^+ \cup \{\varepsilon\}.$$

Каждый язык в алфавите Σ является подмножеством множества Σ^* .

Напомним, что *декартовым произведением* множеств A и B называется множество

$$A \times B = \{(a,b) \mid a \in A, b \in B\}.$$

Как было сказано в главе 1, (*порождающая*) грамматика G – это четверка (N, T, S, P) , где:

T – алфавит *терминальных символов (терминалов)*,

N – алфавит *нетерминальных символов (нетерминалов)*, не пересекающийся с T ;

P – конечное подмножество множества

$(T \cup N)^+ \times (T \cup N)^*$; элемент (α, β) множества P называется *продукцией* или *правилом вывода* и записывается в виде $\alpha \rightarrow \beta$,

S – *начальный символ (цель)* грамматики, $S \in N$.

В соответствии с принятыми ранее соглашениями для записи правил вывода с одинаковыми левыми частями

$$\alpha \rightarrow \beta_1$$

$$\alpha \rightarrow \beta_2$$

...

$$\alpha \rightarrow \beta_n$$

используется сокращенная запись:

$$\alpha \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n.$$

Каждое $\beta_i, i = 1, 2, \dots, n$, будем называть альтернативой правила вывода из строки α .

Дадим теперь классификацию грамматик и языков по Хомскому (при этом грамматики классифицируются по виду их правил вывода).

ТИП 0

Грамматика $G = (N, T, S, P)$ называется *грамматикой типа 0*, если на правила вывода не накладывается никаких ограничений (кроме тех, которые указаны в определении грамматики).

ТИП 1

Грамматика $G = (N, T, S, P)$ называется *неукорачивающей грамматикой*, если каждое правило из P имеет вид

$$\alpha \rightarrow \beta,$$

где $\alpha \in (T \cup N)^+$, $\beta \in (T \cup N)^+$ и $|\alpha| \leq |\beta|$.

Грамматика $G = (N, T, S, P)$ называется *контекстно-зависимой (КЗ)*, если каждое правило из P имеет вид $\alpha \rightarrow \beta$, где

$$\alpha = \xi_1 A \xi_2; \beta = \xi_1 \gamma \xi_2; A \in N; \gamma \in (T \cup N)^+; \xi_1, \xi_2 \in (T \cup N)^*.$$

Грамматику типа 1 можно определить как неукорачивающую либо как контекстно-зависимую. Выбор определения не влияет на множество языков, порождаемых грамматиками этого класса, поскольку доказано, что множество языков, порождаемых неукорачивающими грамматиками, совпадает с множеством языков, порождаемых КЗ-грамматиками.

ТИП 2

Грамматика $G = (N, T, S, P)$ называется *контекстно-свободной (КС)*, если каждое правило из P имеет вид $A \rightarrow \beta$, где $A \in N$, $\beta \in (T \cup N)^+$.

Грамматика $G = (N, T, S, P)$ называется *укорачивающей контекстно-свободной (УКС)*, если каждое правило из P имеет вид $A \rightarrow \beta$, где $A \in N$, $\beta \in (T \cup N)^*$.

Граматику типа 2 можно определить как контекстно-свободную либо как укорачивающую контекстно-свободную. Возможность выбора обусловлена тем, что для каждой УКС-грамматики существует почти эквивалентная КС-грамматика.

ТИП 3

Грамматика $G = (N, T, S, P)$ называется *праволинейной* (см. гл. 3), если каждое правило из P имеет вид $A \rightarrow aB$ либо $A \rightarrow a$, где $A \in N, B \in N, a \in T$.

Грамматика $G = (N, T, S, P)$ называется *леволинейной* (см. гл. 3), если каждое правило из P имеет вид $A \rightarrow Va$ либо $A \rightarrow a$, где $A \in N, V \in N, a \in T$.

Граматику типа 3 (*регулярную, R-грамматику*) можно определить как праволинейную либо как леволинейную.

Выбор определения не влияет на множество языков, порождаемых грамматиками этого класса, поскольку, как было доказано в главе 3, множество языков, порождаемых праволинейными грамматиками, совпадает с множеством языков, порождаемых леволинейными грамматиками.

Соотношения между типами грамматик:

- любая регулярная грамматика является КС-грамматикой;
- любая регулярная грамматика, очевидно, является УКС-грамматикой;
- любая КС-грамматика является КЗ-грамматикой;
- любая КС-грамматика является неукорачивающей грамматикой;
- любая КЗ-грамматика является грамматикой типа 0;
- любая неукорачивающая грамматика является грамматикой типа 0.

Замечание: УКС-грамматика, содержащая продукции вида $A \rightarrow \varepsilon$, не является КЗ-грамматикой и не является неукорачивающей грамматикой.

Определение 47.

Язык L является языком типа k , если его можно описать грамматикой G типа k , т. е. $L = L(G)$, $k = 0, 1, 2, 3$.

Соотношения между типами языков:

- каждый регулярный язык является КС-языком, но существуют КС-языки, которые не являются регулярными (например, $L = \{a^n b^n \mid n > 0\}$);

- каждый КС-язык является КЗ-языком, но существуют КЗ-языки, которые не являются КС-языками (например, $L = \{a^n b^n c^n \mid n > 0\}$);

- каждый КЗ-язык является языком типа 0.

Замечание: УКС-язык, содержащий пустую строку, не является КЗ-языком. Кроме того, следует подчеркнуть, что если язык задан грамматикой типа k , то это не значит, что не существует грамматики типа k' ($k' > k$), описывающей тот же язык. Поэтому, когда говорят о языке типа k , обычно имеют в виду максимально возможный номер k .

Например, КЗ-грамматика $G_1 = (\{A, S\}, \{0, 1\}, S, P_1)$ и КС-грамматика $G_2 = (\{S\}, \{0, 1\}, S, P_2)$, где

$$P_1: S \rightarrow 0A1P_2: S \rightarrow 0S1 \mid 01,$$

$$0A \rightarrow 00A1$$

$$A \rightarrow \varepsilon$$

описывают один и тот же язык

$$L = L(G_1) = L(G_2) = \{0^n 1^n \mid n > 0\}.$$

Язык L является КС-языком, так как существует КС-грамматика, его описывающая. Но он не является регулярным языком, так как не существует регулярной грамматики, описывающей этот язык.

9.2. Примеры грамматик и языков

Заметим, что если при описании грамматики указаны только правила вывода P , то будем считать, что большие латинские буквы обозначают нетерминальные символы, S – цель грамматики, все остальные символы – терминальные.

1) Язык типа 0:

$$L = \{ \mid n \geq 1 \}$$

$$G(L): S \rightarrow aaCFD$$

$$F \rightarrow AFB \mid AB$$

$$AB \rightarrow bBA$$

$$Ab \rightarrow bA$$

$$AD \rightarrow D$$

$$Cb \rightarrow bC$$

$$CB \rightarrow C$$

$$bCD \rightarrow \varepsilon$$

2) Язык типа 1:

$L = \{\text{все строки из } 0 \text{ и } 1 \text{ с одинаковым числом } 0 \text{ и } 1\}$

$G(L): S \rightarrow ASB \mid AB$

$$AB \rightarrow BA$$

$$A \rightarrow 0$$

$$B \rightarrow 1$$

3) Язык типа 2:

$$L = \{(ac)^n (cb)^n \mid n > 0\}$$

$G(L): S \rightarrow aAb \mid accb$

$$A \rightarrow cSc$$

4) Язык типа 3:

$$L = \{\omega \perp \mid \omega \in \{a, b\}^+, \text{ где нет двух рядом стоящих } a\}$$

$G(L): S \rightarrow A\perp \mid B\perp$

$$A \rightarrow a \mid Ba$$

$$B \rightarrow b \mid Bb \mid Ab$$

9.3. Грамматический разбор и преобразования грамматик

Строка принадлежит языку, порождаемому грамматикой, только в том случае, если существует ее вывод из цели этой грамматики. Процесс построения такого вывода (а следовательно, и определения принадлежности строки языку) называется *грамматическим разбором*.

С практической точки зрения наибольший интерес представляет разбор поконтэкстно-свободным (КС или УКС) грамматикам. Их порождающей мощности достаточно для описания большей части синтаксической структуры языков программирования, для различных подклассов КС-грамматик имеются хорошо разработанные практически приемлемые способы решения задачи разбора.

Для КС-грамматик удобным графическим представлением вывода является дерево вывода. Напомним, что дерево называ-

ется *деревом вывода* (или *деревом разбора*) в КС-грамматике $G = (N, T, S, P)$, если выполнены следующие условия:

(1) каждая вершина дерева помечена символом из множества $N \cup T \cup \{\varepsilon\}$, при этом корень дерева помечен символом S ; листья – символами из $T \cup \{\varepsilon\}$;

(2) если вершина дерева помечена символом $A \in N$, а ее непосредственные потомки – символами a_1, a_2, \dots, a_n , где каждое $a_i \in T \cup N$, то $A \rightarrow a_1 a_2 \dots a_n$ – правило вывода в этой грамматике;

(3) если вершина дерева помечена символом $A \in N$, а ее единственный непосредственный потомок помечен символом ε , то $A \rightarrow \varepsilon$ – правило вывода в этой грамматике.

Дерево вывода можно строить *нисходящим* либо *восходящим* способом. При нисходящем разборе дерево вывода формируется от корня к листьям; на каждом шаге для вершины, помеченной нетерминальным символом, пытаются найти такое правило вывода, чтобы имеющиеся в нем терминальные символы «проектировались» на символы исходной строки.

Метод восходящего разбора заключается в том, что исходную строку пытаются «свернуть» к начальному символу S ; на каждом шаге ищут подстроку, которая совпадает с правой частью какого-либо правила вывода; если такая подстрока находится, то она заменяется нетерминалом из левой части этого правила. Если грамматика однозначная, то при любом способе построения будет получено одно и то же дерево разбора.

В некоторых случаях КС-грамматика может содержать недостижимые и бесполезные нетерминальные символы, которые не участвуют в порождении строк языка и поэтому могут быть удалены из грамматики вместе с теми продукциями, в которые они входят.

Как было сказано ранее, символ $x \in T \cup N$ называется *недостижимым* в грамматике $G = (N, T, S, P)$, если он не появляется ни в одной сентенциальной форме этой грамматики.

Алгоритм удаления недостижимых символов

Вход: КС-грамматика $G = (N, T, S, P)$.

Выход: КС-грамматика $G' = (N', T', S, P')$, не содержащая недостижимых символов, для которой $L(G) = L(G')$.

Метод:

1. $\Sigma_0 = \{S\}$; $i = 1$.
2. $\Sigma_i = \{x \mid x \in T \cup N, \text{ в } P \text{ есть } A \rightarrow \alpha x \beta, A \in \Sigma_{i-1}\} \cup \Sigma_{i-1}$.
3. Если $\Sigma_i \neq \Sigma_{i-1}$, то $i = i + 1$ и переходим к шагу 2, иначе $N' = \Sigma_i \cap N$; $T' = \Sigma_i \cap T$; P' состоит из правил множества P , содержащих только символы из Σ_i ;
 $G' = (N', T', S, P')$.

Символ $A \in N$ называется бесполезным в грамматике $G = (N, T, S, P)$, если множество $\{\alpha \in T^* \mid A \Rightarrow^* \alpha\}$ пусто.

Алгоритм удаления бесполезных символов

Вход: КС-грамматика $G = (N, T, S, P)$.

Выход: КС-грамматика $G' = (T, N', P', S)$, не содержащая бесполезных символов, для которой $L(G) = L(G')$.

Метод:

Рекурсивно строим множества N_0, N_1, \dots

1. $N_0 = \emptyset$, $i = 1$.
2. $N_i = \{A \mid (A \rightarrow \alpha) \in P \text{ и } \alpha \in (N_{i-1} \cup T)^*\} \cup N_{i-1}$.
3. Если $N_i \neq N_{i-1}$, то полагаем $i = i + 1$ и переходим к шагу 2, иначе $N' = N_i$; P' состоит из правил множества P , содержащих только символы из $N' \cup T$; $G' = (T, N', P', S)$.

Определение 48.

КС-грамматику G назовем *приведенной*, если в ней нет недостижимых и бесполезных символов.

Алгоритм преобразования грамматики к приведенному виду:

(1) обнаруживаются и удаляются все бесполезные нетерминалы;

(2) обнаруживаются и удаляются все недостижимые символы.

Удаление символов сопровождается удалением продукций, содержащих эти символы.

Заметим, что если в этом алгоритме переставить шаги (1) и (2), то не всегда результатом будет приведенная грамматика.

Для описания синтаксиса языков программирования стараются использовать однозначные приведенные КС-грамматики.

Глава 10. Методы построения трансляторов. Лексический анализ

Практически во всех трансляторах (и в компиляторах, и в интерпретаторах) в том или ином виде присутствуют перечисленные ниже процессы:

- лексический анализ,
- синтаксический анализ,
- семантический анализ,
- генерация внутреннего представления программы,
- оптимизация,
- генерация объектной программы.

В конкретных компиляторах порядок этих процессов может быть несколько иным, некоторые из них могут объединяться в одну фазу, другие могут выполняться в течение всего процесса компиляции. В интерпретаторах и при смешанной стратегии трансляции некоторые этапы могут вообще отсутствовать.

Далее мы рассмотрим некоторые методы, используемые для построения анализаторов (лексического, синтаксического и семантического), язык промежуточного представления программы, способ генерации промежуточной программы, ее интерпретации. Излагаемые алгоритмы и методы иллюстрируются на примере модельного паскалеподобного языка (М-языка). Все алгоритмы записаны на языке Си.

Информацию о других методах, алгоритмах и приемах, используемых при создании трансляторов, можно найти в [1, 2, 3, 4, 5, 8].

10.1. Описание модельного языка

Рассматриваемый нами модельный язык (М-язык) определим следующим образом:

$P \rightarrow \mathbf{program} D1; B \perp$

$D1 \rightarrow \mathbf{var} D \{;D\}$

$D \rightarrow I \{,I\}: [\mathbf{int} | \mathbf{bool}]$

$B \rightarrow \mathbf{begin} S \{;S\} \mathbf{end}$

$S \rightarrow I := E | \mathbf{if} E \mathbf{then} S \mathbf{else} S | \mathbf{while} E \mathbf{do} S | B | \mathbf{read} (I) | \mathbf{write} (E)$

$E \rightarrow E1 [= | < | > | !=] E1$
 $E1 \rightarrow T \{ [+ | - | or] T \}$
 $T \rightarrow F \{ [* | / | and] F \}$
 $F \rightarrow I | N | L | not F | (E)$
 $L \rightarrow \mathbf{true} | \mathbf{false}$
 $I \rightarrow C | IC | IR$
 $N \rightarrow R | NR$
 $C \rightarrow a | b | \dots | z | A | B | \dots | Z$
 $R \rightarrow 0 | 1 | 2 | \dots | 9$

Здесь запись вида $\{\alpha\}$ означает итерацию строки α , то есть в порождаемой строке в этом месте может находиться либо ϵ , либо α , либо $\alpha\alpha$, либо $\alpha\alpha\alpha$, и т. д. Запись вида $[\alpha | \beta]$ означает, что в порождаемой строке в этом месте может находиться либо α , либо β . P – главный символ грамматики; символ \perp – маркер конца текста программы.

Теперь введем так называемые *контекстные условия*.

- Любое имя, используемое в программе, должно быть описано и только один раз.
- В операторе присваивания типы переменной и выражения должны совпадать.
- В условном операторе и в операторе цикла в качестве условия возможно только логическое выражение.
- Операнды операции отношения должны быть целочисленными.
- Тип выражения и совместимость типов операндов в выражении определяются по обычным правилам; старшинство операций задано синтаксисом.

В любом месте программы, кроме идентификаторов, служебных слов и чисел, может находиться произвольное число пробелов и комментариев вида $\{< \text{любые символы, кроме} \} \text{ и } \perp\}$. **True, false, read, write** – служебные слова (их нельзя переопределять, как стандартные идентификаторы Паскаля). Сохраняется паскалевское правило о разделителях между идентификаторами, числами и служебными словами.

10.2. Лексический анализ

Рассмотрим методы и средства, которые обычно используются при построении лексических анализаторов. В основе таких анализаторов лежат регулярные грамматики, поэтому рассмотрим грамматики этого класса более подробно.

Примем следующие *соглашения*:

- в дальнейшем, если не оговорено особо, под регулярной грамматикой будем понимать левостороннюю грамматику;
- будем считать, что анализируемая строка всегда заканчивается специальным символом \perp (признаком конца строки).

Для грамматик этого типа существует алгоритм определения того, принадлежит ли анализируемая строка языку, порожаемой этой грамматикой (*алгоритм разбора*):

(1) первый символ исходной строки $a_1 a_2 \dots a_n \perp$ заменяем нетерминалом A , для которого в грамматике есть правило вывода $A \rightarrow a_1$ (другими словами, производим «свертку» терминала a_1 к нетерминалу A);

(2) затем многократно (до тех пор, пока не считаем признак конца строки) выполняем следующие шаги: полученный на предыдущем шаге нетерминал A и расположенный непосредственно справа от него очередной терминал a_i исходной строки заменяем нетерминалом B , для которого в грамматике есть правило вывода $B \rightarrow A a_i$ ($i = 2, 3, \dots, n$).

Это эквивалентно построению дерева разбора методом «снизу-вверх»: на каждом шаге алгоритма строим один из уровней в дереве разбора, «поднимаясь» от листьев к корню.

При работе этого алгоритма возможны следующие ситуации:

- прочитана вся строка; на каждом шаге находилась единственная нужная «свертка»; на последнем шаге свертка произошла к символу S (это означает, что исходная строка $a_1 a_2 \dots a_n \perp \in L(G)$);
- прочитана вся строка; на каждом шаге находилась единственная нужная «свертка»; на последнем шаге свертка произошла к символу, отличному от S , что означает, что исходная строка $a_1 a_2 \dots a_n \perp \notin L(G)$;
- на некотором шаге не нашлось нужной свертки, то есть для полученного на предыдущем шаге нетерминала A и располо-

женного непосредственно справа от него очередного терминала a_i исходной строки не нашлось нетерминала B , для которого в грамматике было бы правило вывода $B \rightarrow Aa_i$ (это означает, что исходная строка $a_1a_2\dots a_n \perp \notin L(G)$);

- на некотором шаге работы алгоритма оказалось, что есть более одной подходящей свертки, то есть в грамматике разные нетерминалы имеют правила вывода с одинаковыми правыми частями, и поэтому непонятно, к какому из них производить свертку. Это говорит о *недетерминированности разбора*. Анализ этой ситуации будет дан ниже.

Допустим, что разбор на каждом шаге детерминированный. Для того чтобы быстрее находить правило с подходящей правой частью, зафиксируем все возможные свертки (это определяется только грамматикой и не зависит от вида анализируемой строки).

Это можно сделать в виде таблицы, строки которой помечены нетерминальными символами грамматики, столбцы – терминальными. Значение каждого элемента таблицы – это нетерминальный символ, к которому можно свернуть пару «нетерминал-терминал», которыми помечены соответствующие строка и столбец. Например, для грамматики $G = (\{S, A, B, C\}, \{a, b, \perp\}, S, P)$, такая таблица будет выглядеть следующим образом:

P: $S \rightarrow C\perp$
 $C \rightarrow Ab \mid Ba$
 $A \rightarrow a \mid Ca$
 $B \rightarrow b \mid Cb$

	a	b	\perp
C	A	B	S
A	–	C	–
B	C	–	–
S	–	–	–

Знак «–» ставится в том случае, если для пары «терминал-нетерминал» свертки нет.

Но чаще информацию о возможных свертках представляют в виде *диаграммы состояний* (ДС) – неупорядоченного ориентированного помеченного графа, который строится следующим образом:

(1) строят вершины графа, помеченные нетерминалами грамматики (для каждого нетерминала – одну вершину), и еще одну

вершину, помеченную символом, отличным от нетерминальных (например, H). Эти вершины будем называть *состояниями*, H – начальное состояние.

(2) соединяем эти состояния дугами по следующим правилам:

а) для каждого правила грамматики вида $W \rightarrow t$ соединяем дугой состояния H и W (от H к W) и помечаем дугу символом t ;

б) для каждого правила $W \rightarrow Vt$ соединяем дугой состояния V и W (от V к W) и помечаем дугу символом t ;

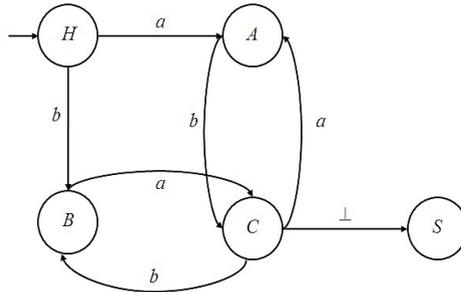


Рис. 34. Диаграмма состояний для грамматики G (см. пример выше)

Алгоритм разбора по диаграмме состояний:

(1) объявляем текущим состояние H ;

(2) затем многократно (до тех пор, пока не считаем признак конца строки) выполняем следующие шаги: считываем очередной символ исходной строки и переходим из текущего состояния в другое состояние по дуге, помеченной этим символом. Состояние, в которое мы при этом попадаем, становится текущим.

При работе этого алгоритма возможны следующие ситуации (аналогичные ситуациям, которые возникают при разборе непосредственно по регулярной грамматике):

(1) прочитана вся строка; на каждом шаге находилась единственная дуга, помеченная очередным символом анализируемой строки; в результате последнего перехода оказались в состоянии S . Это означает, что исходная строка принадлежит $L(G)$.

(2) прочитана вся строка; на каждом шаге находилась единственная «нужная» дуга; в результате последнего шага оказались в состоянии, отличном от S . Это означает, что исходная строка не принадлежит $L(G)$.

(3) на некотором шаге не нашлось дуги, выходящей из текущего состояния и помеченной очередным анализируемым символом. Это означает, что исходная строка не принадлежит $L(G)$.

(4) на некотором шаге работы алгоритма оказалось, что есть несколько дуг, выходящих из текущего состояния, помеченных очередным анализируемым символом, но ведущих в разные состояния. Это говорит о *недетерминированности разбора*. Анализ этой ситуации будет приведен ниже.

Диаграмма состояний определяет *конечный автомат*, построенный по регулярной грамматике, который допускает множество строк, составляющих язык, определяемый этой грамматикой. Состояния и дуги ДС – это графическое изображение функции переходов конечного автомата из состояния в состояние при условии, что очередной анализируемый символ совпадает с символом-меткой дуги. Среди всех состояний выделяется начальное (считается, что в начальный момент своей работы автомат находится в этом состоянии) и конечное (если автомат завершает работу переходом в это состояние, то анализируемая строка им допускается).

Для более удобной работы с диаграммами состояний введем несколько соглашений:

а) если из одного состояния в другое выходит несколько дуг, помеченных разными символами, то будем изображать одну дугу, помеченную всеми этими символами;

б) непомеченная дуга будет соответствовать переходу при любом символе, кроме тех, которыми помечены другие дуги, выходящие из этого состояния.

в) введем состояние ошибки (ER); переход в это состояние будет означать, что исходная строка языку не принадлежит.

По диаграмме состояний легко написать анализатор для регулярной грамматики. Например, для грамматики $G = (\{S, A, B, C\}, \{a, b, \perp\}, S, P)$, где

$$\begin{aligned} P: & S \rightarrow C\perp \\ & C \rightarrow Ab \mid Ba \\ & A \rightarrow a \mid Ca \\ & B \rightarrow b \mid Cb \end{aligned}$$

анализатор будет таким:

```
#include<stdio.h>
int scan_G()
{
    enum state {H, A, B, C, S, ER};
    /* множество состояний */
    stateCS;
    /* CS – текущее состояние */
    FILE* fp; /* указатель на файл, в котором находится анализируемая строка */
    int c;
    CS=H;
    fp = fopen («data»,»r»);
    c = fgetc (fp);
    do {switch (CS)
    {
        case H: if (c == 'a')
        {
            c = fgetc(fp); CS = A;
        }
        else if (c == 'b')
        {
            c = fgetc(fp); CS = B;
        }
        else CS = ER;
        break;
        case A: if (c == 'b')
        {
            c = fgetc(fp); CS = C;
        }
        else CS = ER;
        break;
        case B: if (c == 'a')
        {
            c = fgetc(fp); CS = C;
        }
    }
}
```

```

else CS = ER;
break;
case C: if (c == 'a')
{
    c = fgetc(fp); CS = A;
}
else if (c == 'b')
{
    c = fgetc(fp); CS = B;
}
else if (c == '⊥') CS = S;
else CS = ER;
break;
}
}
while (CS != S && CS != ER);
if (CS == ER) return -1; else return 0;
}

```

10.3. О недетерминированном разборе

При анализе по регулярной грамматике может оказаться, что несколько нетерминалов имеют одинаковые правые части, и поэтому неясно, к какому из них делать свертку (см. ситуацию 4 в описании алгоритма). В терминах диаграммы состояний это означает, что из одного состояния выходят несколько дуг, ведущих в разные состояния, но помеченных одним и тем же символом.

Например, для грамматики $G = (\{S, A, B\}, \{a, b, \perp\}, S, P)$, где

$P: S \rightarrow A\perp$

$A \rightarrow a \mid Bb$

$B \rightarrow b \mid Bb$

разбор будет недетерминированным (так как у нетерминалов A и B есть одинаковые правые части – Bb). Такой грамматике будет соответствовать недетерминированный конечный автомат. В этом случае можно предложить алгоритм, который будет перебирать все возможные варианты сверток (переходов)

один за другим; если строка принадлежит языку, то будет найден путь, ведущий к успеху; если будут просмотрены все варианты, и каждый из них будет завершаться неудачей, то строка языку не принадлежит. Однако такой алгоритм практически неприемлем, поскольку при переборе вариантов мы, скорее всего, снова окажемся перед проблемой выбора и, следовательно, будем иметь «дерево отложенных вариантов». Один из наиболее важных результатов теории конечных автоматов состоит в том, что класс языков, определяемых недетерминированными конечными автоматами, совпадает с классом языков, определяемых детерминированными конечными автоматами.

Это означает, что для любого НКА всегда можно построить детерминированный КА, определяющий тот же язык.

10.4. Задачи лексического анализа

Лексический анализ (ЛА) – это первый этап процесса компиляции. На этом этапе символы, составляющие исходную программу, группируются в отдельные лексические элементы, называемые *лексемами*. Лексический анализ важен для процесса компиляции по нескольким причинам:

а) замена в программе идентификаторов, констант, ограничителей и служебных слов лексемами делает представление программы более удобным для дальнейшей обработки;

б) лексический анализ уменьшает длину программы, устраняя из ее исходного представления несущественные пробелы и комментарии;

в) если будет изменена кодировка в исходном представлении программы, то это отразится только на лексическом анализаторе.

Выбор конструкций, которые будут выделяться как отдельные лексемы, зависит от языка и от точки зрения разработчиков компилятора. Обычно принято выделять следующие типы лексем: идентификаторы, служебные слова, константы и ограничители. Каждой лексеме сопоставляется пара:

(тип_лексемы, указатель_на_информацию_о_ней).

Эту пару тоже будем называть лексемой, если это не будет вызывать недоразумений.

Таким образом, лексический анализатор – это транслятор, входом которого служит строка символов, представляющих исходную программу, а выходом – последовательность лексем. Очевидно, что лексемы перечисленных выше типов можно описать с помощью регулярных грамматик.

Например, идентификатор (I):

$$I \rightarrow a|b|\dots|z|Ia|Ib|\dots|Iz|I0|I1|\dots|I9$$

целое без знака (N):

$$N \rightarrow 0|1|\dots|9|N0|N1|\dots|N9$$

и т. д.

Для грамматик этого класса, как мы уже видели, существует простой и эффективный алгоритм анализа того, принадлежит ли заданная строка языку, порождаемому этой грамматикой. Однако перед лексическим анализатором стоит более сложная задача: он должен сам выделить в исходном тексте строку символов, представляющую лексему, а также преобразовать ее в пару (тип_лексемы, указатель_на_информацию_о_ней). Для того, чтобы решить эту задачу, опираясь на способ анализа с помощью диаграммы состояний, введем на дугах дополнительный вид пометок – пометки-действия. Теперь каждая дуга в ДС может выглядеть так:

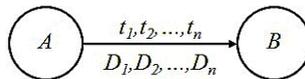


Рис. 35. Вид дуги в ДС

Смысл t_i прежний – если в состоянии A очередной анализируемый символ совпадает с t_i для какого-либо $i = 1, 2, \dots, n$, то осуществляется переход в состояние B ; при этом необходимо выполнить действия D_1, D_2, \dots, D_n .

10.5 Лексический анализатор для М-языка

Вход лексического анализатора – символы исходной программы на М-языке; результат работы – исходная программа в виде последовательности лексем (их внутреннего представления).

Лексический анализатор для модельного языка будем писать в два этапа: сначала построим диаграмму состояний с действиями для распознавания и формирования внутреннего представления лексем, а затем по ней напишем программу анализатора.

Первый этап: разработка ДС.

Представление лексем: все лексемы М-языка разделим на несколько классов; классы пронумеруем:

- служебные слова – 1,
- ограничители – 2,
- константы (целые числа) – 3,
- идентификаторы – 4.

Внутреннее представление лексем – это пара (номер_класса, номер_в_классе). Номер_в_классе – это номер строки в таблице лексем соответствующего класса.

Соглашение (об используемых переменных, типах и функциях).

1. Вводим переменные:

- buf – буфер для накопления символов лексемы;
- c – очередной входной символ;
- d – переменная для формирования числового значения константы;
- TW – таблица служебных слов М-языка;
- TD – таблица ограничителей М-языка;
- TID – таблица идентификаторов анализируемой программы;
- TNUM – таблица чисел-констант, используемых в программе.

Таблицы TW и TD заполнены заранее, так как их содержимое не зависит от исходной программы. TID и TNUM будут формироваться в процессе анализа. Для простоты будем считать, что все таблицы одного типа. Пусть tabl – имя типа этих таблиц, ptabl – указатель на tabl.

2. Вводим функции:

- void clear (void); – очистка буфера buf;
- void add (Void); – добавление символа c в конец буфера buf;

- `intlook (ptabl T)`; – поиск в таблице T лексемы из буфера `buf`; результат: номер строки таблицы с информацией о лексеме либо 0, если такой лексемы в таблице T нет;

- `inputl (ptabl T)`; – запись в таблицу T лексемы из буфера `buf`, если ее там не было; результат: номер строки таблицы с информацией о лексеме;

- `inputnum ()`; – запись в TNUM константы из `d`, если ее там не было; результат: номер строки таблицы TNUM с информацией о константе-лексеме;

- `voidmakelex (intk, inti)`; – формирование и вывод внутреннего представления лексемы; `k` – номер класса, `i` – номер в классе;

- `voidgc (void)` – функция, читающая из входного потока очередной символ исходной программы и заносящая его в переменную `s`.

Тогда диаграмма состояний для лексического анализатора будет иметь вид, представленный на рис. 36. Символом N_x в диаграмме (и в тексте программы) обозначен номер лексемы `x` в ее классе.

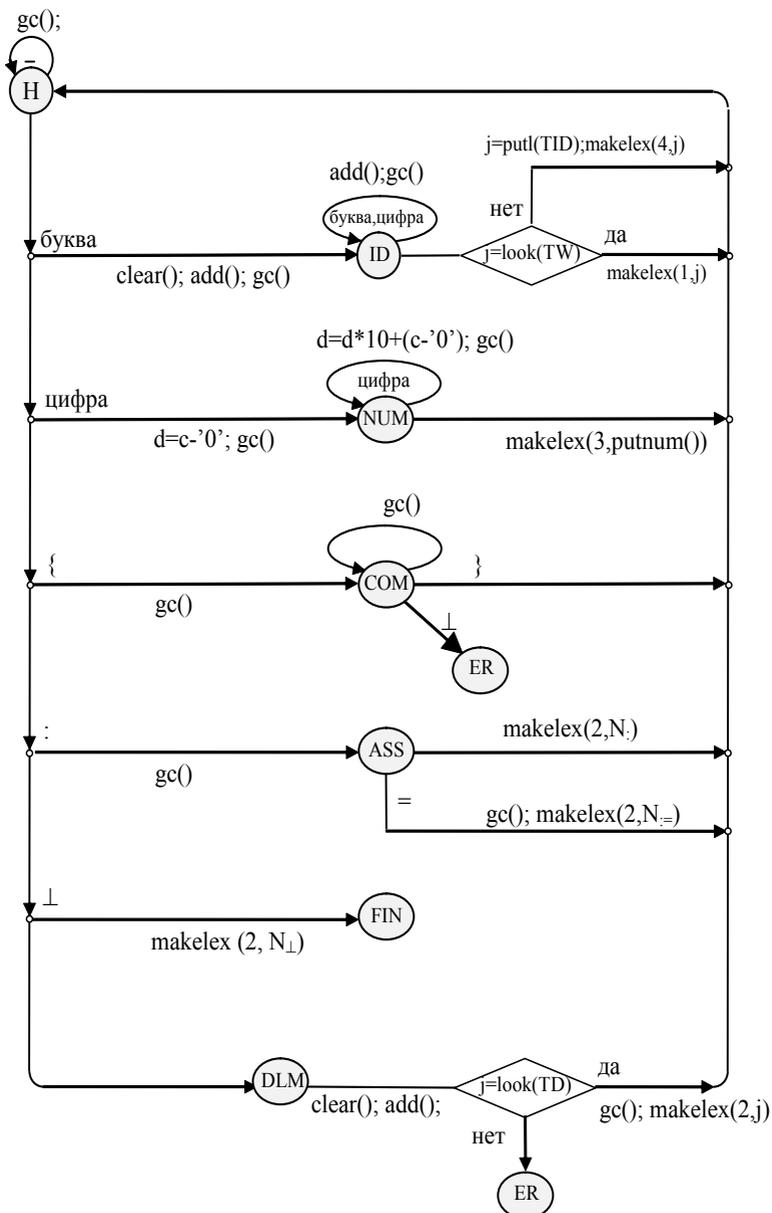


Рис. 36. Диаграмма состояний для лексического анализатора

Второй этап. По ДС пишем программу:

```
#include <stdio.h>
#include <ctype.h>
#define BUFSIZE 80
typedef struct tabl * ptabl;
extern ptabl TW, TID, TD, TNUM;
charbuf[BUFSIZE]; /* для накопления символов лексемы */
intc; /* очередной символ */
intd; /* для формирования числового значения константы */
voidclear(void); /* очистка буфера buf*/
voidadd(void); /* добавление символа с в конец буфера buf*/
intlook(ptabl); /* поиск в таблице лексемы из buf; результат: номер строки таблицы либо 0 */
intputl(ptabl); /* запись в таблицу лексемы из buf, если ее там не было; результат: номер строки таблицы */
intputnum(); /* запись в TNUM константы из d, если ее там не было; результат: номер строки таблицы TNUM*/
intj /* номер строки в таблице, где находится лексема, найденная функцией look*/
voidmakelex(int,int); /* формирование и вывод внутреннего представления лексемы */
voidscan (void)
{
    enum state {H,ID,NUM,COM,ASS,DLM,ER,FIN};
    state TC; /* текущее состояние */
    FILE* fp;
    TC = H;
    fp = fopen(«prog»,«r»); /* в файле «prog» находится текст исходной программы */
    c = fgetc(fp);
    do
    {
        switch (TC)
        {
            case H:
```

```

    if (c == ' ')
        c = fgetc(fp);
    else
        if (isalpha(c))
            {
                clear();
                add();
                c = fgetc(fp);
                TC = ID;
            }
        else if (isdigit (c))
            {
                d = c - '0';
                c = fgetc(fp);
                TC = NUM;
            }
        else if (c=='{')
            {
                c=fgetc(fp);
                TC = COM;
            }
        else if (c == ':')
            {
                c = fgetc(fp);
                TC = ASS;
            }
        else if (c == '⊥')
            {
                makelex(2, N1);
                TC = FIN;
            }
        else TC = DLM;
break;
case ID:
    if (isalpha(c) ||
        isdigit(c))

```

```

        {
            add(); c=fgetc(fp);
        }
    else
        {
            if (j = look(TW))
                makelex (1,j);
    else
        {
            j = putl (TID); makelex (4,j);
        };
        TC = H;
    };
    break;
    caseNUM:
        if (isdigit(c))
            {
                d=d*10+(c - '0');
                c=fgetc (fp);
            }
        else
            {
                makelex (3, putnum());
                TC = H;
            }
        break;
    /* ..... */
} /* конец switch */
} /* конец тела цикла */
while (TC != FIN && TC != ER);
if (TC == ER) printf(«ERROR!!!»);
elseprintf(«O.K.!!!»);
}

```

Глава 11. Синтаксический и семантический анализ

На этапе синтаксического анализа нужно установить, имеет ли строка лексем структуру, заданную синтаксисом языка, и зафиксировать эту структуру. Следовательно, снова надо решать задачу разбора: дана строка лексем, и надо определить, выводима ли она в грамматике, определяющей синтаксис языка. Однако структура таких конструкций как выражение, описание, оператор и т. п., более сложная, чем структура идентификаторов и чисел. Поэтому для описания синтаксиса языков программирования нужны более мощные грамматики, чем регулярные стороны, позволяют достаточно полно описать синтаксическую структуру реальных языков программирования; с другой стороны, для разных подклассов УКС-грамматик существуют достаточно эффективные алгоритмы разбора.

С теоретической точки зрения существует алгоритм, который по любой данной КС-грамматике и данной строке выясняет, принадлежит ли строка языку, порождаемому этой грамматикой. Но время работы такого алгоритма (синтаксического анализа с возвратами) экспоненциально зависит от длины строки, что с практической точки зрения совершенно неприемлемо.

Существуют табличные методы анализа ([3]), применимые ко всему классу КС-грамматик и требующие для разбора строк длины n времени cn^3 (алгоритм Кока-Янгера-Касами) либо cn^2 (алгоритм Эрли). Их разумно применять только в том случае, если для интересующего нас языка не существует грамматики, по которой можно построить анализатор с линейной временной зависимостью.

Алгоритмы анализа, расходующие на обработку входной строки линейное время, применимы только к некоторым подклассам КС-грамматик. Рассмотрим один из таких методов.

11.1. Метод рекурсивного спуска

Пример 56.

Пусть дана грамматика $G = (\{S, A, B\}, \{a, b, c, \perp\}, S, P)$, где

$P: S \rightarrow AB\perp$

$A \rightarrow a \mid cA$

$B \rightarrow bA$

и надо определить, принадлежит ли строка $caba$ языку $L(G)$.

Построим вывод этой строки:

$S \Rightarrow AB\perp \Rightarrow cAB\perp \Rightarrow caB\perp \Rightarrow cabA\perp \Rightarrow caba\perp$

Следовательно, строка принадлежит языку $L(G)$. Последовательность применений правил вывода эквивалентна построению дерева разбора методом «сверху вниз»:

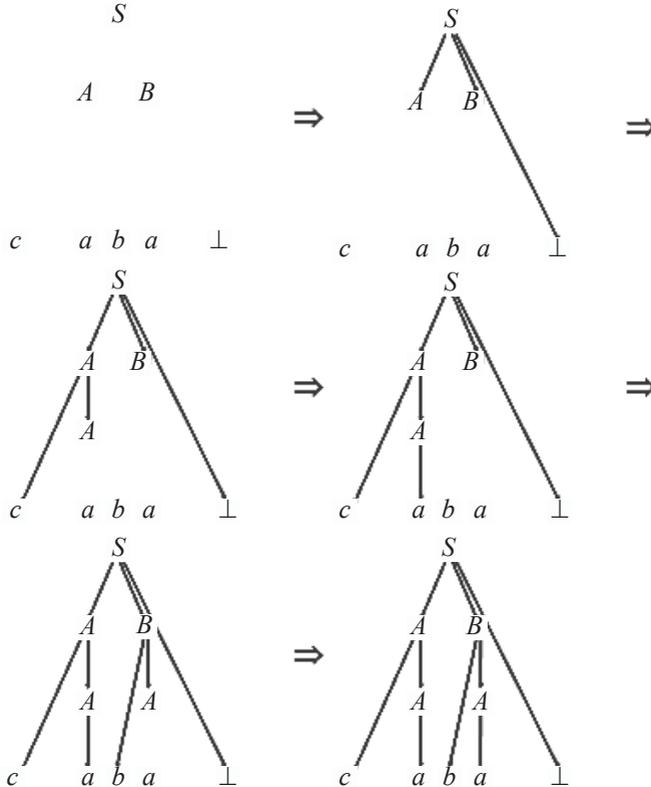


Рис. 37. Дерево разбора

Метод рекурсивного спуска (РС-метод) реализует этот способ практически «в лоб»: для каждого нетерминала грамматики создается своя процедура, носящая его имя; ее задача – начиная с указанного места исходной строки найти подстроку, которая выводится из этого нетерминала. Если такую подстроку считать не удастся, то процедура завершает свою работу вызовом процедуры обработки ошибки, которая выдает сообщение о том, что строка не принадлежит языку, и останавливает разбор. Если подстроку удалось найти, то работа процедуры считается нормально завершённой и осуществляется возврат в точку вызова. Тело каждой такой процедуры пишется непосредственно по правилам вывода соответствующего нетерминала: для правой части каждого правила осуществляется поиск подстроки, выводимой из этой правой части. При этом терминалы распознаются самой процедурой, а нетерминалы соответствуют вызовам процедур, носящих их имена.

Пример 57.

Совокупность процедур рекурсивного спуска для грамматики $G = (\{S, A, B\}, \{a, b, c, \perp\}, S, P)$, где

$$P: \begin{aligned} S &\rightarrow AB\perp \\ A &\rightarrow a \mid cA \\ B &\rightarrow bA \end{aligned}$$

будет такой:

```
#include <stdio.h>
intc;
FILE* fp; /* указатель на файл, в котором находится анализируемая строка */
voidA();
voidB();
voidERROR();/*функция обработки ошибок*/
void S()
{
    A(); B();
    if (c != '⊥') ERROR();
}
```

```

void A()
{
    if (c=='a')
        c = fgetc(fp);
    else
        if (c == 'c')
            {
                c = fgetc(fp);
                A();
            }
        else ERROR();
}
void B()
{
    if (c == 'b')
        {
            c = fgetc(fp);
            A();
        }
    else ERROR();
}
void ERROR()
{
    printf(«ERROR!!!»);
    exit(1);
}
main()
{
    fp = fopen(«data»,»r»);
    c = fgetc(fp);
    S();
    printf(«SUCCESS!!!»); exit(0);
}

```

11.2. О применимости метода рекурсивного спуска

Метод рекурсивного спуска применим в том случае, если каждое правило грамматики имеет вид:

а) либо $A \rightarrow \alpha$, где $\alpha \in (T \cup N)^*$, и это единственное правило вывода для этого нетерминала;

б) либо $A \rightarrow a_1\alpha_1 \mid a_2\alpha_2 \mid \dots \mid a_n\alpha_n$, где $a_i \in T$ для всех $i = 1, 2, \dots, n$; $a_i \neq a_j$ для $i \neq j$; $\alpha_i \in (T \cup N)^*$, т. е. если для нетерминала A правил вывода несколько, то они должны начинаться с терминалов, причем все эти терминалы должны быть различными. Ясно, что если правила вывода имеют такой вид, то рекурсивный спуск может быть реализован по выше изложенной схеме.

Естественно, возникает вопрос: если грамматика не удовлетворяет этим условиям, то существует ли эквивалентная КС-грамматика, для которой метод рекурсивного спуска применим? К сожалению, нет алгоритма, отвечающего на поставленный вопрос, то есть это алгоритмически неразрешимая проблема.

Изложенные выше ограничения являются достаточными, но не необходимыми. Попытаемся ослабить требования на вид правил грамматики.

1. При описании синтаксиса языков программирования часто встречаются правила, описывающие последовательность однотипных конструкций, отделенных друг от друга каким-либо знаком-разделителем (например, список идентификаторов при описании переменных, список параметров при вызове процедур и функций и т. п.).

Общий вид этих правил:

$L \rightarrow a \mid a, L$ (в сокращенной форме $L \rightarrow a \{,a\}$).

Формально здесь не выполняются условия применимости метода рекурсивного спуска, так как две альтернативы начинаются одинаковыми терминальными символами.

Действительно, в строке a, a, a, a, a из нетерминала L может выводиться и подстрока a , и подстрока a, a , и вся строка a, a, a, a, a . Неясно, какую из них выбрать в качестве подстроки, выводимой из L . Если принять решение, что в таких случаях будем выбирать самую длинную подстроку (что и требуется при разборе реальных языков), то разбор становится детерминированным.

Тогда для метода рекурсивного спуска процедура L будет такой:

```
void L()
{
    if (c != 'a') ERROR();
    while ((c = fgetc(fp)) == ',')
        if ((c = fgetc(fp)) != 'a')
            ERROR();
}
```

Важно, чтобы подстроки, следующие за строкой символов, выводимых из L , не начинались с разделителя (в нашем примере – с запятой), иначе процедура L попытается считать часть исходной строки, которая не выводится из L . Например, она может породиться нетерминалом B – «соседом» L в сентенциальной форме, как в грамматике

$$S \rightarrow LB\perp$$

$$L \rightarrow a \{, a\}$$

$$B \rightarrow ,b$$

Если для этой грамматики написать анализатор, действующий РС-методом, то строка a,a,a,b будет признана им ошибочной, хотя в действительности это не так. Нужно отметить, что в языках программирования ограничителем подобных серий всегда является символ, отличный от разделителя, поэтому подобных проблем не возникает.

2. Если грамматика не удовлетворяет требованиям применимости метода рекурсивного спуска, то можно попытаться преобразовать ее, то есть получить эквивалентную грамматику, пригодную для анализа этим методом.

а) если в грамматике есть нетерминалы, правила вывода которых леворекурсивны, то есть имеют вид

$$A \rightarrow A\alpha_1 \mid \dots \mid A\alpha_n \mid \beta_1 \mid \dots \mid \beta_m,$$

где $\alpha_i \in (T \cup N)^+$, $\beta_j \in (T \cup N)^*$, $i = 1, 2, \dots, n$; $j = 1, 2, \dots, m$, то непосредственно применять РС-метод нельзя.

Левую рекурсию всегда можно заменить правой:

$$A \rightarrow \beta_1 A' \mid \dots \mid \beta_m A'$$

$$A' \rightarrow \alpha_1 A' \mid \dots \mid \alpha_n A' \mid \varepsilon$$

Будет получена грамматика, эквивалентная данной, так как из нетерминала A по-прежнему выводятся строки вида $\beta_j \{\alpha_i\}$, где $i = 1, 2, \dots, n; j = 1, 2, \dots, m$.

б) если в грамматике есть нетерминал, у которого несколько правил вывода начинаются одинаковыми терминальными символами, то есть имеют вид

$$A \rightarrow a\alpha_1 \mid a\alpha_2 \mid \dots \mid a\alpha_n \mid \beta_1 \mid \dots \mid \beta_m,$$

где $a \in T; \alpha_i, \beta_j \in (T \cup N)^*$, то непосредственно применять РС-метод нельзя. Можно преобразовать правила вывода данного нетерминала, объединив правила с общими началами в одно правило:

$$A \rightarrow aA' \mid \beta_1 \mid \dots \mid \beta_m$$

$$A' \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$$

Будет получена грамматика, эквивалентная данной.

с) если в грамматике есть нетерминал, у которого несколько правил вывода, и среди них есть правила, начинающиеся нетерминальными символами, то есть имеют вид

$$A \rightarrow B_1 \alpha_1 \mid \dots \mid B_n \alpha_n \mid a_1 \beta_1 \mid \dots \mid a_m \beta_m$$

$$B_i \rightarrow \gamma_{i1} \mid \dots \mid \gamma_{ik}$$

...

$$B_n \rightarrow \gamma_{n1} \mid \dots \mid \gamma_{np},$$

где $B_i \in N; a_j \in T; \alpha_i, \beta_j, \gamma_{ij} \in (T \cup N)^*$, то можно заменить вхождения нетерминалов B_i их правилами вывода в надежде, что правило нетерминала A станет удовлетворять требованиям метода рекурсивного спуска:

$$A \rightarrow \gamma_{11} \alpha_1 \mid \dots \mid \gamma_{1k} \alpha_1 \mid \dots \mid \gamma_{n1} \alpha_n \mid \dots \mid \gamma_{np} \alpha_n \mid a_1 \beta_1 \mid \dots \mid a_m \beta_m$$

д) если допустить в правилах вывода грамматики пустую альтернативу, то есть правила вида

$$A \rightarrow a_1 \alpha_1 \mid \dots \mid a_n \alpha_n \mid \varepsilon,$$

то метод рекурсивного спуска может оказаться неприменимым (несмотря на то, что в остальном достаточные условия применимости выполняются).

Например, для грамматики $G = (\{S, A\}, \{a, b\}, S, P)$, где

$$P: \quad S \rightarrow bAa \\ A \rightarrow aA \mid \varepsilon$$

РС-анализатор, реализованный по обычной схеме, будет таким:

```
void S(void)
{
    if (c == 'b')
    {
        c = fgetc(fp);
        A();
    }
    if (c != 'a')
        ERROR();
}
else
    ERROR();
}
void A(void)
{
    if (c == 'a')
    {
        c = fgetc(fp);
        A();
    }
}
```

Тогда при анализе строки *baaa* функция *A()* будет вызвана три раза; она прочитает подстроку *aaa*, хотя третий символ *a* – это часть подстроки, выводимой из *S*. В результате окажется, что *baaa* не принадлежит языку, порождаемому грамматикой, хотя в действительности это не так.

Проблема заключается в том, что подстрока, следующая за строкой, выводимой из *A*, начинается таким же символом, как и строка, выводимая из *A*.

Однако в грамматике $G = (\{S, A\}, \{a, b, c\}, S, P)$,
 где $P: S \rightarrow bAc$

$$A \rightarrow aA \mid \varepsilon$$

нет проблем с применением метода рекурсивного спуска.

Выпишем условие, при котором ε -продукция делает неприемлемым РС-метод.

Определение 49.

Множество $FIRST(A)$ – это множество терминальных символов, которыми начинаются строки, выводимые из A в грамматике $G = (N, T, S, P)$, то есть

$$FIRST(A) = \{a \in T \mid A \Rightarrow^* a\alpha, A \in N, \alpha \in (T \cup N)^*\}.$$

Определение 50.

Множество $FOLLOW(A)$ – это множество терминальных символов, которые следуют за строками, выводимыми из A в грамматике $G = (N, T, S, P)$, то есть

$$FOLLOW(A) = \{a \in T \mid S \Rightarrow^* \alpha A \beta, \beta \Rightarrow^* a\gamma, A \in N, \alpha, \beta, \gamma \in (T \cup N)^*\}.$$

Тогда, если $FIRST(A) \cap FOLLOW(A) \neq \emptyset$, то метод рекурсивного спуска неприменим к данной грамматике.

Если

$$A \rightarrow \alpha_1 A \mid \dots \mid \alpha_n A \mid \beta_1 \mid \dots \mid \beta_m \mid \varepsilon$$

$$B \rightarrow \alpha A \beta$$

и $FIRST(A) \cap FOLLOW(A) \neq \emptyset$ (из-за вхождения A в правила вывода для B), то можно попытаться преобразовать такую грамматику:

$$B \rightarrow \alpha A'$$

$$A' \rightarrow \alpha_1 A' \mid \dots \mid \alpha_n A' \mid \beta_1 \beta \mid \dots \mid \beta_m \beta \mid \beta$$

$$A \rightarrow \alpha_1 A \mid \dots \mid \alpha_n A \mid \beta_1 \mid \dots \mid \beta_m \mid \varepsilon$$

Полученная грамматика будет эквивалентна исходной, так как из B по-прежнему выводятся строки вида $\alpha\{\alpha_i\}\beta$, либо $\alpha\{\alpha_i\}\beta$.

Однако правило вывода для нетерминального символа A' будет иметь альтернативы, начинающиеся одинаковыми терминальными символами, следовательно, потребуются дальнейшие преобразования, и успех не гарантирован. Метод рекурсивного спуска применим к достаточно узкому подклассу КС-грамматик. Известны более широкие подклассы КС-грамматик, для которых существуют эффективные анализаторы, обладающие тем же свойством, что и анализатор, написанный методом рекурсивного спуска, – входная строка считывается один раз слева направо и процесс разбора полностью детерминирован, в результате на обработку строки длины n расходуется время cn . К таким грамматикам относятся LL(k)-грамматики, LR(k)-грамматики, грамматики предшествования и некоторые другие (см., например, [2], [3]).

11.3. Синтаксический анализатор для М-языка

Будем считать, что синтаксический и лексический анализаторы взаимодействуют следующим образом: анализ исходной программы идет под управлением синтаксического анализатора; если для продолжения анализа ему нужна очередная лексема, то он запрашивает ее у лексического анализатора; тот выдает одну лексему и «замирает» до тех пор, пока синтаксический анализатор не запросит следующую лексему.

Соглашение об используемых переменных и типах:

- пусть лексический анализатор выдает лексемы типа `structlex {intclass; intvalue;}`;
- при описанном выше характере взаимодействия лексического и синтаксического анализаторов естественно считать, что лексический анализатор – это функция `getlex` с прототипом `structlexgetlex (void)`;
- в переменной `structlexcurr_lex` будем хранить текущую лексему, выданную лексическим анализатором.

Соглашение об используемых функциях:

- `intid (void)`; – результат равен 1, если `curr_lex.class = 4`, то есть `curr_lex` представляет идентификатор, и 0 – в противном случае;

- `intnum (void)`; – результат равен 1, если `curr_lex.class = 3`, то есть `curr_lex` представляет число-константу, и 0 – в противном случае;

- `inteq (char*s)`; – результат равен 1, если `curr_lex` представляет строку `s`, и 0 – иначе ;

- `voiderror(void)` – функция обработки ошибки; при обнаружении ошибки работа анализатора прекращается.

Тогда метод рекурсивного спуска реализуется с помощью процедур, создаваемых для каждого нетерминала грамматики.

P → **program D' ; B**⊥

```
void P (void)
{
    if (eq («program»))
        curr_lex = getlex();
    else
        ERROR();
    D1();
    if (eq («;»))
        curr_lex = getlex();
    else
        ERROR();
    B();
    if (!eq («⊥»))
        ERROR();
}
```

D' → **var D {; D}**

```
void D1 (void)
{
    if (eq («Var»))
```

```

        curr_lex = getlex();
    else
        ERROR();
    D();
    while (eq («;»))
    {
        curr_lex = getlex ();
        D();
    }
}

```

D → I {,I}: [int | bool]

```

void D (void)
{
    if (!id())
        ERROR();
    else
    {
        curr_lex = getlex();
        while (eq («;»))
        {
            curr_lex = getlex();
        }
        if (!id())
            ERROR();
        else
            curr_lex = getlex ();
    }
    if (!eq («;»))
        ERROR();
    else
    {
        curr_lex = getlex();
    }
}

```

```

if (eq («int»)
    || eq («bool»))
    curr_lex = getlex();
else
    ERROR();
}
}
}

```

E1 → T { [+ | - | or] T }

```

void E1 (Void)
{
    T();
    while (eq («+») || eq («-»)
        || eq («or»))
    {
        curr_lex = getlex();
        T();
    }
}

```

...

Для остальных нетерминалов грамматики модельного языка процедуры рекурсивного спуска пишутся аналогично.
«Запуск» синтаксического анализатора:

... curr_lex = getlex(); P(); ...

11.4. О семантическом анализе

Контекстно-свободные грамматики, с помощью которых описывают синтаксис языков программирования, не позволяют задавать контекстные условия, имеющиеся в любом языке.

Примеры наиболее часто встречающихся контекстных условий:

а) каждый используемый в программе идентификатор должен быть описан, но не более одного раза в одной зоне описания;

б) при вызове функции число фактических параметров и их типы должны соответствовать числу и типам формальных параметров;

с) обычно в языке накладываются ограничения на типы операндов любой операции, определенной в этом языке; на типы левой и правой частей в операторе присваивания; на тип параметра цикла; на тип условия в операторах цикла и условном операторе и т. п.

Проверку контекстных условий часто называют семантическим анализом. Его можно выполнять сразу после синтаксического анализа, некоторые требования можно контролировать во время генерации кода (например, ограничения на типы операндов в выражении), а можно совместить с синтаксическим анализом.

Мы выберем последний вариант: как только синтаксический анализатор распознает конструкцию, на компоненты которой наложены некоторые ограничения, проверяется их выполнение. Это означает, что на этапе синтаксического анализа придется выполнять некоторые дополнительные действия, осуществляющие семантический контроль.

Если для синтаксического анализа используется метод рекурсивного спуска, то в тела процедур РС-метода необходимо вставить вызовы дополнительных «семантических» процедур (семантические действия). Причем, как показывает практика, удобнее вставить их сначала в синтаксические правила, а потом по этим расширенным правилам строить процедуры РС-метода. Чтобы отличать вызовы семантических процедур от других символов грамматики, будем заключать их в угловые скобки.

Фактически, мы расширили понятие контекстно-свободной грамматики, добавив в ее правила вывода символы-действия.

Например, пусть в грамматике есть правило

$$A \rightarrow a\langle D_1 \rangle B\langle D_1; D_2 \rangle \mid bC\langle D_3 \rangle ,$$

здесь $A, B, C \in N$; $a, b \in T$; $\langle D_i \rangle$ означает вызов семантической процедуры D_i , $i = 1, 2, 3$. Имея такое правило грамматики, легко написать процедуру для метода рекурсивного спуска, которая будет выполнять синтаксический анализ и некоторые дополнительные действия:

```
void A()
{
    if (c=='a')
    {
        c = fgetc(fp);
        D1();
        B();
        D1();
        D2();
    }
    else
        if (c == 'b')
        {
            c = fgetc(fp);
            C();
            D3();
        }
    else
        ERROR();
}
```

Пример 58.

Написать грамматику, которая позволит распознавать строки языка $L = \{\alpha \in (0,1)^+ \mid \alpha \text{ содержит равное количество символов } 0 \text{ и } 1\}$.

Этого можно добиться, пытаясь чисто синтаксическими средствами описать строки, обладающие этим свойством. Но гораздо проще с помощью синтаксических правил описать произвольные строки из 0 и 1, а потом вставить действия для отбора строк с равным числом символов 0 и 1:

$$S \rightarrow \langle k0 = 0; k1 = 0; \rangle A \perp$$
$$A \rightarrow 0 \langle k0 = k0 + 1 \rangle A \mid 1 \langle k1 = k1 + 1 \rangle A \mid$$
$$0 \langle k0 = k0 + 1; \text{check()} \rangle \mid 1 \langle k1 = k1 + 1; \text{check()} \rangle,$$

где

```
void check()
{
    if (k0 != k1)
    {
        printf(«ERROR!!!»);
        exit(1);
    }
    else
    {
        printf(«SUCCESS!!!»);
        exit(0);
    }
}
```

Теперь по этой грамматике легко построить анализатор, распознающий строки с нужными свойствами.

11.5. Семантический анализатор для М-языка

Контекстные условия, выполнение которых нам надо контролировать в программах на М-языке, таковы.

1. Любое имя, используемое в программе, должно быть описано и только один раз.

2. В операторе присваивания типы переменной и выражения должны совпадать.

3. В условном операторе и в операторе цикла в качестве условия возможно только логическое выражение.

4. Операнды операции отношения должны быть целочисленными.

5. Тип выражения и совместимость типов операндов в выражении определяются по обычным правилам (как в Паскале).

Проверку контекстных условий совместим с синтаксическим анализом. Для этого в синтаксические правила вставим вызовы процедур, осуществляющих необходимый контроль, а затем перенесем их в процедуры рекурсивного спуска.

Обработка описаний

Для контроля согласованности типов в выражениях и типов выражений в операторах, необходимо знать типы переменных, входящих в эти выражения. Кроме того, нужно проверять, нет ли повторных описаний идентификаторов. Эта информация становится известной в тот момент, когда синтаксический анализатор обрабатывает описания. Следовательно, в синтаксические правила для описаний нужно вставить действия, с помощью которых будем запоминать типы переменных и контролировать единственность их описания.

Лексический анализатор запомнил в таблице идентификаторов TID все идентификаторы-лексемы, которые были им обнаружены в тексте исходной программы. Информацию о типе переменных и о наличии их описания естественно заносить в ту же таблицу.

Пусть каждая строка в TID имеет вид:

```
struct record {
    char *name; /* идентификатор */
    int declare; /* описан ? 1-»да», 0-»нет» */
    char* type; /* тип переменной */
    ...
};
```

Тогда таблица идентификаторов TID – это массив структур

```
#define MAXSIZE_TID 1000
struct record TID [MAXSIZE_TID];
```

причем i -я строка соответствует идентификатору-лексеме вида (4, i).

Лексический анализатор заполнил поле name, а значения полей declare и type будем заполнять на этапе семантического анализа. Для этого нам потребуется следующая функция:

Voiddecid (inti, char* t) – в i -той строке таблицы TID контролирует и заполняет поле declare и, если лексема (4, i) впервые встретилась в разделе описаний, заполняет поле type:

```
void decid (int i, char *t)
{
    if (TID [i].declare)
        ERROR(); /* повторное описание */
    else {TID [i].declare = 1;
        /* описан ! */
        strcpy (TID [i].type, t);}
    /* тип t ! */
}
```

Раздел описаний имеет вид

$D \rightarrow I \{,I\}: [\text{int} \mid \text{bool}]$,

то есть имени типа (int или bool) предшествует список идентификаторов. Эти идентификаторы (вернее, номера соответствующих

щих им строк таблицы TID) надо запоминать (например, в стеке), а когда будет проанализировано имя типа, заполнить поля declare и type в этих строках.

Для этого будем использовать функции работы со стеком целых чисел:

```
void ipush (inti);  
/* значение i – в стек*/  
int ipop (void); /*из стека – целое*/
```

Будем считать, что (-1) – «дно» стека; тогда функция

```
void dec (char *t)  
{  
    int i;  
    while ((i = ipop()) != -1)  
        decid(i,t);  
}
```

считывает из стека номера строк TID и заносит в них информацию о наличии описания и о типе t.

С учетом этих функций правило вывода с действиями для обработки описаний будет таким:

```
D → < ipush (-1) > I < ipush (curr_lex.Value) >  
    {, I < ipush (curr_lex.Value) > }:  
    [ int < dec («int») > | bool < dec («bool») > ]
```

Контроль контекстных условий в выражении.

Пусть есть функция

```
char* gettype (char* op, char*t1,  
               char*t2),
```

которая проверяет допустимость сочетания операндов типа t1 (первый операнд) и типа t2 (второй операнд) в операции op; если типы совместимы, то выдает тип результата этой операции; иначе – строку «no».

Типы операндов и обозначение операции будем хранить в стеке; для этого нам нужны функции для работы со стеком строк:

```
Voidspush (char*s);
           /* значение s – в стек */
char*spop (Void);
           /* из стека – строку */
```

Если в выражении встречается лексема – целое_число или логические константы true или false, то соответствующий тип сразу заносим в стек с помощью spush(«int») или spush(«bool»).

Если операнд – лексема-переменная, то необходимо проверить, описана ли она; если описана, то ее тип надо занести в стек. Эти действия можно выполнить с помощью функции checkid:

```
voidcheckid (void)
{
    int i;
    i = curr_lex.Value;
    if (TID [i].declare) /* описан? */
        spush (TID [i].type);
                           /* тип – в стек */
    else
        ERROR();
        /* описание отсутствует */
}
```

Тогда для контроля контекстных условий каждой тройки – «операнд-операция-операнд» будем использовать функцию checkop:

```
void checkop (void)
{
    char *op;
    char *t1;char *t2;
    char *res;
```

```

t2 = spop(); /* из стека – тип второго операнда */
op = spop(); /* из стека – обозначение операции */
t1 = spop(); /* из стека – тип первого операнда */
res = gettype (op,t1,t2);
        /* допустимо ?*/
if (strcmp (res, «no»))
    spush (res); /* да! */
else
    ERROR(); /* нет! */
}

```

Для контроля за типом операнда одноместной операции *not* будем использовать функцию *checknot*:

```

void checknot (void)
{
    if (strcmp (spop (), «bool»))
        ERROR();
    else
        spush («bool»);
}

```

Теперь главный вопрос: когда вызывать эти функции?

В грамматике модельного языка задано старшинство операций: наивысший приоритет имеет операция отрицания, затем в порядке убывания приоритета – группа операций умножения (*, /, and), группа операций сложения (+,-,or), операции отношения.

$$\begin{aligned}
 E &\rightarrow E1 \mid E1 [= \mid < \mid >] E1 \\
 E1 &\rightarrow T \{ [+ \mid - \mid \text{or}] T \} \\
 T &\rightarrow F \{ [* \mid / \mid \text{and}] F \} \\
 F &\rightarrow I \mid N \mid [\text{true} \mid \text{false}] \mid \text{not } F \mid (E)
 \end{aligned}$$

Именно это свойство грамматики позволит провести синтаксически-управляемый контроль контекстных условий.

Предлагаем читателю сравнить грамматики, описывающие выражения, состоящие из символов +, *, (,), i, и оценить, насколько они удобны для трансляции выражений.

G1: $E \rightarrow E + E \mid E * E \mid (E) \mid i$ G4: $E \rightarrow T \mid E + T$

$T \rightarrow F \mid T * F$

G2: $E \rightarrow E + T \mid E * T \mid TF \rightarrow i \mid (E)$

$T \rightarrow i \mid (E)$

G5: $E \rightarrow T \mid T + E$

G3: $E \rightarrow T + E \mid T * E \mid TT \rightarrow F \mid F * T$

$T \rightarrow i \mid (E) \mid F \rightarrow i \mid (E),$

Правила вывода выражений модельного языка с действиями для контроля контекстных условий:

$E \rightarrow E1 \mid E1 [= | < | >] \langle \text{spush}(\text{TD}[\text{curr_lex.Value}]) \rangle$
 $E1 \langle \text{checkop}() \rangle$

$E1 \rightarrow T \{ [+ | - | \text{or}] \langle \text{spush}(\text{TD}[\text{curr_lex.Value}]) \rangle$
 $T \langle \text{checkop}() \rangle \}$

$T \rightarrow F \{ [* | / | \text{and}] \langle \text{spush}(\text{TD}[\text{curr_lex.Value}]) \rangle$
 $F \langle \text{checkop}() \rangle \}$

$F \rightarrow I \langle \text{checkid}() \rangle \mid N \langle \text{spush}(\langle \text{int} \rangle) \rangle \mid [\text{true} \mid \text{false}]$
 $\langle \text{spush}(\langle \text{bool} \rangle) \rangle \mid \text{not } F \langle \text{checknot}() \rangle \mid (E)$

TD – это таблица ограничителей, к которым относятся и знаки операций; будем считать, что это массив

```
#define MAXSIZE_TD 50
char* TD[MAXSIZE_TD];
```

именно из этой таблицы по номеру лексемы в классе выбираем обозначение операции в виде строки.

11.6. Контроль контекстных условий в операторах

$S \rightarrow I := E \mid \text{if } E \text{ then } S \text{ else } S \mid \text{while } E \text{ do } S \mid B$
 $\mid \text{read } (I) \mid \text{write } (E)$

1) Оператор присваивания $I := E$

Контекстное условие: в операторе присваивания типы переменной I и выражения E должны совпадать.

В результате контроля контекстных условий выражения E в стеке останется тип этого выражения (как тип результата последней операции); если при анализе идентификатора I проверить, описан ли он, и занести его тип в тот же стек (для этого можно использовать функцию `checkid()`), то достаточно будет в нужный момент считать из стека два элемента и сравнить их:

```
void eqtype (void)
{
    if (strcmp (spop (), spop ()))
        ERROR();
}
```

Следовательно, правило для оператора присваивания:

$I \langle \text{checkid}() \rangle := E \langle \text{eqtype}() \rangle$

2) Условный оператор и оператор цикла

$\text{if } E \text{ then } S \text{ else } S \mid \text{while } E \text{ do } S$

Контекстные условия: в условном операторе и в операторе цикла в качестве условия возможны только логические выражения. В результате контроля контекстных условий выражения E в стеке останется тип этого выражения (как тип результата последней операции); следовательно, достаточно извлечь его из стека и проверить:

```
Voideqbool(void)
{
    if(strcmp(spop(),»bool«))
        ERROR();
}
```

Тогда правила для условного оператора и оператора цикла будут такими:

if E <eqbool()> then S else S | while E <eqbool()> do S

В итоге получаем процедуры для синтаксического анализа методом рекурсивного спуска с синтаксически-управляемым контролем контекстных условий, которые легко написать по правилам грамматики с действиями.

В качестве примера приведем функцию для нетерминала D (раздел описаний):

```
#include <string.h>
#define MAXSIZE_TID 1000
#define MAXSIZE_TD 50
char * TD[MAXSIZE_TD];
struct record
{
    char *name;
    int declare;
    char *type;
    /* ... */
};
struct record TID [MAXSIZE_TID];
/*описание функций ERROR(), getlex(),id(),eq(char*),типаstruct
lexиперемннойcurr_lex – валгоритмрекурсивногоспускадляМ-
языка*/
void ERROR(void);
struct lex
{
    int class;
    int value;
};
struct lex curr_lex;
struct lex getlex (void);
int id (void);
```

```

int eq (char *s);
void ipush (int i);
int ipop (void);

void decid (int i, char *t)
{
    if (TID [i].declare)
        ERROR();
    else
    {
        TID [i].declare = 1;
        strcpy (TID [i].type, t);
    }
}

void dec (char *t)
{
    int i;
    while ((i = ipop()) != -1)
        decid (i,t);
}

void D (void)
{
    ipush (-1);
    if (!id())
        ERROR();
    else
    {
        ipush (curr_lex.value);
        curr_lex = getlex ();
        while (eq («,»))
        {
            curr_lex = getlex ();
        }
        if (!id ())
            ERROR ();
    }
}

```

```

else
    {
        ipush (curr_lex.value);
curr_lex = getlex();
    }
}
if (!eq («:»))
    ERROR();
else
    {
        curr_lex = getlex ();
if (eq («int»))
    {
        curr_lex = getlex ();
dec («int»);
    }
else
    if (eq («bool»))
    {
        curr_lex = getlex();
dec («bool»);
    }
else
        ERROR();
    }
}
}

```

Глава 12. Генерация внутреннего представления программ

Результатом работы синтаксического анализатора должно быть некоторое внутреннее представление исходной строки лексем, которое отражает ее синтаксическую структуру. Программа в таком виде в дальнейшем может либо транслироваться в объектный код, либо интерпретироваться.

12.1. Язык внутреннего представления программы

Основные свойства языка внутреннего представления программ:

- а) он позволяет фиксировать синтаксическую структуру исходной программы;
- б) текст на нем можно автоматически генерировать во время синтаксического анализа;
- с) его конструкции должны относительно просто транслироваться в объектный код либо достаточно эффективно интерпретироваться.

Некоторые общепринятые способы внутреннего представления программ:

- а) постфиксная запись
- б) префиксная запись
- с) многоадресный код с явно именуемыми результатами
- д) многоадресный код с неявно именуемыми результатами
- е) связанные списочные структуры, представляющие синтаксическое дерево.

В основе каждого из этих способов лежит некоторый метод представления синтаксического дерева.

Замечание: чаще всего синтаксическим деревом называют дерево вывода исходной строки, в котором удалены вершины, соответствующие цепным продукциям вида $A \rightarrow B$, где $A, B \in N$.

Выберем в качестве языка для представления промежуточной программы *постфиксную запись* (ее часто называют *ПОЛИЗ* – польская инверсная запись).

В ПОЛИЗе операнды выписаны слева направо в порядке их использования. Знаки операций стоят таким образом, что знаку операции непосредственно предшествуют ее операнды.

Например, обычной (инфиксной) записи выражения

$$a*(b+c) -(d-e)/f$$

соответствует такая постфиксная запись:

$$abc+*de-f/-.$$

Обратите внимание на то, что в ПОЛИЗе порядок операндов остался таким же, как и в инфиксной записи, учтено старшинство операций, а скобки исчезли.

Более формально постфиксную запись выражений можно определить таким образом:

(1) если E является единственным операндом, то ПОЛИЗ выражения E – это этот операнд;

(2) ПОЛИЗом выражения $E_1 \theta E_2$, где θ – знак бинарной операции, E_1 и E_2 операнды для θ , является запись $E_1' E_2' \theta$, где E_1' и E_2' – ПОЛИЗ выражений E_1 и E_2 соответственно;

(3) ПОЛИЗом выражения θE , где θ – знак унарной операции, а E – операнд θ , является запись $E' \theta$, где E' – ПОЛИЗ выражения E ;

(4) ПОЛИЗом выражения (E) является ПОЛИЗ выражения E .

Запись выражения в такой форме очень удобна для последующей интерпретации (то есть вычисления значения этого выражения) с помощью стека: выражение просматривается один раз слева направо, при этом

(1) если очередной элемент ПОЛИЗа – это операнд, то его значение заносится в стек;

(2) если очередной элемент ПОЛИЗа – это операция, то на «верхушке» стека сейчас находятся ее операнды (это следует из определения ПОЛИЗа и предшествующих действий алгоритма); они извлекаются из стека, над ними выполняется операция, результат снова заносится в стек;

(3) когда выражение, записанное в ПОЛИЗе, прочитано, в стеке останется один элемент – это значение всего выражения.

Для интерпретации, кроме ПОЛИЗ-выражения, необходима дополнительная информация об операндах, хранящаяся в таблицах.

Может оказаться так, что знак бинарной операции по написанию совпадает со знаком унарной; например, знак «←» в большинстве языков программирования означает и бинарную операцию вычитания, и унарную операцию изменения знака. В этом случае во время интерпретации операции «←» возникнет неоднозначность: сколько операндов надо извлекать из стека и какую операцию выполнять. Устранить неоднозначность можно, по крайней мере, двумя способами:

а) заменить унарную операцию бинарной, то есть считать, что «←а» означает «0←а»;

б) либо ввести специальный знак для обозначения унарной операции; например, «←а» заменить на «&a». Важно отметить, что это изменение касается только внутреннего представления программы и не требует изменения входного языка.

Теперь необходимо разработать ПОЛИЗ для операторов входного языка. Каждый оператор языка программирования может быть представлен как n -местная операция с семантикой, соответствующей семантике этого оператора.

Оператор присваивания

$$I := E$$

в ПОЛИЗе будет записан как

$$\underline{I} E :=$$

где «:=» – это двухместная операция, а \underline{I} и E – ее операнды; \underline{I} означает, что операндом операции «:=» является адрес переменной I , а не ее значение.

Оператор перехода

в терминах ПОЛИЗа означает, что процесс интерпретации надо продолжить с того элемента ПОЛИЗа, который указан как операнд операции перехода.

Чтобы можно было ссылаться на элементы ПОЛИЗа, будем считать, что все они перенумерованы, начиная с 1 (допустим, занесены в последовательные элементы одномерного массива).

Пусть ПОЛИЗ оператора, помеченного меткой L , начинается с номера p , тогда оператор перехода **goto** L в ПОЛИЗе можно записать как

$$p !$$

где $!$ – операция выбора элемента ПОЛИЗа, номер которого равен p .

Немного сложнее окажется запись в ПОЛИЗе *условных операторов* и *операторов цикла*. Введем вспомогательную операцию – условный переход «по лжи» с семантикой

$$\text{if (not}B\text{) then goto}L$$

Это двухместная операция с операндами B и L . Обозначим ее $!F$, тогда в ПОЛИЗе она будет записана как

$$B p !F$$

где p – номер элемента, с которого начинается ПОЛИЗ оператора, помеченного меткой L .

Семантика условного оператора

$$\text{if}B\text{then}S_1\text{else}S_2$$

с использованием введенной операции может быть описана так:

$$\text{if (not }B\text{) then goto }L_2; S_1; \text{goto }L_3; L_2: S_2; L_3: \dots$$

Тогда ПОЛИЗ условного оператора будет таким:

$$B p_2 !F S_1 p_3 ! S_2 \dots ,$$

где p_i – номер элемента, с которого начинается ПОЛИЗ оператора, помеченного меткой L_i , $i = 2, 3$.

Семантика оператора цикла **while** B **do** S может быть описана так:

$$L_0: \text{if (not }B\text{) then goto }L_1; S; \text{goto }L_0; L_1: \dots .$$

Тогда ПОЛИЗ оператора цикла **while** будет таким:

$$B p_1 !F S p_0 ! \dots ,$$

где p_i – номер элемента, с которого начинается ПОЛИЗ оператора, помеченного меткой L_i , $i = 0, 1$.

Операторы ввода и вывода M -языка являются одноместными операциями. Пусть R – обозначение операции ввода, W – обозначение операции вывода.

Тогда оператор ввода **read** (I) в ПОЛИЗе будет записан как $I R$; оператор вывода **write** (E) – как $E W$.

Постфиксная польская запись операторов обладает всеми свойствами, характерными для постфиксной польской записи выражений, поэтому алгоритм интерпретации выражений

пригоден для интерпретации всей программы, записанной на ПОЛИЗе (нужно только расширить набор операций; кроме того, выполнение некоторых из них не будет давать результата, записываемого в стек).

Постфиксная польская запись может использоваться не только для интерпретации промежуточной программы, но и для генерации по ней объектной программы. Для этого в алгоритме интерпретации вместо выполнения операции нужно генерировать соответствующие команды объектной программы.

12.2. Синтаксически управляемый перевод

На практике синтаксический, семантический анализ и генерация внутреннего представления программы часто осуществляются одновременно.

Существует несколько способов построения промежуточной программы. Один из них, называемый синтаксически управляемым переводом, особенно прост и эффективен.

В основе синтаксически управляемого перевода лежит уже известная нам грамматика с действиями (см. раздел о контроле контекстных условий). Теперь, параллельно с анализом исходной строки лексем, будем выполнять действия по генерации внутреннего представления программы. Для этого дополним грамматику вызовами соответствующих процедур генерации.

Содержательный пример – генерация внутреннего представления программы для М-языка, приведен ниже, а здесь в качестве иллюстрации рассмотрим более простой пример.

Пусть есть грамматика, описывающая простейшее арифметическое выражение:

$$E \rightarrow T \{+T\}$$
$$T \rightarrow F \{*F\}$$
$$F \rightarrow a \mid b \mid (E)$$

Тогда грамматика с действиями по переводу этого выражения в ПОЛИЗ будет такой:

$$E \rightarrow T \{+T \langle \text{putchar}(' +') \rangle\}$$
$$T \rightarrow F \{*F \langle \text{putchar}('*') \rangle\}$$
$$F \rightarrow a \langle \text{putchar}(' a') \rangle \mid b \langle \text{putchar}(' b') \rangle \mid (E)$$

Этот метод можно использовать для перевода строк одного языка в строки другого языка (что, собственно, мы и делали, занимаясь переводами в ПОЛИЗ строк лексем).

Например, с помощью грамматики с действиями выполним перевод строк языка

$$L_1 = \{0^n 1^m \mid n, m > 0\}$$

в соответствующие строки языка

$$L_2 = \{a^m b^n \mid n, m > 0\}:$$

Язык L_1 можно описать грамматикой

$$S \rightarrow 0S \mid 1A$$

$$A \rightarrow 1A \mid \varepsilon$$

Вставим действия по переводу строк вида $0^n 1^m$ в соответствующие строки вида $a^m b^n$:

$$S \rightarrow 0S \langle \text{putchar}('b') \rangle \mid 1 \langle \text{putchar}('a') \rangle A$$

$$A \rightarrow 1 \langle \text{putchar}('a') \rangle A \mid \varepsilon$$

Теперь при анализе строк языка L_1 с помощью действий будут порождаться соответствующие строки языка L_2 .

12.3. Генератор внутреннего представления программы на M-языке

Каждый элемент в ПОЛИЗе – это лексема, то есть пара вида (номер_класса, номер_в_классе). Нам придется расширить набор лексем:

1) будем считать, что новые операции (!, !F, R, W) относятся к классу ограничителей, как и все другие операции модельного языка;

2) для ссылок на номера элементов ПОЛИЗа введем лексемы класса 0, то есть $(0, p)$ – лексема, обозначающая p -ый элемент в ПОЛИЗе;

3) для того, чтобы различать операнды-значения-переменных и операнды-адреса-переменных (например, в ПОЛИЗе оператора присваивания), операнды-значения будем обозначать лексемами класса 4, а для операндов-адресов введем лексемы класса 5.

Будем считать, что генерируемая программа размещается в массиве P, переменная free – номер первого свободного элемента в этом массиве:

```

#define MAXLEN_P 10000
struct lex
{
    int class;
    int value;
}
struct lex P [ MAXLEN_P];
intfree = 0;

```

Для записи очередного элемента в массив P будем использовать функцию put_lex:

```

void put_lex (struct lex l)
{
    P [ free++] = l;
}

```

Кроме того, введем модификацию этой функции – функцию put_lex5, которая записывает лексему в ПОЛИЗ, изменяя ее класс с 4-го на 5-й (с сохранением значения поля value):

```

void put_lex5 (struct lex l)
{
    l.class = 5;
    P [ free++] = l;
}

```

Пусть есть функция

```
struct lex make_or(char *op),
```

которая по символьному изображению операции op находит в таблице ограничителей соответствующую строку и формирует лексему вида (2, *i*), где *i* – номер найденной строки.

Генерация внутреннего представления программы будет проходить во время синтаксического анализа параллельно с контролем контекстных условий, поэтому для генерации можно использовать информацию, «собранную» синтаксическим и семантическим анализаторами; например, при генерации ПОЛИЗа вы-

ражений можно воспользоваться содержимым стека, с которым работает семантический анализатор.

Кроме того, можно дополнить функции семантического анализа действиями по генерации:

```
void checkop_p (void)
{
    char *op;
    char *t1;
    char *t2;
    char *res;
    t2 = spop();
    op = spop();
    t1 = spop();
    res = gettype (op,t1,t2);
    if (strcmp (res, «no»))
    {
        spush (res);
        put_lex (make_op (op));
    } /* дополнение! – операция
    op заносится в ПОЛИЗ */
    else
        ERROR();
}
```

Тогда грамматика, содержащая действия по контролю контекстных условий и переводу выражений модельного языка в ПОЛИЗ, будет такой:

$$\begin{aligned}
 E &\rightarrow E1 \mid E1 [=|>|<] \langle \text{spush (TD[curr_lex.value])} \rangle \\
 &\hspace{15em} E1 \langle \text{checkop_p()} \rangle \\
 E1 &\rightarrow T \{ [+|-|or] \langle \text{spush (TD[curr_lex.value])} \rangle \\
 &\hspace{15em} T \langle \text{checkop_p()} \rangle \} \\
 T &\rightarrow F \{ [*|/|and] \langle \text{spush (TD[curr_lex.value])} \rangle \\
 &\hspace{15em} F \langle \text{checkop_p()} \rangle \} \\
 F &\rightarrow I \langle \text{checkid(); put_lex (curr_lex)} \rangle \\
 &\hspace{15em} | N \langle \text{spush(«int»); put_lex (curr_lex)} \rangle
 \end{aligned}$$

```
[[true|false] <spush («bool»); put_lex (curr_lex)>
|not F <checknot(); put_lex (make_op («not»))> | (E)
```

Действия, которыми нужно дополнить правило вывода оператора присваивания, также достаточно очевидны:

```
S → I <checkid(); put_lex5 (curr_lex)> :=
E <eqtype(); put_lex (make_op («:=«»))>
```

При генерации ПОЛИЗа выражений и оператора присваивания элементы массива P заполнялись последовательно. Семантика условного оператора if E then S1 else S2 такова, что значения операндов для операций безусловного перехода и перехода «по лжи» в момент генерации операций еще неизвестны:

```
if (!E) goto l2; S1; goto l3; l2: S2; l3...
```

Поэтому придется запоминать номера элементов в массиве P, соответствующих этим операндам, а затем, когда станут известны их значения, заполнять пропущенное.

Пусть есть функция

```
struct lex make_labl (int k),
```

которая формирует лексему-метку ПОЛИЗа вида (0,k).

Тогда грамматика, содержащая действия по контролю контекстных условий и переводу условного оператора модельного языка в ПОЛИЗ, будет такой:

```
S → if E <eqbool(); p12 = free++; put_lex (make_op («!F»))>
then S <p13 = free++; put_lex (make_op («!»))>
P[p12] = make_labl (free) >
else S < P[p13] = make_lable (free) >
```

Замечание: переменные p12 и p13 должны быть локализованы в процедуре S, иначе возникнет ошибка при обработке вложенных условных операторов.

Аналогично можно описать способ генерации ПОЛИЗа других операторов модельного языка.

12.4. Интерпретатор ПОЛИЗа для модельного языка

Польская инверсная запись была выбрана нами в качестве языка внутреннего представления программы, в частности, потому, что записанная таким образом программа может быть легко проинтерпретирована.

Идея алгоритма очень проста: просматриваем ПОЛИЗ слева направо; если встречаем операнд, то записываем его в стек; если встретили знак операции, то извлекаем из стека нужное количество операндов и выполняем операцию, результат (если он есть) заносим в стек и т.д.

Итак, программа на ПОЛИЗе записана в массиве P; пусть она состоит из N элементов-лексем. Каждая лексема – это структура

```
structlex {intclass; intvalue;};
```

возможные значения поля class:

- 0 – лексемы-метки (номера элементов в ПОЛИЗе)
- 1 – логические константы **true** либо **false** (других лексем – служебных слов в ПОЛИЗе нет)
- 2 – операции (других лексем-ограничителей в ПОЛИЗе нет)
- 3 – целые константы
- 4 – лексемы-идентификаторы (во время интерпретации будет использоваться значение)
- 5 – лексемы-идентификаторы (во время интерпретации будет использоваться адрес).

Считаем, что к моменту интерпретации распределена память под константы и переменные, адреса занесены в поле address таблиц TID и TNUM, значения констант размещены в памяти.

В программе-интерпретаторе будем использовать некоторые переменные и функции, введенные нами ранее.

```
void interpreter(void)
{
    int *ip;
    int i, j, arg;
    for (i = 0; i<=N; i++)
```

```

{
    curr_lex = P[i];
    switch (curr_lex.class)
    {
        case0:
            ipush(curr_lex.value);
            break;
        /* метку ПОЛИЗ – в стек */
        case 1:
            if (eq («true»))
                ipush (1);
            else
                ipush (0);
                break;
        /* логическое значение – в стек */
        case 2:
            if (eq («+»))
            {
                ipush(ipop()+ipop());
            }
            break
        };
        /* выполнили операцию сложения, результат – в стек */
        if (eq («-»))
        {
            arg = ipop();
            ipush (ipop()-arg);
            break;
        }
        /*аналогично для других двуместных арифметических
        и логических операций */
        if (eq («not»))
        {
            ipush (!ipop());
            break;
        }
        if (eq («!»))

```

```

        {
            j = ipop();
            i = j-1;
            break;
        };
/* интерпретация будет продолжена с j-го элемента ПОЛИЗа */
if (eq («!F»))
    {
        j = ipop();
        arg = ipop();
if (!arg)
        {
            i = j-1
        };
        break;
    };
/* если значение arg ложно, то интерпретация будет продол-
жена с j -го элемента ПОЛИЗа, иначе порядок не изменится */
if (eq («:=«))
    {
        arg = ipop();
        ip = (int*)ipop();
*ip = arg;
        break;
    };
if (eq («R»))
    {
        ip = (*int) ipop();
scanf(«%d», ip);
        break;
    };
/* «R» – обозначение операции ввода */
if (eq («W»))
    {
        arg = ipop();

```

```

printf («%d», arg);
    break;
};
/* «W» – обозначение операции вывода */
    case 3:
        ip =
        =TNUM[curr_lex.value].address;
        ipush(*ip);
        break;
/* значение константы – в стек */
    case 4:
        ip =
        = TID [curr_lex.value].address;
        ipush(*ip);
        break;
/* значение переменной – в стек */
    case 5:
        ip =
        = TID [curr_lex.value}.address;
        ipush((int)ip);
        break;
/* адрес переменной – в стек */
    } /* конец switch*/
} /* конец for*/

```

Задачи и упражнения

1. Дана грамматика. Построить вывод заданной строки.

$$S \rightarrow T \mid T+S \mid T-S \mid bS \mid aSBC \mid abC$$

$$T \rightarrow F \mid F*T \mid CB \rightarrow BC$$

$$F \rightarrow a \mid bb \mid B \rightarrow bb$$

Строка $a - b * a + bbC \rightarrow bc$

$$cC \rightarrow cc$$

Строка $aaabbbccc$

2. Построить все сентенциальные формы для грамматики с правилами:

$$S \rightarrow A+B \mid B+A$$

$$A \rightarrow a$$

$$B \rightarrow b$$

3. Какой язык порождается грамматикой с правилами:

a) $S \rightarrow ACAB \mid S \rightarrow aBb \mid \varepsilon$

$$C \rightarrow + \mid - \mid B \rightarrow cSc$$

$$A \rightarrow a \mid b$$

с) $S \rightarrow 1Bd \mid S \rightarrow A \mid SA \mid SB$

$$B \rightarrow B0 \mid 1A \rightarrow a$$

$$B \rightarrow b$$

4. Построить грамматику, порождающую язык :

$$L = \{a^n b^m \mid n, m \geq 1\}$$

$$L = \{\alpha\beta\gamma\alpha \mid \alpha, \beta, \gamma - \text{любые строки из } a \text{ и } b\}$$

$$L = \{a_1 a_2 \dots a_n a_n \dots a_2 a_1 \mid a_i = 0 \text{ или } a_i = 1, n \geq 1\}$$

$$L = \{a^n b^m \mid n \neq m; n, m \geq 0\}$$

$$L = \{\text{строки из } 0 \text{ и } 1 \text{ с неравным числом } 0 \text{ и } 1\}$$

$$L = \{\alpha\alpha \mid \alpha \in \{a, b\}^+\}$$

$L = \{\omega \mid \omega \in \{0, 1\}^+ \text{ и содержит равное количество } 0 \text{ и } 1, \text{ причем любая подстрока, взятая с левого конца, содержит единиц не меньше, чем нулей}\}$.

$$L = \{(a^{2m} b^m)^n \mid m \geq 1, n \geq 0\}$$

$$L = \{ a^{3^n+1} \mid n \geq 1 \}$$

$$L = \{ a^{n^2} \mid n \geq 1 \}$$

$$L = \{ a^{n^3+1} \mid n \geq 1 \}$$

5. К какому типу по Хомскому относится грамматика с правилами:

- a) $S \rightarrow a \mid \text{Bab} \mid S \rightarrow \text{Ab}$
 $B \rightarrow \text{Bb} \mid \text{bA} \rightarrow \text{Aa} \mid \text{ba}$
- c) $S \rightarrow 0A1 \mid 01d \mid S \rightarrow \text{AB}$
 $0A \rightarrow 00A1 \mid \text{AB} \rightarrow \text{BA}$
 $A \rightarrow 01 \mid A \rightarrow a$
 $B \rightarrow b$

6. Эквивалентны ли грамматики с правилами :

- a) $S \rightarrow \text{AB} \mid S \rightarrow \text{AS} \mid \text{SB} \mid \text{AB}$
 $A \rightarrow a \mid \text{Aa} \mid A \rightarrow a$
 $B \rightarrow b \mid \text{Bb} \mid B \rightarrow b$
- b) $S \rightarrow \text{aSL} \mid \text{aL} \mid S \rightarrow \text{aSBc} \mid \text{abc}$
 $L \rightarrow \text{KccB} \mid \text{Bc}$
 $cK \rightarrow \text{Kcb} \mid \text{B} \rightarrow \text{bb}$
 $K \rightarrow b$

7. Построить КС-грамматику, эквивалентную грамматике с правилами:

- a) $S \rightarrow \text{aAbb} \mid S \rightarrow \text{AB} \mid \text{ABS}$
 $aA \rightarrow \text{aaAb} \mid \text{AB} \rightarrow \text{BA}$
 $A \rightarrow \varepsilon$ $\text{BA} \rightarrow \text{AB}$
 $A \rightarrow a$
 $B \rightarrow b$

8. Построить регулярную грамматику, эквивалентную грамматике с правилами:

$$\begin{aligned} \text{a) } S &\rightarrow A \mid ASb)S \rightarrow A.A \\ A &\rightarrow a \mid bbA \rightarrow B \mid BA \\ B &\rightarrow 0 \mid 1 \end{aligned}$$

9. Доказать, что грамматика с правилами:

$$\begin{aligned} S &\rightarrow aSBC \mid abC \\ CB &\rightarrow BC \\ bB &\rightarrow bb \\ bC &\rightarrow bc \\ cC &\rightarrow cc \end{aligned}$$

порождает язык $L = \{a^n b^n c^n \mid n \geq 1\}$. Для этого показать, что в данной грамматике

- 1) выводится любая строка вида $a^n b^n c^n (n \geq 1)$ и
- 2) не выводятся никакие другие строки.

10. Дана грамматика с правилами:

$$\begin{aligned} \text{a) } S &\rightarrow S0 \mid S1 \mid D0 \mid D1b)S \rightarrow \text{if } B \text{ then } S \mid B = E \\ D &\rightarrow H. \quad \quad \quad E \rightarrow B \mid B + E \\ H &\rightarrow 0 \mid 1 \mid H0 \mid H1 \quad B \rightarrow a \mid b \end{aligned}$$

Построить восходящим и нисходящим методами дерево вывода для строки:

$$\begin{aligned} &10.1001 \\ \text{if then } b &= a+b+b \end{aligned}$$

11. Определить тип грамматики. Описать язык, порождаемый этой грамматикой. Написать для этого языка КС-грамматику.

$$\begin{aligned} S &\rightarrow P \perp \\ P &\rightarrow 1P1 \mid 0P0 \mid T \\ T &\rightarrow 021 \mid 120R \\ R1 &\rightarrow 0R \\ R0 &\rightarrow 1 \\ R \perp &\rightarrow 1 \perp \end{aligned}$$

12. Построить регулярную грамматику, порождающую строки в алфавите $\{a,b\}$, в которых символ a не встречается два раза подряд.

13. Написать КС-грамматику для языка L , построить дерево вывода и левосторонний вывод для строки $aabbbccccc$: $L = \{a^{2n}b^m c^{2k} \mid m=n+k, m>1\}$.

14. Построить грамматику, порождающую сбалансированные относительно круглых скобок строки в алфавите $\{a, (,), \perp\}$. Сбалансированную строку α определим рекурсивно так: строка α сбалансирована, если

α не содержит скобок,

$\alpha = (\alpha_1)$ или $\alpha = \alpha_1\alpha_2$, где α_1 и α_2 сбалансированы.

15. Написать КС-грамматику, порождающую язык L , и вывод для строки $aacbbbaa$ в этой грамматике.

$$L = \{a^n c b^m c a^n \mid n, m > 0\}.$$

16. Написать КС-грамматику, порождающую язык L , и вывод для строки 110000111 в этой грамматике.

$$L = \{1^n 0^m 1^p \mid n+p > m; n, p, m > 0\}.$$

17. Дана грамматика G . Определить ее тип; язык, порождаемый этой грамматикой; тип языка.

$$\begin{aligned} G: \quad S &\rightarrow 0A1 \\ 0A &\rightarrow 00A1 \\ A &\rightarrow \varepsilon \end{aligned}$$

18. Даны язык $L = \{1^{3n+2}0^n \mid n \geq 0\}$. Определить его тип, написать грамматику, порождающую L . Построить левосторонний и правосторонний выводу, дерево разбора для строки 1111111100 .

19. Привести пример грамматики, все правила которой имеют вид $A \rightarrow Bt$, либо $A \rightarrow tB$, либо $A \rightarrow t$, для которой не существует эквивалентной регулярной грамматики.

20. Написать общие алгоритмы построения по данным КС-грамматикам G_1 и G_2 , порождающим языки L_1 и L_2 , КС-грамматики для

$$L_1 \cup L_2$$

$$L_1 \cdot L_2$$

$$L_1^*$$

Замечание: $L = L_1 \cdot L_2$ – это сцепление языков L_1 и L_2 , т.е. $L = \{ \alpha\beta \mid \alpha \in L_1, \beta \in L_2 \}$;

$L = L_1^*$ – это замыкание Клини языка L_1 , то есть

$$L_1^* = \{ \varepsilon \} \cup L_1 \cup L_1 \cdot L_1 \cup L_1 \cdot L_1 \cdot L_1 \cup \dots$$

21. Написать КС-грамматику для $L = \{ \omega_i 2 \omega_{i+1}^R \mid i \in \mathbb{N}, \omega_i = (i)_2 \}$ – двоичное представление числа i , ω^R – обращение строки ω . Написать КС-грамматику для языка L^* (см. задачу 20).

22. Показать, что грамматика

$$E \rightarrow E+E \mid E^*E \mid (E) \mid i$$

неоднозначна. Как описать этот же язык с помощью однозначной грамматики?

23. Показать, что наличие в КС-грамматике правил вида

$$A \rightarrow AA \mid \alpha$$

$$A \rightarrow A\alpha A \mid \beta$$

$$A \rightarrow \alpha A \mid A\beta \mid \gamma$$

где $\alpha, \beta, \gamma \in (T \cup N)^*$, $A \in N$, делает ее неоднозначной. Можно ли преобразовать эти правила таким образом, чтобы полученная эквивалентная грамматика была однозначной?

24. Показать, что КС-грамматика G неоднозначна.

$$G: \quad S \rightarrow abC \mid aB$$

$$B \rightarrow bc$$

$$C \rightarrow bc$$

25. Дана КС-грамматика $G = \{N, T, S, P\}$. Предложить алгоритм построения множества

$$X = \{A \mid A \in N, A \Rightarrow^* \varepsilon\}.$$

26. Для произвольной КС-грамматики G предложить алгоритм, определяющий, пуст ли язык $L(G)$.

27. Написать приведенную грамматику, эквивалентную данной: а) $S \rightarrow aABS \mid bCACdb$ б) $S \rightarrow aAB \mid E$

$$A \rightarrow bAB \mid cSA \mid cCCA \rightarrow dDA \mid \varepsilon$$

$$B \rightarrow bAB \mid cSBB \rightarrow bE \mid f$$

$$C \rightarrow cS \mid cC \rightarrow cAB \mid dSD \mid a$$

$$D \rightarrow eA$$

$$E \rightarrow fA \mid g$$

28. Язык называется распознаваемым, если существует алгоритм, который за конечное число шагов позволяет получить ответ о принадлежности любой строки языку. Если число шагов зависит от длины строки и может быть оценено до выполнения алгоритма, язык называется легко распознаваемым. Доказать, что язык, порождаемый неукорачивающей грамматикой, легко распознаваем.

29. Доказать, что любой конечный язык, в который не входит пустая строка, является регулярным языком.

30. Доказать, что нециклическая КС-грамматика порождает конечный язык.

Нетерминальный символ $A \in N$ – циклический, если в грамматике существует вывод

$$A \Rightarrow^* \xi_1 A \xi_2.$$

КС-грамматика называется циклической, если в ней имеется хотя бы один циклический символ.

31. Показать, что условие цикличности грамматики (см. задачу 30) не является достаточным условием бесконечности порождаемого ею языка.

32. Доказать, что язык, порождаемый циклической приведенной КС-грамматикой, содержащей хотя бы один эффективный циклический символ, бесконечен.

Циклический символ называется эффективным, если

$$A \Rightarrow^* \alpha A \beta,$$

где $|\alpha A \beta| > 1$; иначе циклический символ называется *фиктивным*.

33. Дана регулярная грамматика с правилами:

$$S \rightarrow S0 \mid S1 \mid P0 \mid P1$$

$$P \rightarrow N.$$

$$N \rightarrow 0 \mid 1 \mid N0 \mid N1.$$

Построить по ней диаграмму состояний ДС и использовать ДС для разбора строк: 11.010, 0.1, 01., 100. Какой язык порождает эта грамматика?

34. Дана ДС.

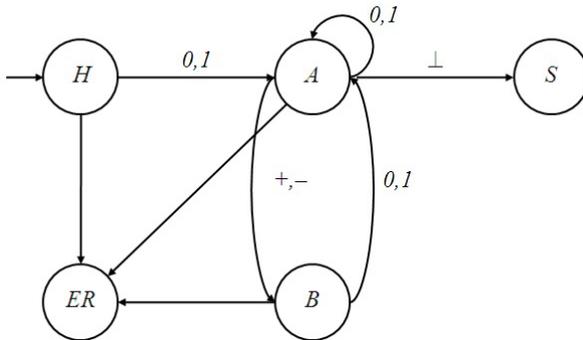


Рис. 38. Диаграмма состояний

- Осуществить разбор строк 1011, 10+011 и 0-101+1.
- Восстановить регулярную грамматику, по которой была построена данная ДС.
- Какой язык порождает полученная грамматика?

35. Пусть имеется переменная c и функция $gc()$, считывающая в c очередной символ анализируемой строки. Дана ДС с действиями:

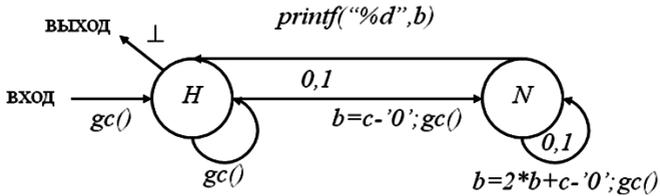


Рис. 39. Диаграмма состояний с действиями

а) Определить, что будет выдано на печать при разборе строки $1+101//p11+++1000/5⊥$?

б) Написать на Си анализатор по этой ДС.

36. Построить регулярную грамматику, порождающую язык

$$L = \{(abb)^k \perp \mid k \geq 1\},$$

по ней построить ДС, а затем по ДС написать на Си анализатор для этого языка.

37. Построить ДС, по которой в заданном тексте, оканчивающемся на \perp , выявляются все парные комбинации $\langle \rangle$, $\langle =$ и $\rangle =$ и подсчитывается их общее количество.

38. Дана регулярная грамматика:

$$S \rightarrow A \perp$$

$$A \rightarrow Ab \mid Bb \mid b$$

$$B \rightarrow Aa$$

Определить язык, который она порождает; построить ДС; написать на Си анализатор.

39. Написать на Си анализатор, выделяющий из текста вещественные числа без знака (они определены, как в Паскале) и преобразующий их из символьного представления в числовое.

40. Даны две грамматики G_1 и G_2 :

$G_1: S \rightarrow 0C \mid 1B \mid \varepsilon$
 $G_2: S \rightarrow 0D \mid 1B$
 $B \rightarrow 0B \mid 1C \mid \varepsilon$ $B \rightarrow 0C \mid 1C$
 $C \rightarrow 0C \mid 1CC \rightarrow 0D \mid 1D \mid \varepsilon$
 $D \rightarrow 0D \mid 1D$

и порождаемые ими языки: $L_1 = L(G_1)$, $L_2 = L(G_2)$.

Построить регулярную грамматику для:

- a) $L_1 \cup L_2$
- b) $L_1 \cap L_2$

Если разбор по ней оказался недетерминированным, найти эквивалентную ей грамматику, допускающую детерминированный разбор.

41. Написать левостороннюю регулярную грамматику, эквивалентную данной правосторонней, допускающую детерминированный разбор.

a) $S \rightarrow 0S \mid 0Bb$
 $S \rightarrow aA \mid aB \mid bA$
 $B \rightarrow 1B \mid 1CA \rightarrow bS$
 $C \rightarrow 1C \mid \perp$ $B \rightarrow aS \mid bB \mid \perp$

c) $S \rightarrow aBd$
 $S \rightarrow 0B$
 $B \rightarrow aC \mid aD \mid dBB \rightarrow 1C \mid 1S$
 $C \rightarrow aBC \rightarrow \perp$
 $D \rightarrow \perp$

42. Для данной грамматики

- a) определить ее тип;
- b) найти язык, который она порождает;
- c) написать Р-грамматику, почти эквивалентную данной;
- d) построить ДС и анализатор на Си.

$S \rightarrow 0S \mid S0 \mid D$
 $D \rightarrow DD \mid 1A \mid \varepsilon$
 $A \rightarrow 0B \mid \varepsilon$
 $B \rightarrow 0A \mid 0$

43. Написать анализатор по следующей грамматике:

$$\begin{aligned} \text{a)} \quad & S \rightarrow C \perp & \text{b)} S \rightarrow C \perp \\ & B \rightarrow B1 \mid 0 \mid D0 \quad C \rightarrow B1 \\ & C \rightarrow B1 \mid C1 \quad B \rightarrow 0 \mid D0 \\ & D \rightarrow D0 \mid 0 \quad D \rightarrow B1 \end{aligned}$$

$$\begin{aligned} \text{c)} \quad & S \rightarrow A0 \\ & A \rightarrow A0 \mid S1 \mid 0 \end{aligned}$$

44. Грамматика G определяет язык $L = L_1 \cup L_2$, причем $L_1 \cap L_2 = \emptyset$. Написать регулярную грамматику G_1 , которая порождает язык $L_1 \cdot L_2$ (см. задачу 20). Для нее построить ДС и анализатор.

$$\begin{aligned} S &\rightarrow 0A \mid 1S \\ A &\rightarrow 0A \mid 1B \\ B &\rightarrow 0B \mid 1B \mid \perp \end{aligned}$$

45. Даны две грамматики G_1 и G_2 , порождающие языки L_1 и L_2 . Построить регулярные грамматики для

- $L_1 \cup L_2$
- $L_1 \cap L_2$
- $L_1 \cdot L_2$ (см. задачу 20)

$$\begin{aligned} G_1: S &\rightarrow 0B \mid 1SG_2: S \rightarrow B \perp \\ B &\rightarrow 0C \mid 1B \mid \varepsilon & A &\rightarrow B1 \mid 0 \\ C &\rightarrow 0B \mid 1SB \rightarrow A1 \mid C1 \mid B0 \mid 1 \\ C &\rightarrow A0 \mid B1 \end{aligned}$$

Для грамматики b) построить ДС и анализатор.

46. По данной грамматике G_1 построить регулярную грамматику G_2 для языка $L_1 \cdot L_1$ (см. задачу 20), где $L_1 = L(G_1)$; по грамматике G_2 – ДС и анализатор.

$$\begin{aligned} G_1: S &\rightarrow 0S \mid 0B \\ & B \rightarrow 1B \mid 1C \\ & C \rightarrow 1C \mid \varepsilon \end{aligned}$$

47. Написать регулярную грамматику, порождающую язык:

a) $L = \{\omega \perp \mid \omega \in \{0,1\}^*, \text{ где за } 1 \text{ непосредственно следует } 0\}$;

b) $L = \{1\omega 1 \perp \mid \omega \in \{0,1\}^+, \text{ где между вхождениями } 1 \text{ нечетное количество } 0\}$; по ней построить ДС, а по ДС написать на Си анализатор.

48. Построить лексический блок (преобразователь) для кода Морзе. Входом служит последовательность «точек», «тире» и «пауз», например,

..-- .- ...- \perp .

Выходом являются соответствующие буквы, цифры и знаки пунктуации. Особое внимание обратить на организацию таблицы.

49. Написать на Си анализатор, действующий методом рекурсивного спуска, для грамматики:

$S \rightarrow E \perp$

$E \rightarrow () \mid (E \{, E\}) \mid A$

$A \rightarrow a \mid b$

b) $S \rightarrow P := E \mid \text{if } E \text{ then } S \mid \text{if } E \text{ then } S \text{ else } S$

$P \rightarrow I \mid I(e)$

$E \rightarrow T \{+T\}$

$T \rightarrow F \{*F\}$

$F \rightarrow P \mid (E)$

$I \rightarrow a \mid b$

c) $S \rightarrow \text{type } I = T \{; I = T\} \perp$

$T \rightarrow \text{int} \mid \text{record } I: T \{; I: T\} \text{ end}$

$I \rightarrow a \mid b \mid c$

d) $S \rightarrow P = E \mid \text{while } E \text{ do } S$

$P \rightarrow I \mid I(E \{, E\})$

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow P \mid (E)$

50. Написать для заданной грамматики процедуры анализа методом рекурсивного спуска, предварительно преобразовав ее.

a) $S \rightarrow E \perp$ b) $S \rightarrow E \perp$
 $E \rightarrow E+T \mid E-T \mid TE \rightarrow E+T \mid E-T \mid T$
 $T \rightarrow T*P \mid PT \rightarrow T*F \mid T/F \mid F$
 $P \rightarrow (E) \mid IF \rightarrow I \mid I^N \mid (E)$
 $I \rightarrow a \mid b \mid c \mid I \rightarrow a \mid b \mid c \mid d$
 $N \rightarrow 2 \mid 3 \mid 4$

c) $F \rightarrow \text{function } I(I) S; I:=E \text{ end}$ d) $S \rightarrow P:=E \mid \text{while } E \text{ do } S$
 $S \rightarrow ; I:=E S \mid \varepsilon P \rightarrow I \mid I(E \{, E\})$
 $E \rightarrow E*I \mid E+I \mid I E \rightarrow E + T \mid T$
 $T \rightarrow T*F \mid F$
 $F \rightarrow P \mid (E)$

51. Восстановить КС-грамматику по функциям, реализующим синтаксический анализ методом рекурсивного спуска.

```
#include <stdio.h>
int c;
FILE *fp;
void A();
void ERROR();
void S (void)
{
    if (c == 'a')
    {
        c = fgetc(fp);
        S();
        if (c == 'b')
            c = fgetc(fp);
        else
            ERROR();
    }
    else A();
}
```

```

void A (void)
{
    if (c == 'b')
        c = fgetc(fp);
    else ERROR();
    while (c == 'b')
        c = fgetc(fp);
}
void main()
{
    fp = fopen("data", "r");
    c = fgetc(fp);
    S();
    printf("O.K.!\n");
}

```

Какой язык она порождает?

52. Предложить алгоритм, использующий введенные ранее преобразования, позволяющий в некоторых случаях получить грамматику, к которой применим метод рекурсивного спуска.

53. Какой язык порождает заданная грамматика? Провести анализ строки $(a,(b,a),(a,(b)),b)\perp$.

$$S \rightarrow \langle k = 0 \rangle E \perp$$

$$E \rightarrow A \mid (\langle k=k+1; \text{if } (k == 3) \text{ ERROR();} \rangle E \{,E\}) \langle k = k-1 \rangle$$

$$A \rightarrow a \mid b$$

54. Дана грамматика, описывающая строки в алфавите $\{0, 1, 2, \perp\}$:

$$S \rightarrow A \perp$$

$$A \rightarrow 0A \mid 1A \mid 2A \mid \varepsilon$$

Дополнить эту грамматику действиями, исключаяющими из языка все строки, содержащие подстроки 002.

55. Дана грамматика, описывающая строки в алфавите $\{a, b, c, \perp\}$:

$$S \rightarrow A\perp$$

$$A \rightarrow aA \mid bA \mid cA \mid \varepsilon$$

Дополнить эту грамматику действиями, исключаящими из языка все строки, в которых не выполняется хотя бы одно из условий:

- 1) в строку должно входить не менее трех букв c ;
- 2) если встречаются подряд две буквы a , то за ними обязательно должна идти буква b .

56. Дана грамматика, описывающая строки в алфавите $\{0, 1\}$:

$$S \rightarrow 0S \mid 1S \mid \varepsilon$$

Дополнить эту грамматику действиями, исключаящими из языка любые строки, содержащие подстроку 101.

57. Написать КС-грамматику с действиями для порождения $L = \{a^m b^n c^k \mid m+k = n \text{ либо } m - k = n\}$.

58. Написать КС-грамматику с действиями для порождения языка

$$L = \{1^n 0^m 1^p \mid n+p > m, m \geq 0\}.$$

59. Дана грамматика с семантическими действиями:

$$S \rightarrow \langle A = 0; B = 0 \rangle L \{L\} \langle \text{if } (A > 5) \text{ ERROR()} \rangle \perp$$

$$L \rightarrow a \langle A = A+1 \rangle \mid b \langle B = B+1; \text{if } (B > 2) \text{ ERROR()} \rangle$$

$$\mid c \langle \text{if } (B == 1) \text{ ERROR()} \rangle$$

Какой язык описывает эта грамматика ?

60. Дана грамматика:

$$S \rightarrow E\perp$$

$$E \rightarrow () \mid (E \{, E\}) \mid A$$

$$A \rightarrow a \mid b$$

Вставить в заданную грамматику действия, контролирующие соблюдение следующих условий:

- 1) уровень вложенности скобок не больше четырех;
- 2) на каждом уровне вложенности происходит чередование скобочных и бесскобочных элементов.

61. Пусть в языке L есть переменные и константы целого, вещественного и логического типов, а также есть оператор цикла

$$S \rightarrow \text{for } I = E \text{ step } E \text{ to } E \text{ do } S$$

Включить в это правило вывода действия, проверяющие выполнение следующих ограничений:

- 1) тип I и всех вхождений E должен быть одинаковым;
- 2) переменная логического типа недопустима в качестве параметра цикла.

Для каждой используемой процедуры привести ее текст на Си.

62. Дана грамматика

$$P \rightarrow \text{program } D; \text{ begin } S \{ ; S \} \text{ end}$$

$$D \rightarrow \text{var } D' \{ ; D' \}$$

$$D' \rightarrow I \{ , I \}; \text{ record } I: R \{ ; I: R \} \text{ end} \mid I \{ , I \} : R$$

$$R \rightarrow \text{int} \mid \text{bool}$$

$$S \rightarrow I := E \mid I.I := E$$

$$E \rightarrow T \{ +T \}$$

$$T \rightarrow F \{ *F \}$$

$$F \rightarrow I \mid (E) \mid I.I \mid N \mid L ,$$

где I – идентификатор, N – целая константа, L – логическая константа.

Вставить в заданную грамматику действия, контролирующие соблюдение следующих условий:

- 1) все переменные, используемые в выражениях и операторах присваивания, должны быть описаны и только один раз;
- 2) тип левой части оператора присваивания должен совпадать с типом его правой части.

Замечание: а) все записи считаются переменными различных типов (даже если они имеют одинаковую структуру); б) допускается присваивание записей.

63. Представить в ПОЛИЗе следующие выражения:

- a) $a+b-c$
- b) $a*b+c/a$
- c) $a/(b+c)*a$
- d) $(a+b)/(c+a*b)$
- e) a and b or c
- f) not a or b and a
- g) $x+y=x/y$
- h) $(x*x+y*y < 1)$ and $(x > 0)$

64. Для следующих выражений в ПОЛИЗе дать обычную инфиксную запись:

- a) $ab*c$
- b) $abc*/$
- c) $ab+c*$
- d) $ab+bc-/a+$
- e) a not b and not
- f) abca and or and
- g) $2x+2x*<$

65. Используя стек, вычислить следующие выражения в ПОЛИЗе:

- a) $xy*xy/+$ при $x = 8, y = 2$;
- b) $a2+b/b4*+$ при $a = 4, b = 3$;
- c) ab not and a or not при $a = b = true$;
- d) $xy*0 > y2x- <$ and при $x = y = 1$.

66. Записать в ПОЛИЗе следующие операторы языка Си и, используя стек, выполнить их при указанных начальных значениях переменных:

- a) if (x != y) x = x+1 ; при x = 3 ;
- b) if (x > y) x = y ; else y = x ; при x = 5, y = 7;
- c) while (b > a) {b = b-a;} ; при a = 3, b = 7;
- d) do {x = y; y = 2;} while (y > 9); при y = 2;
- e) S = 0; for (i = 1; i <= k; i = i + 1) {S = S + i*i;} при k = 3;
- f) switch (k)
 - {
 - case 1: a = not a; break;
 - case 2: b = a or not b ;
 - case 3: a = b ;
 - }
 при k = 2, a = b = false .

67. Используя стек, выполнить следующие действия, записанные в ПОЛИЗе, при x = 9, y = 15 (считаем, что элементы ПОЛИЗа перенумерованы с 1).

\underline{z} , x, y, *, :=, x, y, <, 30, !F, x, y, <, 23, !F, \underline{y} , y, x, -, :=, 6, !, \underline{x} , x, y, -, :=, 6, !, \underline{z} , z, x, /, :=

Описать заданные действия на Си, не используя оператор goto.

68. Записать в ПОЛИЗе следующие операторы Паскаля:

- a) for I := E1 to E2 do S
- b) case E of
 - c1: S1;
 - c2: S2;
 -
 - cn: Sn
 end;
- c) repeat S1; S2; ... ; Sn until B;

69. Записать в ПОЛИЗе следующие фрагменты программ на Паскале:

```
a) case k of
    1: begin a:=not(a or b and c); b:=a and c or b end;
    2: begin a:=a and (b or not c); b:= not a end;
    3: begin a:=b or c or not a; b:=b and c or a end
end
```

```
S:= 0; for i:= 1 to N do
    begin d:=i*2; a:=a+d*((i-1)*N+5)
    S:=-a*d+S
end
```

```
c:=a*b; while a<>b do
    if a<b then b:=b-a else a:=a-b;
    c:=c/a
```

70. Написать грамматику для выражений, содержащих переменные, знаки операций +, -, *, / и скобки (), где операции должны выполняться строго слева направо, но приоритет скобок сохраняется. Определить действия по переводу таких выражений в ПОЛИЗ.

71. Изменить приоритет операций отношения в М – языке (сделать его наивысшим). Построить соответствующую грамматику, отражающую этот приоритет. Написать синтаксический анализатор, обеспечить контроль типов, задать перевод в ПОЛИЗ.

72. Написать КС-грамматику, аналогичную данной,

$$E \rightarrow T \{+T\}$$

$$T \rightarrow F \{*F\}$$

$$F \rightarrow (E) \mid i$$

с той лишь разницей, что в новом языке будет допускаться унарный минус перед идентификатором, имеющий наивысший приоритет (например, $a*-b+-c$ допускается и означает $a*(-b)+(-c)$).

В созданную грамматику вставить действия по переводу такого выражения в ПОЛИЗ. Для каждой используемой процедуры привести ее текст на Си.

73. Дана грамматика, описывающая выражения:

$$E \rightarrow TE'$$

$$E' \rightarrow +TE' \mid \varepsilon$$

$$T \rightarrow FT'$$

$$T' \rightarrow *FT' \mid \varepsilon$$

$$F \rightarrow PF'$$

$$F' \rightarrow ^PF' \mid \varepsilon$$

$$P \rightarrow (E) \mid i$$

Включить в эту грамматику действия по переводу этих выражений в ПОЛИЗ. Для каждой используемой процедуры привести ее текст на Си.

74. Написать грамматику для выражений, содержащих переменные, знаки операций $+$, $-$, $*$, $/$, $**$ и скобки $(,)$ с обычным приоритетом операций и скобок. Включить в эту грамматику действия по переводу этих выражений в префиксную запись (операции предшествуют операндам). Предложить интерпретатор префиксной записи выражений.

75. В грамматику, описывающую выражения, включить действия по переводу выражений из инфиксной формы (операция между операндами) в функциональную запись.

Например,

$$a+b \implies +(a, b)$$

$$a+b*c \implies +(a, *(b, c))$$

76. Построить регулярную грамматику для языка L_1 , вставить в нее действия по переводу L_1 в L_2 .

$$L_1 = \{1^m 0^n \mid n, m > 0\}$$

$$L_2 = \{1^{m-n} \mid \text{если } m > n; 0 \mid \text{если } m < n; \varepsilon \mid \text{если } m = n\}.$$

(Эта задача аналогична задаче выдачи сообщений об ошибке в балансе скобок.)

77. В регулярную грамматику для языка L_1 вставить действия по переводу строк языка L_1 в соответствующие строки языка L_2 .

$$L_1 = \{1^n 0^m \mid m, n > 0\}$$

$$L_2 = \{1^m 0^{n+m} \mid m, n > 0\}$$

78. Построить регулярную грамматику для языка L_1 , вставить в нее действия по переводу строк языка L_1 в соответствующие строки языка L_2 .

$L_1 = \{b_i \mid b_i = (i)_2\}$, то есть b_i — это двоичное представление числа $i \in \mathbb{N}$

$$L_2 = \{(b_{i+1})^R \mid b_{i+1} = (i+1)_2, \omega^R \text{ — перевернутая строка } \omega\}$$

79. Построить грамматику, описывающую целые двоичные числа (допускаются незначащие нули). Вставить в нее действия по переводу этих целых чисел в четверичную систему счисления.

Литература

1. Грис, Д. Конструирование компиляторов для цифровых вычислительных машин / Д. Грис. – М. : Мир, 1975.
2. Льюис, Ф. Теоретические основы проектирования компиляторов / Ф. Льюис, Д. Розенкранц, Р. Стирнз. – М. : Мир, 1979.
3. Ахо, А. Теория синтаксического анализа, перевода и компиляции / А. Ахо, Дж. Ульман. – Т. 1, 2. – М. : Мир, 1979.
4. Вайнгартен, Ф. Трансляция языков программирования / Ф. Вайнгартен. – М. : Мир, 1977.
5. Братчиков, И. Л. Синтаксис языков программирования / И. Л. Братчиков. – М. : Наука, 1975.
6. Гинзбург, С. Математическая теория контекстно-свободных языков / С. Гинзбург. – М. : Мир, 1970.
7. Фостер, Дж. Автоматический синтаксический анализ / Дж. Фостер. – М. : Мир, 1975.
8. Лебедев, В. Н. Введение в системы программирования / В. Н. Лебедев. – М. : Статистика, 1975.
9. Волкова, И. А. Формальные грамматики и языки. Элементы теории трансляции : учебное пособие / И. А. Волкова, Т. В. Руденко. – М., 1996.
10. Хопкрофт, Дж. Введение в теорию автоматов, языков и вычислений / Дж. Хопкрофт, Р. Мотвани, Дж. Ульман. – 2-е изд. – М. ; СПб. ; Киев, 2002.
11. Соколов, В. А. Формальные языки и грамматики : курс лекций / В. А. Соколов. – Ярославль : ЯрГУ, 2003.
12. Рейуорд-Смит, В. Дж. Теория формальных языков : Вводный курс / В. Дж. Рейуорд-Смит. – М. : Радио и связь, 1988.
13. Гордеев, А. В. Системное программное обеспечение / А. В. Гордеев, А. Ю. Молчанов. – СПб. : Питер, 2001.
14. Столбоушкин, А. П. Математические основания информатики / А. П. Столбоушкин, М. А. Тайцлин. – Ч. 1. – Тверь : ТГУ, 1998.
15. Соколов, В. А. Технологии трансляции : учебное пособие / В. А. Соколов, Д. Ю. Чалый. – Ярославль : ЯрГУ, 2008.

Учебное издание

Соколов Валерий Анатольевич

Введение в теорию формальных языков

Учебное пособие

Редактор, корректор М. В. Никулина
Правка, верстка Е. Б. Половкова

Подписано в печать 08.09.2014. Формат 60×84^{1/16}.

Усл. печ. л. 12,09. Уч.-изд. л. 6,7.

Тираж 60 экз. Заказ .

Оригинал-макет подготовлен
в редакционно-издательском отделе ЯрГУ.

Ярославский государственный университет
им. П. Г. Демидова.
150000, Ярославль, ул. Советская, 14.

