Министерство образования и науки Российской Федерации
ФГАО УВО “Севастопольский государственный университет”

МЕТОДИЧЕСКИЕ УКАЗАНИЯ

к выполнению лабораторных и контрольных работ по дисциплине “Теоретические основы построения компиляторов” для студентов основного профиля 09.03.02 – “Информационные системы и технологии” всех форм обучения
Основные алгоритмы

Севастополь
2018

УДК 004.423 + 453

Методические указания к выполнению лабораторных и контрольных работ по дисциплине “Теоретические основы построения компиляторов” для студентов всех форм обучения основного профиля 09.03.02 – “Информационные системы и технологии”. Основные алгоритмы [Текст] / Разраб. В.Ю. Карлусов. – Севастополь: Изд-во СевГУ, 2018. – 44 с.

Цель методических указаний: Обеспечение студентов дидактическим материалом для качественного выполнения контрольных работ; по разделам: "формальные языки и грамматики", "элементы теории конечных автоматов", "отношения предшествования" в приложении к задачам синтаксического анализа программ на алгоритмических языках.

Методическое пособие рассмотрено и утверждено на заседании кафедры Информационных систем,
протокол № *10* от *18 мая* 20*18* г.

Рецензент Кожаев Е.А., кандидат техн. наук, доцент кафедры кибернетики и вычислительной техники.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В основе настоящих Указаний положен многолетний опыт преподавания дисциплины “Системное программирование” студентам кафедры информационных систем СевНТУ в части, касающейся синтаксического анализа.

Ряд теоретических вопросов изложен в специальной литературе на весьма высоком уровне, что, наряду с ограниченным тиражом изданий, является субъективной причиной неудовлетворительного усвоения материала отдельными студентами.

Поэтому ключевые, с точки зрения авторов данного методического указания, моменты теории формальных грамматик, конечных автоматов и синтаксического анализа изложены в доступной для среднего студента форме с использованием примеров.

Методические указания призваны обеспечить качественное проведение лабораторных занятий и вычислительного практикума. Студентам заочной формы обучения они будут пособием, чтобы оказать посильную помощь в самостоятельном изучении вопросов, связанных с теорией формальных языков и грамматик и её приложение к задачам синтаксического анализа и компиляции.

Также настоящие методические указания окажутся небесполезными при изучении дисциплины “Основы дискретной математики” в части, касающейся конечных автоматов и формальных грамматик.

1. ОБЩИЕ ПОЛОЖЕНИЯ

Алфавит (словарь) V – некоторое конечное непустое множество, его элементами являются ***символы***.

Цепочка над алфавитом (в алфавите) есть произвольная конечная последовательность символов.

Язык L – подмножество множества всевозможных цепочек, выделенное с помощью некоторого конечного множества правил.

Грамматика G – набор правил описания синтаксиса.

Порождающая грамматика – набор синтаксических правил, описывающий процедуру получения цепочек (выражений) языка.

Распознающая грамматика – набор синтаксических правил синтаксических, описывающих процедуру проверки корректности цепочек (выражений) языка.

Формальной грамматикой $G[S]$ (или просто грамматикой) называется [18] конечное непустое множество, задаваемое упорядоченной четвёркой

$$G[S] = (V_T, V_N, S, R),$$

в которой

V_T – словарь терминальных символов;

V_N – словарь нетерминальных символов;

S – нетерминальный символ, который должен появиться в левой части хотя бы одного правила (он называется аксиомой или помеченным символом);

R – конечное (счётное) множество правил грамматики (правил подстановки, продукций) вида

$$\alpha \to \beta$$

где $\alpha \in (V_N \cup V_T)^+$, $\beta \in (V_N \cup V_T)^*$, индексы означают "+" – отсутствуют пустые цепочки, "*" – есть пустые цепочки.

Объединение словарей терминалов и нетерминалов дает словарь грамматики $V = V_N \cup V_T$.

Заметим, что различные грамматики могут порождать одни те же языки.

Пример записи правил подстановки R грамматики $G[Z]$:

$\mathbf{Z} \to b\mathbf{A}|\mathbf{A}$

$\mathbf{A} \to a|a\mathbf{A}$

В записи обозначено: символ "$\to$" означает "определяется как", "это есть" либо "заменяется на"; | – "или", "(альтернатива)", служит для сокращённой записи нескольких правил с одинаковыми правыми частями; $V_T = \{a, b\}$ – терминалы, словарь терминальных символов; $V_N = \{\mathrm{Z}, \mathrm{A}\}$ – нетерминалы, словарь нетерминальных символов; Z – аксиома.

Цепочки, которые порождаются данной грамматикой суть *a, aa, aaa, a…a* или *ba*, *baa*, *ba...a*. То есть, $L\{G[Z]\} = \{a^n, ba^n, n > 0\}$.

Сентенциальная форма – это строка, порожденная аксиомой грамматики (выведенная из аксиомы).

Существуют лево- и правосторонние формы вывода, отражающие порядок применения правил грамматики к нетерминальным символам, находящимся в цепочки, при порождении очередной цепочки цепочек. Например,

$$\mathbf{S} \Rightarrow b\mathbf{A} \Rightarrow ba\mathbf{A} \Rightarrow baa,$$

(иногда сокращённо записывают $\mathbf{S} \Rightarrow^* baa$). Причём каждая цепочка является сентенциальной формой.

Предложение – сентенциальная форма, в которой все символы терминальные.

Левосторонняя форма вывода от правосторонней отличается тем, что заменяется самый левый нетерминальный символ. Вывод приведённый в примере в равной мере можно считать как левосторонним, так и правосторонним, поскольку в каждой цепочке нетерминальный символ будет единственным.

Процесс вывода удобно представлять в виде ориентированного графа, который называется ***синтаксическим деревом*** или ***деревом вывода***. Для рассмотренной сентенциальной формы оно имеет вид:

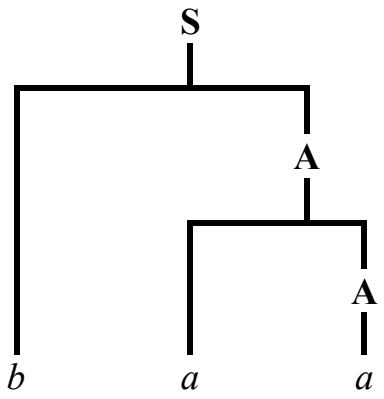


Рисунок 1.1 – Дерево вывода сентенциальной формы *baa*

Грамматики и языки классифицируются по Хомскому по виду продукций следующим образом:

0. Естественные языки

$$\alpha \to \beta, \text{ где } \alpha \in (V_N \cup V_T)^+, \ \beta \in (V_N \cup V_T)^*.$$

1. Контекстно-зависимые (контекстно-чувствительные) языки

$$\alpha\mathbf{U}\beta \to \alpha u\beta, \text{ где } \mathbf{U} \in V_N, u \in (V_N \cup V_T)^+, \alpha, \beta \in (V_N \cup V_T)^*.$$

2. Контекстно-свободные языки

$$\mathbf{U} \to u, \text{ где } \mathbf{U} \in V_N, u \in (V_N \cup V_T)^*.$$

3. Автоматные (регулярные) языки

$\mathbf{U} \to t|t\mathbf{W}$, (либо $\mathbf{U} \to \mathbf{W}t$) где $\mathbf{U}, \mathbf{W} \in V_N$, $t \in V_T$.

Грамматика является ***однозначной***, если ***для каждой*** сентенциальной формы существует ***единственное*** синтаксическое дерево, т.е. любая сентенциальная форма выводится единственным образом. Отметим, что ***порядок*** применения правил ***не существенен***.

Пример 1. Определить, однозначны ли грамматики:

а) $\mathbf{S} \to \mathbf{AB}|a$
$\mathbf{A} \to b\mathbf{A}|a$
$\mathbf{B} \to c\mathbf{A}$

б) $\mathbf{S} \to \mathbf{S}b\mathbf{S}|\mathbf{S}c\mathbf{S}|a$

Для ответа на вопрос, поставленный в задаче, воспользуемся определением: необходимо построить деревья.

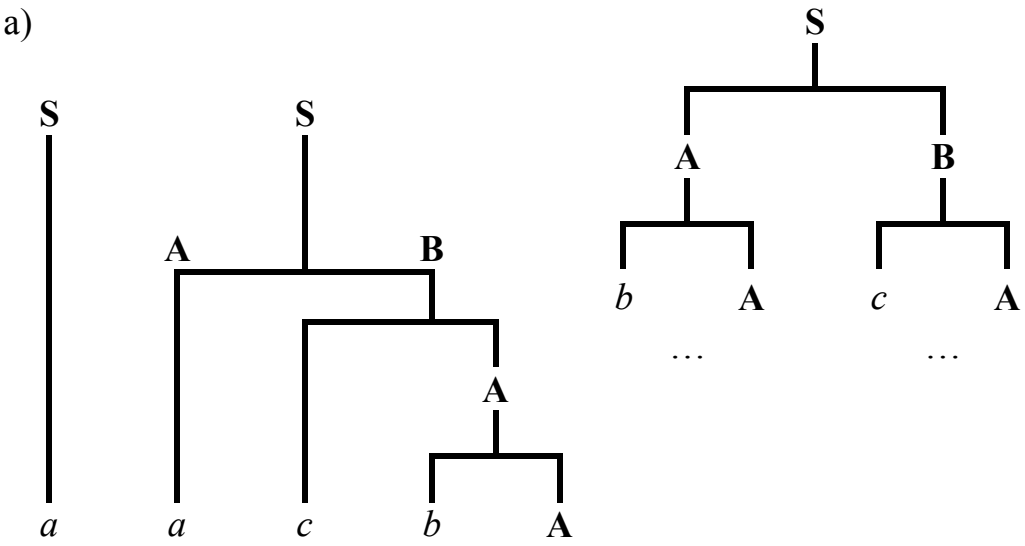


Рисунок 1.2 – Деревья выводов грамматики а)

Грамматика вида а) однозначна, а порождаемый ею язык представлен цепочками типа L = {*a*, *ac*a, *acb*...*ba*, *bb*...*bacb*...*ba*}.

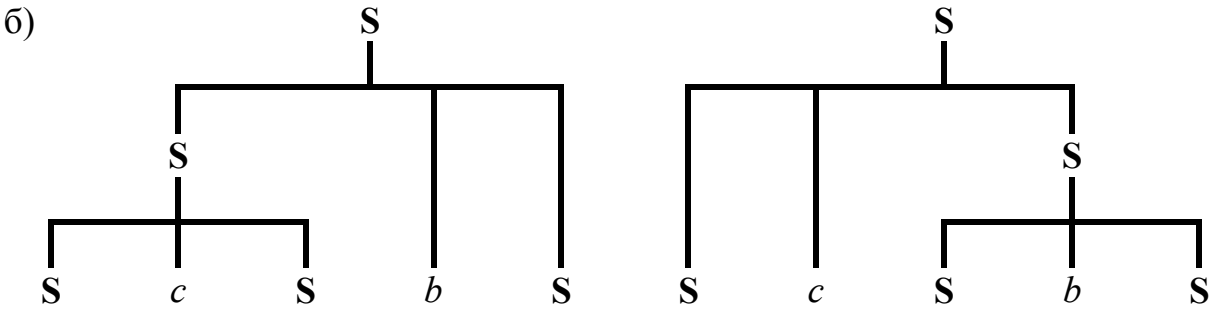


Рисунок 1.3 – Два дерева выводов сентенциальной формы **S***c***S***b***S**

Как видно из сопоставления деревьев, цепочке **S***c***S***b*S соответствуют два дерева. Следовательно, грамматика б) неоднозначна.

Пример 2. Определить, однозначны ли грамматики:

а) **S** → *b***A**|*a***B**
A → *a*|*a***S**|*b***AA**
B → *b*|*b***S**|*a***BB**

б) **S** → *a***B**|*b***AS**|*b***A**
A → *b***AA**|*a*
B → *a***BB**|*b*

Как и в предыдущем случае, воспользуемся определением

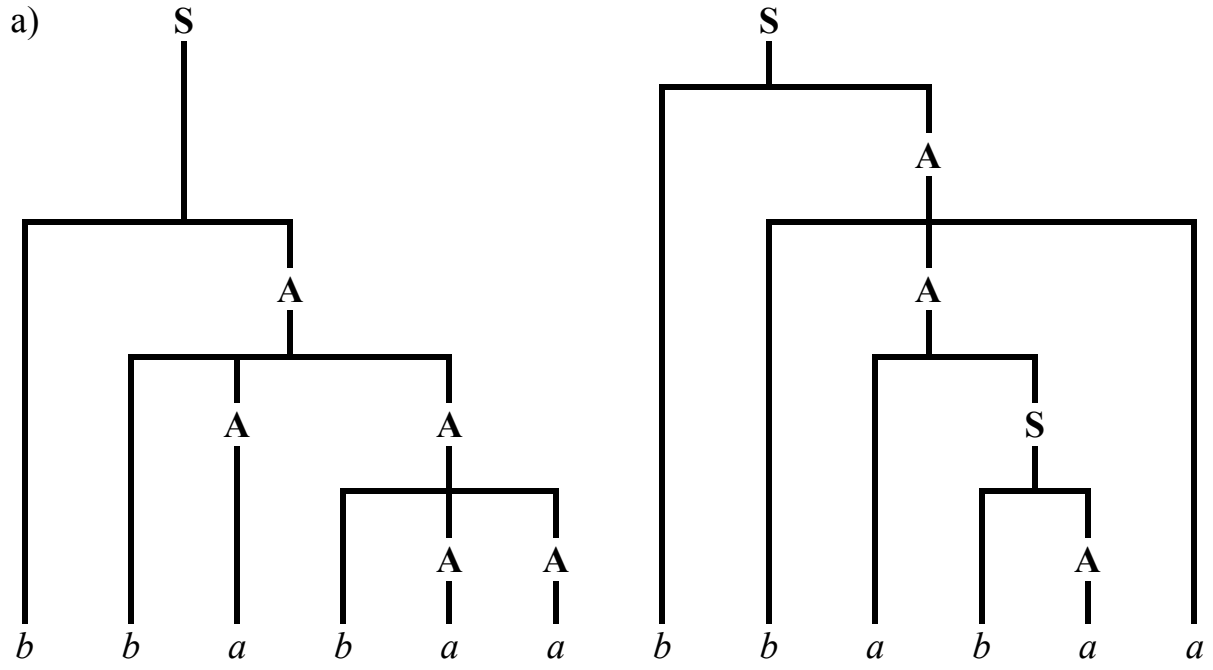


Рисунок 1.4 – Деревья выводов сентенциальной формы *bbabaa*

Представленные деревья показывают, что грамматика неоднозначна.

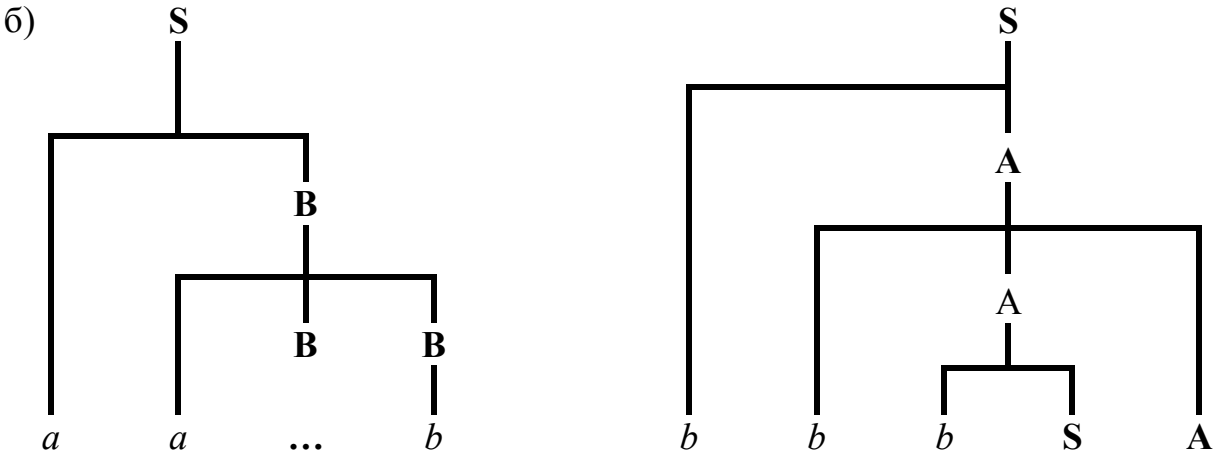


Рисунок 1.5 – Деревья выводов сентенциальных форм

А эта грамматика, напротив, является однозначной грамматикой.

2. РЕГУЛЯРНЫЕ ГРАММАТИКИ И КОНЕЧНЫЕ АВТОМАТЫ.

Согласно теории, любой регулярной грамматике соответствует конечный автомат (КА), множество входных цепочек которого соответствует множеству цепочек порождаемых грамматикой.

Конечным автоматом (КА) формально называют совокупность следующих объектов (компонентов):

$$A = (Q, \Sigma, \delta, q_0, F),$$

где Q – конечное множество неструктурируемых состояния автомата;

Σ – непустое множество (алфавит) входных символов (букв или литер);

δ – функция переходов КА, определяемая на декартовом произведении $Q \otimes \Sigma$;

$q_0 \in Q$ – начальное состояние КА;

$F \subset Q$ – множество заключительных состояний КА.

“Физический смысл” функции перехода состоит в ***определении очередного состояния*** КА в зависимости от ***символа на входе*** и ***текущего состояния*** автомата.

Конечных состояний может быть много, начальное – одно.

Существует система формальных правил, обеспечивающих соответствие КА и регулярной грамматики.

При этом предполагается представление конечного автомата в виде ориентированного графа, на котором особо обозначены начальные и конечные состояния.

По виду продукций различают праволинейную грамматику, в продукциях которой нетерминалы являются головными символоми в правых частях, и леволинейную грамматику, в правых частях правил которых нетерминальные символы находятся последними. По этой причине существуют два алгоритма преобразования.

Для праволинейной автоматной грамматики.

1. Каждый нетерминал представляется узлом или состоянием на диаграмме.

2. Каждому правилу вида $Q ::= t$, $t \in V_T$, $Q \in V_N$ соответствует дуга, направленная от начального состояния к состоянию Q, помеченная терминальным символом t.

3. Каждому правилу вида $Q ::= Rt$, $t \in V_T$, $Q, R \in V_N$ соответствует дуга, направленная от состояния R к состоянию Q, помеченная терминальным символом t.

Для леволинейной автоматной грамматики.

1. Каждый нетерминал представляется узлом или состоянием на диаграмме.

2. Каждому правилу вида $Q ::= t$, $t \in V_T$, $Q \in V_N$ соответствует дуга, направленная от состояния Q к конечному состоянию, помеченная терминальным символом t.

3. Каждому правилу вида $Q ::= tR$, $t \in V_T$, $Q, R \in V_N$ соответствует дуга, направленная от состояния Q к состоянию R, помеченная терминальным символом t.

Представление КА в виде графа привлекательно тем, что совмещает “в одном лице” все объекты, определяющие автомат.

Возможно описание КА в виде функции переходов, задаваемой как перечисления образов функции перехода $q_{i+1} = \delta(q_i, a_i)$, где q_i, q_{i+1} – суть текущее и последующее состояния КА , $a_i \in V_T$ – текущий символ на входе, иногда эта функция оформляется в виде таблицы переходов.

Существуют две разновидности КА:

- детерминированный конечный автомат (ДКА);
- недетерминированный конечный автомат (НКА).

Для ДКА выполняется условие: из текущего состояния переход по конкретному символу входного алфавита может быть осуществлен только в одно из следующих состояний. Если условие не выполняется, имеем недетерминированный КА (НКА).

Различают ***частичный*** и ***полный*** КА.

Для полного КА все образы функций переходов полностью определяются на декартовом произведении $Q \otimes \Sigma$, в противном случае КА считается частичный.

Для неопределённых образов функции переходов применяется термин “образ функции перехода пуст”.

КА функционирует по следующим правилам.

1. Исходным состоянием КА является начальное состояние q_0.

2. Цепочка, составленная из литер алфавита Σ, по одной букве поступает на вход КА. Возврат к начальным литерам цепочки не возможен.

3. Литера s_j допускается КА, находящимся в состоянии q_i, если образ функции переходов не пуст: $\delta(q_i, s_j) \neq \{\}$. Иными словами, определён переход из текущего состояния КА в следующее.

4. Если литера входной цепочки не допускается, то цепочка отвергается, признаётся не принадлежащей входному языку КА, сам автомат остаётся в текущем состоянии.

5. Цепочка литер допускается КА, если после поступления её последнего символа автомат оказывается в одном из заключительных состояний $q_r \in F$. В противном случае цепочка отвергается.

Эти особенности работы конечного автомата используется в практике синтаксического анализа для проверки корректности цепочек.

Примеры описания конечного автомата.

1. В виде таблицы переходов

	Состояния				
Символы	0	1	2	3	4
a	4	1	3	4	1
b	3	2	-	1	2
c	2	4	-	2	3

$Q = \{0, 1, 2, 3, 4\}$; $F = \{2\}$; $q_0=0$; $\Sigma = \{a, b, c\}$; функция переходов δ задана таблицей.

Так как переходы из состояния 2 по символам *b* и *c* не определены, то КА является неполным (частичным).

2. В виде функции переходов

При этом δ задаётся явно как перечисление образов

$4 = \delta(0, a)$; $3 = \delta(0, b)$; $2 = \delta(0, c)$;

$1 = \delta(1, a)$; $2 = \delta(1, b)$; $4 = \delta(1, c)$;

$3 = \delta(2, a)$;

$4 = \delta(3, a)$; $1 = \delta(3, b)$; $2 = \delta(3, c)$;

$1 = \delta(4, a)$; $2 = \delta(4, b)$; $3 = \delta(4, c)$.

3. В виде ориентированного графа

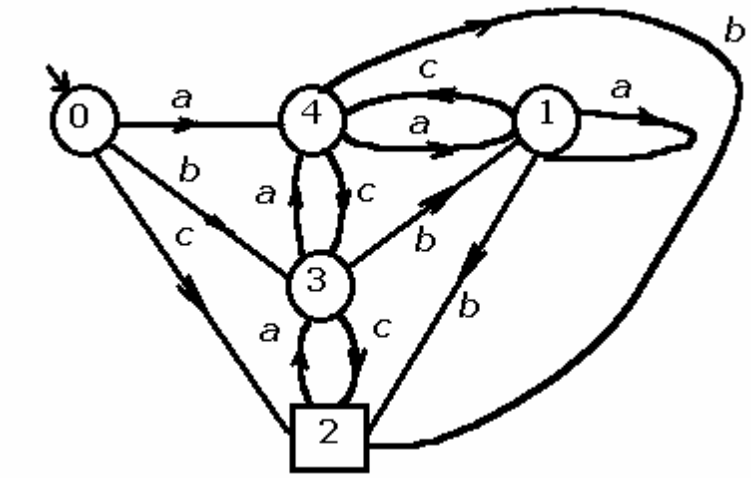


Рисунок 2.1 – Граф конечного автомата

Стрелкой на графе отмечено начальное состояние, а прямоугольником – конечное.

4. В виде регулярной грамматики

Установим следующие соответствия между номерами состояний и нетерминальными символами грамматики: 1 ~ W, 2 ~ Z, 3 ~ X, 4 ~ Y.

Воспользуемся правилами для построения праволинейной грамматики. Тогда продукции грамматики G[Z] следующие

$\mathbf{Z} \to \mathbf{X}c|\mathbf{Y}b|\mathbf{W}b|c$

$\mathbf{Y} \to a|\mathbf{X}a|\mathbf{W}c$

$\mathbf{X} \to \mathbf{Y}c|\mathbf{R}a|b$

$\mathbf{W} \to \mathbf{W}a|\mathbf{X}b|\mathbf{Y}a$

КА, построенный по формальной грамматике, может быть недетерминированным, но для реализации необходимо, чтобы автомат был детерминированным. По этой причине были разработан алгоритм построения ДКА по НКА.

Алгоритм построения ДКА, эквивалентного НКА

1. Выполняем построение таблицы переходов НКА.
2. Рассмотрим функцию переходов: если значение функции не содержится во множестве состояний автомата, то дополним это множество данной функцией.
3. Для вновь введенного состояния строим функцию переходов; анализируем функцию переходов в состояниях, определяющих данное состояние.

Определяем выходной сигнал, как дизъюнкцию выходных сигналов, входящих в данное обобщенное состояние состояний.

4. Повторяем п.п. 1 и 3 до тех пор, пока множество состояний автомата не будет изменено.
5. Если получившийся КА является частичным, то доопределим функцию перехода, вводя фиктивное состояние, соответствующее ошибке.

Замечание. Последний пункт можно выполнить как до начала применения алгоритма, так и по получнии ДКА

Пример 1. Построить ДКА для регулярной грамматики:

$\mathbf{Z} \to \mathbf{T}x|\mathbf{V}y$

$\mathbf{T} \to \mathbf{T}x|x$

$\mathbf{V} \to \mathbf{V}y|y$

Решение.

1. Грамматике соответствует граф КА, показанный на рисунке 2.2. Функция перехода его однозначна для образов $\mathbf{T}= \delta(\mathbf{S}, x)$ и $\mathbf{V} = \delta(\mathbf{S}, y)$, неоднозначна для пар $\mathbf{Z} = \delta(\mathbf{T}, x)$, $\mathbf{T} = \delta(\mathbf{T}, x)$ и $Z = \delta(\mathbf{V}, y)$, $\mathbf{V} =\delta(\mathbf{V}, y)$, а тако же не определена для образов ?= $\delta(\mathbf{T}, y)$ и ?= $\delta(\mathbf{V}, y)$.

Таким образом, заданной в условии задачи грамматике соответствует частичный НКА, так как образы функций перехода определены и не полностью, и неоднозначно.

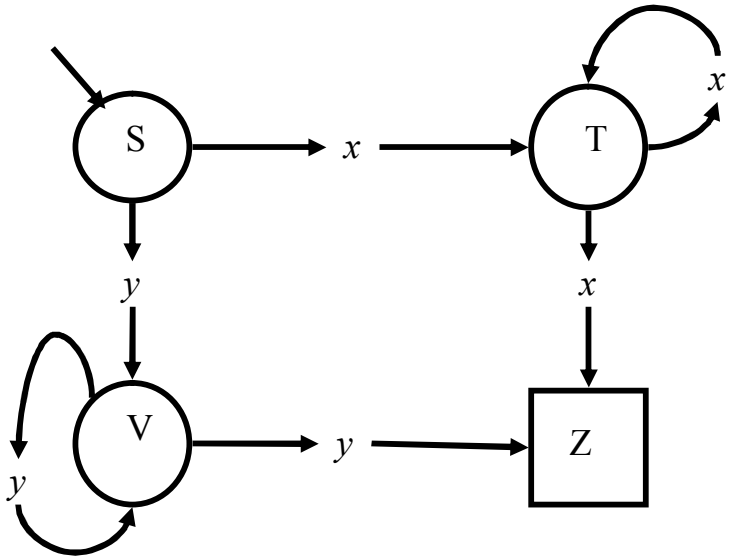


Рисунок 2.2 – Граф переходов частичного КА

2. Таблица управления КА, соответствующая функции переходов, имеет вид:

	S	**T**	**V**	**Z**
x	**T**	{**Z**, **T**}	—	—
y	**V**	—	{**Z**, **V**}	—

3. Введем фиктивное состояние E, с его помощью частичный КА будет преобразован к полному КА

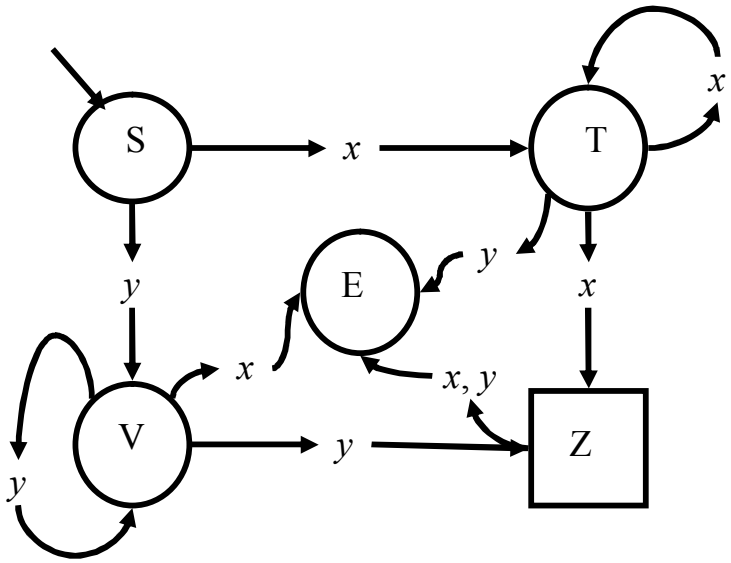


Рисунок 2.3 – Граф переходов полного КА

Для этого случая построим таблицу переходов

	S	**T**	**V**	**Z**	**{Z, T}**	**{Z, V}**	**E**
x	**T**	**{Z, T}**	**E**	**E**	**{Z, T}**	**E**	**E**
y	**V**	**E**	**{Z, T}**	**E**	**E**	{Z, T}	**E**

Так как в состояние Z нет ни одного перехода, то его можно удалить из множества внутренних состояний автомата.

Окончательно имеем:

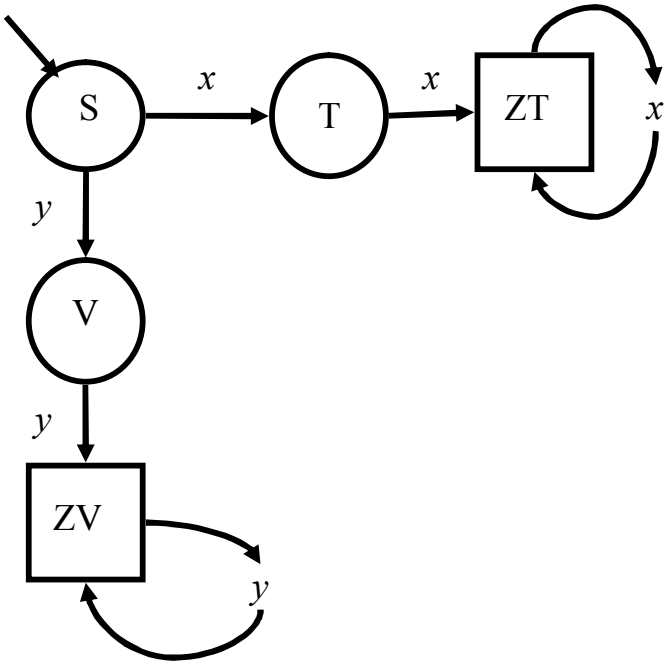


Рисунок 2.4 – Граф переходов ДКА

Пример 2. Пусть множество строк входного языка *L* определено на алфавите {0, 1} таким образом, что справа от символа 0 всегда находится символ 1. Определить ДКА, которым могут быть приняты все строки языка *L*, записать правила грамматики, их порождающей.

Решение

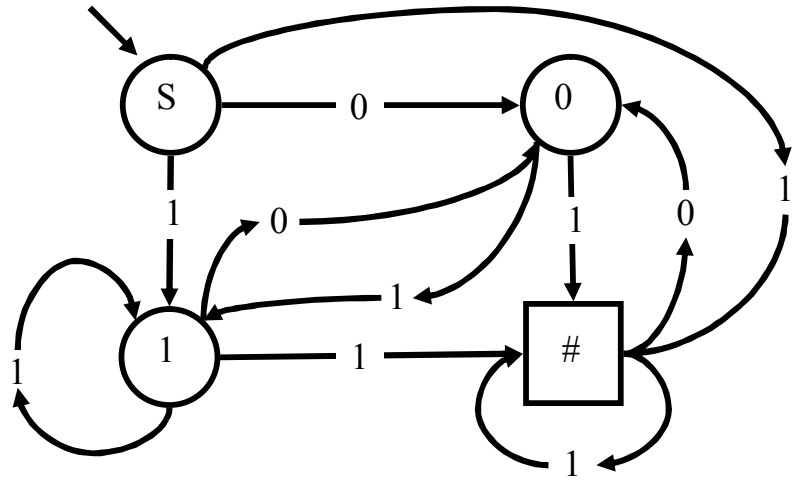


Рисунок 2.5 – Граф переходов исходного НКА

По условию задачи построим граф конечного автомата (рисунок 2.5).

Таблица переходов выглядит таким образом

	S	0	1	#	{1, #}
0	0		0	0	0
1	{1, #}	{1, #}	{1, #}	#	{1, #}

Так как состояния 1 и # не достигаются, то они удаляются из множества внутренних состояний автомата. Окончательно получим

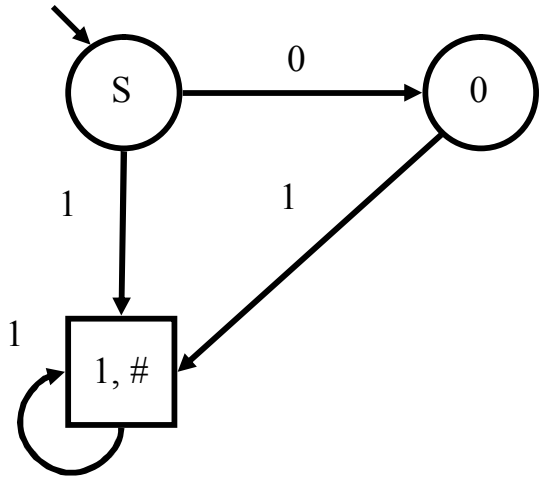


Рисунок 2.6 – Граф переходов ДКА

Таблицу переходов, соответствующую этому графу, читателю, в качестве упражнения, предлагается построить самостоятельно

Введя обозначения Z={1, #}, A={0}, сконструируем, по правилам, изложенным выше, следующую праволинейную грамматику G[Z]

Z → 1|**Z**1|**A**1

A → 0|**Z**0

Леволинейная грамматика G[S] имеет вид

S → 0A|1**Z**|1

A → 1**Z**|1.

Пример 3. Построить КА, который будет принимать любое английское слово, начинающееся на `un` и заканчивающееся на `d`. Написать на языке программирования `C` программу подсчета числа таких слов в произвольном текстовом файле.

Решение

Построим граф проверяющий слова. Его вид показан на рисунке 2.7.

На дугах графа нанесены обозначения: *d*, *n*, u - символы латинского алфавита, соответствующие написанию; *б* – любая буква латинского языка, не совпадающая с буквами {*d*, *n*, *u*}; " "– пробел между словами.
Состояние "ошибка" соответствует словам, не удовлетворяющим условию.

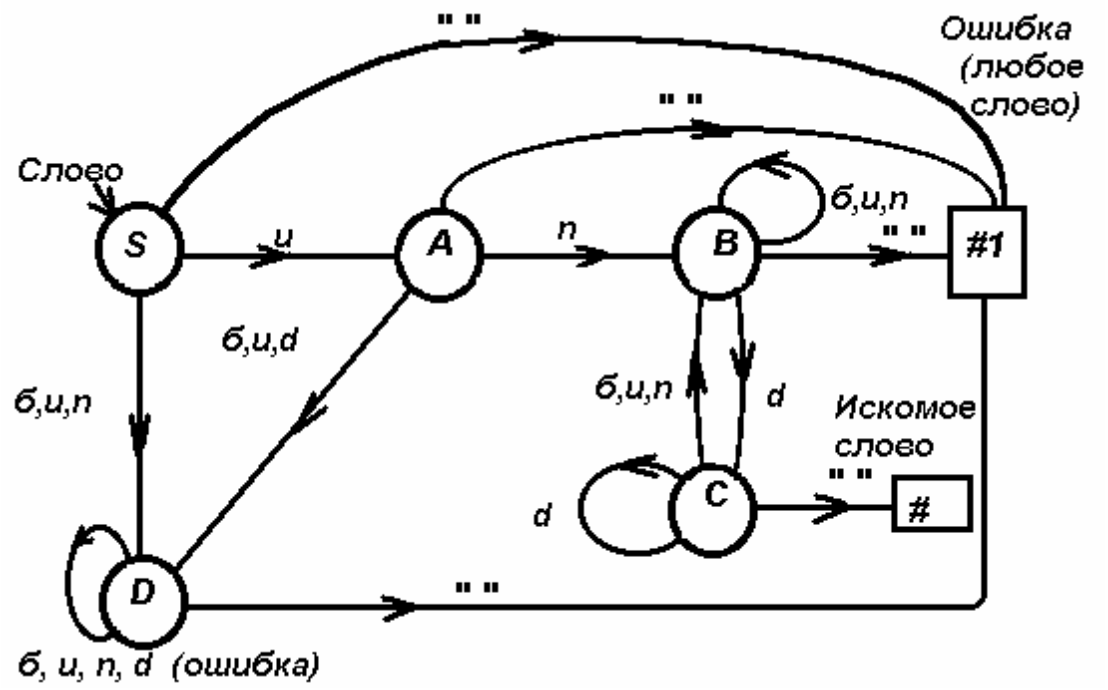


Рисунок 2.7 – Граф переходов ДКА, проверяющего слова

2. Построим по этому графу таблицу переходов.

	S	***A***	***B***	***C***	***D***	***#1***	#
u	A	D	B	B	D	-	-
n	D	B	B	B	D	-	-
d	D	D	C	C	D	-	-
" "	#1	#1	#1	#	#1	-	-
б	D	D	B	B	D	-	-

Имея в виду дальнейшую программную реализацию на языке C, введём следующую нумерацию состояний:

S ⇒ 0, A ⇒ 1, B ⇒ 2, C ⇒ 3, D ⇒ 4, #1 ⇒ 5, # ⇒6.

Нумерация выполнена для программной реализации, это позволит сэкономить количество символов в операторе инициализации таблицы переходов конечного автомата.

3. Текст программы

```
main()
{
char c;                  /* текущая литера */
int i, n;
file * text;
```

```
int mp[5][6] =   {{1, 4, 2, 2, 4}, // управляющая
                                    // таблица
                 {4, 2, 2, 2, 4},
                 {4, 4, 3, 3, 4},
                 {5, 5, 5, 6, 5},
                 {4, 4, 2, 2, 4}};
text = fopen('xx.txt', 'r'); // исходный текст в файле
i=0; n=0;        /* i – номер состояния КА, n – число
правильных слов */
while ((c = getc(text))! = EOF)
    {
    switch(c)
        {
        case 'u': i = mp[0][i]; break;
        case 'n': i = mp[1][i]; break;
        case 'd': i = mp[2][i]; break;
        case '  ': i = mp[3][i]; break;
        default:  i = mp[4][i];break;
        }
    if (i ==6) {i = 0; n++;}
    if (i ==5) {i = 0;}
    }
printf("количество символов =%d", n);
fclose(text);
}
```

3. ПОСТРОЕНИЕ МИНИМАЛЬНЫХ КОНЕЧНЫХ АВТОМАТОВ ПО РЕГУЛЯРНЫМ ВЫРАЖЕНИЯМ

Регулярное выражение – один из точных способов написания регулярной грамматики.

Правила написания регулярных выражений для типовых конструкций таковы: фигурные скобки обозначают итерацию, то, что в них заключено, может повторяться от нуля до бесчисленного числа раз. Круглые или квадратные скобки – используют для объединения альтернатив, которые разделяются символом дизъюнкции. Знак конъюнкции иногда опускают.

Пусть имеется алфавит, составленный из символов (букв, литер) $V_T = \{x_1, x_2, \ldots, x_n\}$.

1. Множество всевозможных цепочек, составленных из букв $x_i \in V_T$:

$$L = \{x_1 \vee x_2 \vee x_3 \vee ... \vee x_n\}.$$

2. Множество цепочек, составленных из литер $x_i \in V_T$ и оканчивающихся литерой x_1:

$$L = \{x_1 \vee x_2 \vee x_3 \vee ... \vee x_n\} \wedge x_1.$$

3. Множество цепочек, составленных из литер $x_i \in V_T$ начинающихся цепочкой l_1 и оканчивающихся l_2.

$$L = l_1 \wedge \{x_1 \vee x_2 \vee x_3 \vee ... \vee x_n\} \wedge l_2.$$

4. Множество однолитерных цепочек (однобуквенных слов) совпадает с алфавитом V_T:

$$L = x_1 \vee x_2 \vee x_3 \vee ... \vee x_n.$$

5. Множество двулитерных цепочек (двухбуквенных слов):

$$L = (x_1 \vee x_2 \vee x_3 \vee ... \vee x_n) \wedge (x_1 \vee x_2 \vee x_3 \vee ... \vee x_n).$$

6. Множество m – буквенных слов

$$L = (x_1 \vee x_2 \vee x_3 \vee ... \vee x_n) \underbrace{\wedge \ldots \wedge (x_1 \vee x_2 \vee x_3 \vee ... \vee x_n)}_{m-2}$$

Алгебру регулярных выражений поясним на примерах.

Пусть $L_1 = (x_1, x_1x_2)$, $L_2 = (x_3)$, $L_3 = (x_2, x_1)$. Тогда операции выполняются следующим образом.

Дизъюнкция:

$L = L_1 \mid L_2 \mid L_3 = L_1 \vee L_2 \vee L_3 = (x_1, x_1x_2, x_3, x_2)$.

Конъюнкция:

$L = L_1L_2 = L_1 \wedge L_2 = (x_1x_3, x_1x_2x_3)$.

Рекурсия:

$L = \{x_1x_2\} = (\varepsilon, x_1x_2, x_1x_2x_1x_2, x_1x_2x_1x_2 \ldots x_1x_2)$.

Регулярное выражение однозначно определяет конечный автомат.

Рассмотрим фрагменты регулярных выражений, соответствующие элементарным операциям в совокупности с продукциями автоматной грамматики, ими определяемыми.

Дизъюнкция с конъюнкцией $L = xy \vee x \vee y \vee z$

$\mathbf{B} \rightarrow \mathbf{A}y \mid x \mid y \mid z$

$\mathbf{A} \rightarrow x$

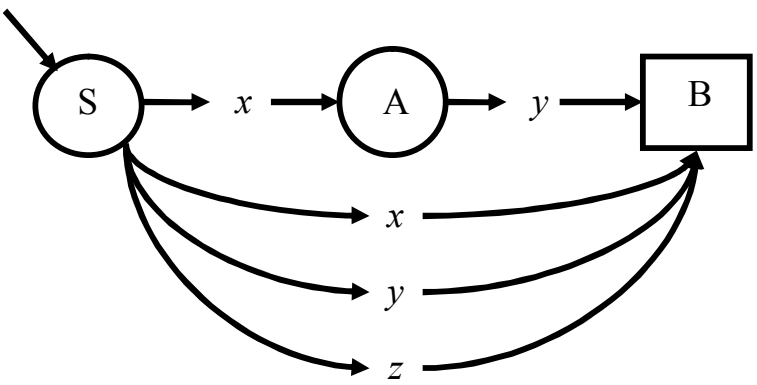


Рисунок 3.1 – Граф КА, принимающего цепочки *xy*, *x*, *y*, и *z*

Конъюнкция с рекурсией $L = x\ \{\ yx\ \}$

$A \rightarrow x|\ Bx$

$B \rightarrow Ay$

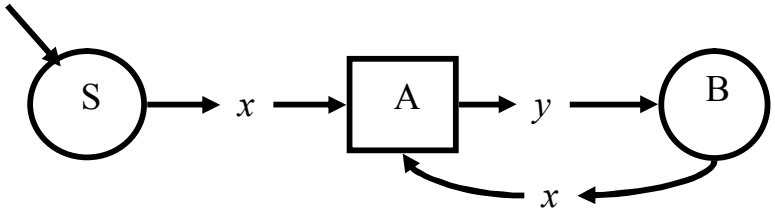


Рисунок 3.2 – Граф КА, принимающего цепочки *x*, *yx*, *yxyx*, *yx…yx*

Алгоритм построения ДКА по регулярному выражению

Алгоритм разработан профессором кафедры кибернетики и вычислительной техники Е. А. Бутаковым для построения и минимизации КА по регулярному выражению и основывается на системе индексации мест регулярного выражения.

Место регулярного выражения – это промежуток между двумя его компонентами (двумя буквами, буквой и знаком, скобкой и буквой, буквой и скобкой), а также начало и конец выражения.

Различают следующие типы мест

- Начало выражения – ***начальное*** место.
- Конец выражения – ***конечное*** место.
- ***Основное*** место – место, слева от которого находится литера, а также начальное место.
- ***Предосновное*** место – место, справа от которого находится литера.

По существу, конечный автомат переходит из предосновного места в основное место по символу (литере) находящемуся между этими местами.

Алгоритм состоит из этапов

1. Разметка мест.
2. Минимизация числа внутренних состояний автомата по разметке.
3. Построение таблицы переходов.
4. Построение ДКА из НДКА, если это необходимо.
5. Минимизация числа состояний путём объединения повторяющихся столбцов при условии равенства выходных сигналов. Конечные автоматы бывают двух видов: Мура и Мили. Они названы по именам их разработчиков. Для автомата Мили сигнал на выходе КА зависит от предыдущего входного сигнала, в отличие от автомата Мура, где такая зависимость отсутствует.

Правила разметки мест.

1. Первоначально осуществляется сквозная нумерация всех основных мест (короткие вертикальные линии на чертеже).

2. Выставляются индексы всех предосновных мест, не являющихся основными (одному месту может соответствовать несколько индексов). Правила

а) Индекс перед любыми скобками распространяется на все начальные места дизъюнктивных членов, записанных в этих скобках.

б) Индекс конечного места любого дизъюнктивного члена, заключённого в любые скобки, распространяется на место, непосердственно следующее за этими скобками.

в) Индекс места перед итерационными скобками распространяется на место, непосредственно следующее за этими скобками.

г) Индекс места за итерационными скобками распространяется в начальные места всех дизъюнктивных членов в этих скобках.

д) Индекс конечного места любого дизъюнктивного члена, заключённого в итерационные скобки, распространяется на начальные места всех дизъюнктивных членов, заключённых в эти скобки.

е) Индексы места, справа и слева от которого стоят буквы никуда не распространяется.

ж) Индекс конечного места распространяется на те же места, на которые и индекс начального места.

Совокупность индексов места ***является множеством***, следовательно, ***не содержит повторяющихся*** элементов.

Правило минимизации.

Если ***несколько предосновных*** мест ***отмечено одинаковой совокупностью индексов*** и ***справа*** от этих мест расположены ***одинаковые литеры***, то ***основные места***, расположенные ***справа*** от этих литер, ***можно отметить одинаковыми индексами***.

Пример. Определить минимальный ДКА, который принимает строки соответствующие регулярному выражению

$$S = \{b \vee ab \vee aab\}\ aaa\ \{a\}\ b$$

Пусть работа конечного автомата сопровождается следующими выходными сигналами:

x – рабочее состояние автомата;

y – встретилось сочетание “aaa”;

z – конец слова.

I-я итерация.

Разметка мест. Расстановка индексов мест проиллюстрирована на рисунке 3.3, на котором арабские цифры со скобкой отмечают этапы разметки, а буквы кириллицы со скобкой – правила, применяемые при выставлении того или иного индекса.

1) Разметка основных мест.

2) Основные места, которые совпадают с предосновными, поэтому индексы предосновных мест совпадают с индексами основных мест.

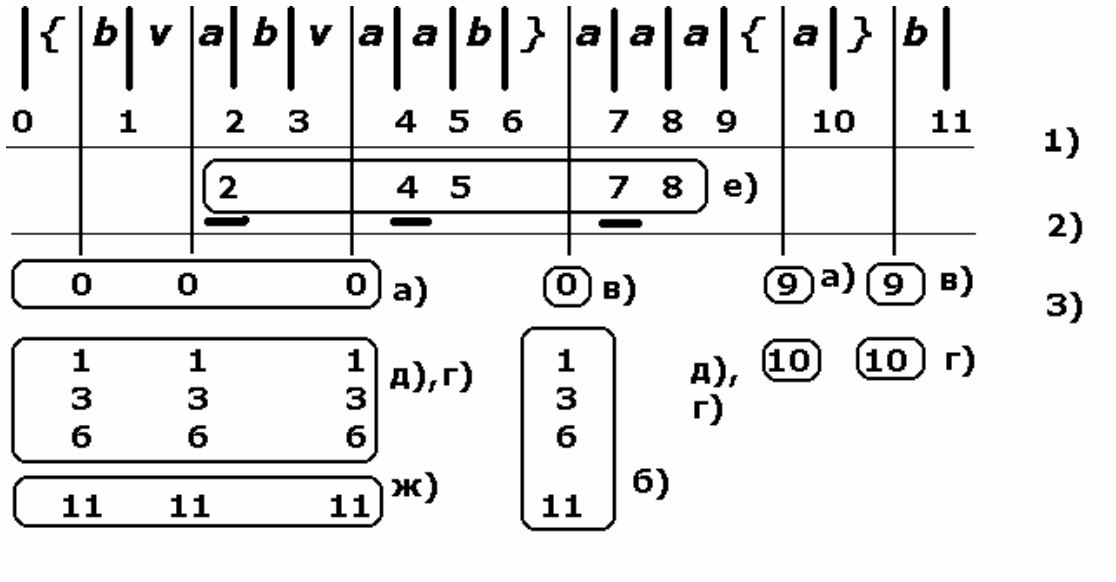


Рисунок 3.3 – Иллюстрация правил разметки мест

3) Выставление индексов предосновных мест.

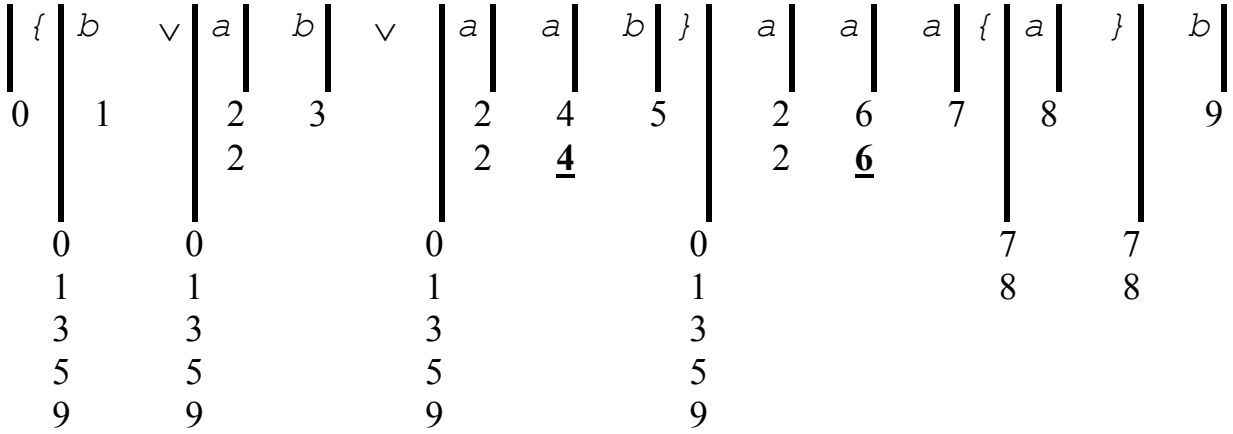
Позициями а) ... ж) обозначены правила, которыми руководствовались при разметке соответствующих мест.

Минимизация.

Под правило минимизации подпадают позиции выражения 2, 4 и 7

II-я итерация.

Разметка осуществляется с учётом эквивалентных состояний

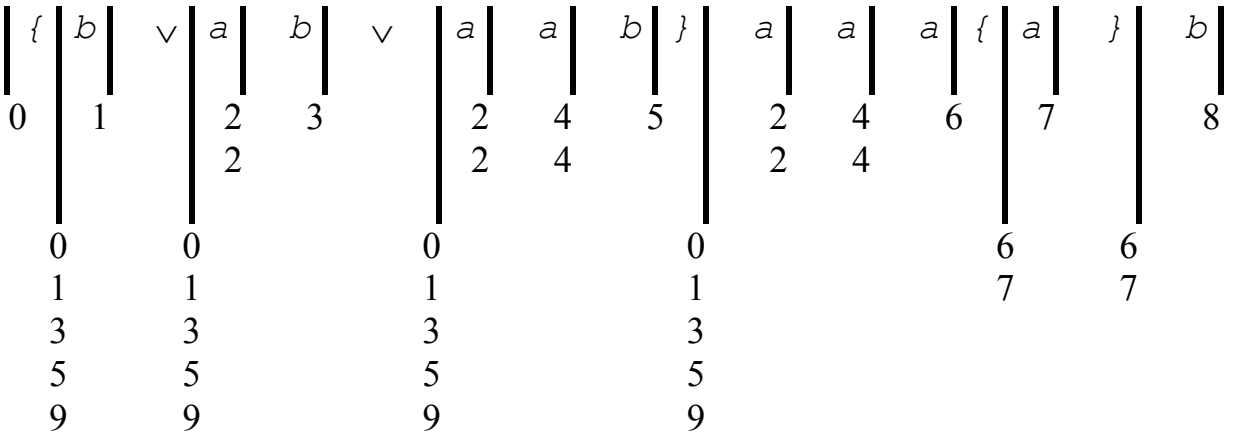


Минимизация.

Судя по разметке, эквивалентны позиции 4 и 6.

III-я итерация.

Перестоим систему разметки с учётом сокращений. Получим:



Видно, что дальнейшую минимизацию по регулярному выражению провести не удаётся.

Построим таблицу переходов КА.

Состояния автомата эквивалентны индексам основных мест регулярного выражения, а между местами располагается символ входного алфавита КА, по которому осуществляется переход.

Выходной сигнал	*x*	*x*	*x*	*x*	*x*	*x*	*y*	*x*	*z*
Состояния	0	1	2	3	4	5	6	7	8
a	2	2	4	2	6	2	7	7	2
b	1	1	3	1	5	1	8	8	1
	*	*		*		*			

Из таблицы видно (*), что состояния (0, 1, 3, 5) неразличимы ни по входному сигналу, ни по выходному. Состояние 8 является конечным согласно пункту ж) правил разметки предосновных мест, и соответствующие образы функции переходов определены так, чтобы из заключительного состояния КА мы автоматически могли перейти на обработку очередной цепочки литер.

При программной реализации КА иногда бывает целесообразным после завершения анализа входной строки принудительно возвращать его в начальное состояние, не прибегая к функции перехода.

Обозначим: A = {0, 1, 3, 5}, B = 2, C = 4, D = 6, E = 7, F = 8.

Перестроим предыдущую таблицу переходов с учётом обозначений. Получим такую таблицу переходов:

Выходной сигнал	*x*	*X*	*x*	*y*	*x*	*z*
Состояния	A	B	C	D	E	F
a	B	C	D	E	E	B
b	A	A	A	F	F	A

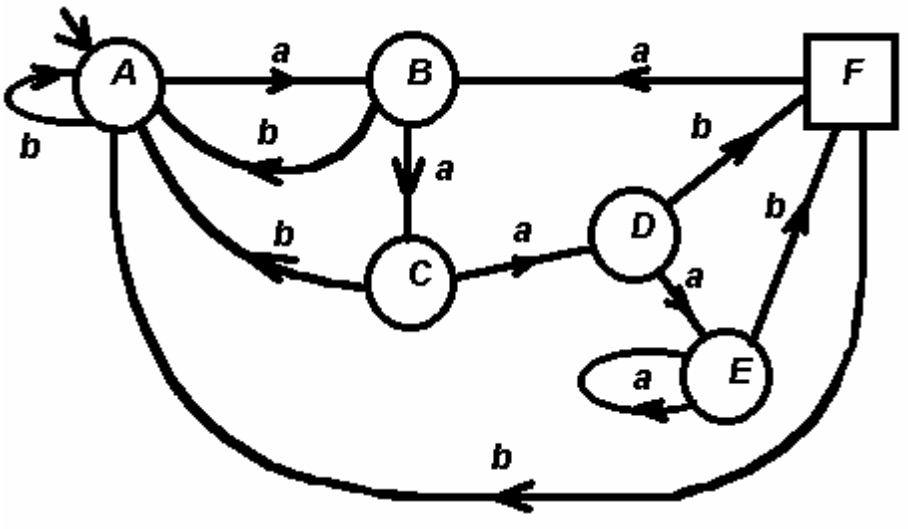


Рисунок 3.4 – Граф МКА, описываемый выражением *{b ∨ ab ∨ aab} aaa {a} b*

Упражнение для закрепления навыков

1. Построить регулярное выражение, описывающее цепочки, состоящие из символов 0 и 1, так что справа от 0 всегда находится символ 1.

Ответ: {1} {01{1}}.

2. Построить минимальный КА по регулярному выражению.

а) *a {ba ∨ b } ∨ b*

Ответ:

	0	1	2	3
a	1	-	1	#
b	3	2	2	–

б) *{ab ∨ {b}} ba ∨ b*

Ответ:

Сигнал	1	1	1	1	2	2
Состояние	0	1	2	3	4	5
a	1	-	1	5	5	–
b	4	2	3	3	3	2

в) *{{{b} a} ba}*

Ответ:

Сигнал	2	1	1	1
Состояние	0	1	2	3
a	2	2	2	0
b	3	1	3	1

4. *LL*(*k*) – ГРАММАТИКИ И НИСХОДЯЩИЙ РАЗБОР

Нисходящий разбор – иными словами разбор сверху-вниз, направленный от аксиомы грамматики к сентенциальной форме, представляющей собой анализируемое предложение языка.

При этом синтаксическое дерево строится от корня.

Существует несколько модификаций указанного метода разбора. Одна из них именуется ***рекурсивным спуском***. Рекурсивный спуск применяется для специального класса грамматик, обозначаемого аббревиатурой *LL*(*k*), от латинского Left – Left, что символизирует левосторонний ввод, левосторонний вывод.

LL(*k*) – грамматика обладает тем свойством, что любые *k* первых символов анализируемой строки однозначно определяют правило грамматики, использованное для их вывода из аксиомы.

Рекурсивный спуск заключается ***в последовательности применения процедур***, распознающих терминальные и нетерминальные символы грамматики, ***в том порядке***, в котором они размещены в правых частях соответствующих продукций.

Алгоритм тестирования грамматики на принадлежность к классу *LL*(*k*)

Для каждой *i*-ой продукции грамматики необходимо найти множество Ω_i голов цепочек длиной *k* символов, которые могут быть получены в процессе применения данного правила.

1. Исследования начинаются для случая $k = 1$.

2. Если символ, стоящий в голове правой части продукции на *k*-ой позиции терминальный, то он участвует в формировании *k*-ого символа цепочки искомой цепочки.

3. Если символ, стоящий в голове правой части продукции на *k*-ой позиции нетерминальный, то ему ставится в соответствие объединённое множество цепочек единичной длины $\Omega_i(\mathrm{N} \in V_N)_{k=1}$, сконструированное для продукций, имеющих в левой части этот нетерминал. Таким образом, формируются несколько цепочек длины *k*, число которых совпадает с мощностью множества $\Omega_i(\mathrm{N} \in V_N)_{k=1}$.

4. Если множества Ω_i, соответствующие продукциям с одинаковыми правыми частями, ***не пересекаются***, то эта грамматика принадлежит к классу *LL*(*k*) –грамматик.

Замечание. Большие значения индекса *k* нецелесообразны, поскольку вызывают необходимость просмотра контекста на глубину, равную *k*, что обуславливает непроизводительные затраты алгоритма.

Пример. Используя метод рекурсивного спуска определить алгоритм работы синтаксического анализатора для *LL*(1) грамматики:

S → *a***AB**|*b***S**

A → *a***A**|*b***S**

B → **AB**|*c*

1. Выписываем все продукции формальной грамматики (какие есть) и для каждой из них ставим в соответствие множество из голов цепочек единичной длины.

Результаты исследований представим таблично

Продукции	Цепочки единичной длины
S → *a***AB**	{ *a* }
S → *b***S**	{ *b* }
A → *a***A**	{ *a*
A → *b***S**	{ *b* }
B → **AB**	{ *a*, *b* }
B → *c*	{ *c* }

Как видно, полученные множества цепочек отвечают п. 4 алгоритма, следовательно, данная грамматика является *LL*(1) – грамматикой.

Пусть, имена процедур анализатора будут начинаться символом `P` и иметь вид `Px`, где *x* – символ алфавита, опознаваемый процедурой.

```
int pA(void), pB(void), pS(void), pa(void), pb(void),
pc(void);
char symbol;
void main(void)
{
     scanf("%S", &symbol);
     if(PS()==0) printf("\n Ошибка ");
     else printf("\n Ok ");
}
int PS(void)
{
     switch (symbol)
     {
          case 'a': return pa()*PA()*PB();
          case 'b': return pb()*PS();
          default:   return 0;
     }
}
int PA(void)
{
```

```
    switch (symbol)
    {
        case 'a': return pa()*PA();
        case 'b': return pb()*PS();
        default:   return 0;
    }
}
int PB(void)
{
    switch (symbol)
    {
        case 'a': return PA()*PB();
        case 'b': return PA()*PB();
        case 'c': return pc();
        default: return 0;
    }
}
int pa(void)
{
    if (symbol!= 'a') then return 0;
    else { scanf("%S", &symbol); return 1; }
}
int pb(void)
{
    if (symbol!= 'b') return 0;
    else { scanf("%S", &symbol); return 1; }
}
int pc(void)
{
    if (symbol!= 'c') return 0;
    else { scanf("%S", &symbol); return 1; }
}
```

Упражнения для закрепления навыков

1. Определите, какие из нижеследующих грамматик являются $LL(1)$ – грамматиками?

а) $\mathbf{S} \to \mathbf{A}|\mathbf{B}$

$\mathbf{A} \to a\mathbf{A}|a$

$\mathbf{B} \to b\mathbf{B}|b$

Ответ: не $LL(1)$

б) **S** → *a***A***a***B**|*b***A***b***B**

A → **S**|*c***B**

B → cb|a

Ответ: *LL*(1)

в) **S** → **AB**

A → **B***a*|**E**

B → **C***b*|*c*

C → *c*|**E**

Ответ: не *LL*(1)

2. Покажите, что данная грамматика является *LL*(2) – грамматикой

S → *a***AS**|**A***b***S***c*|**E**

A → *cb***A**|*a*

5. ПОСТРОЕНИЕ ОТНОШЕНИЙ ПРОСТОГО ПРЕДШЕСТВОВАНИЯ

Для эффективного выполнения восходящего разбора (разбора снизу вверх) необходимо правильно выделить простую фразу (основу) среди символов сентенциальной формы. Это ведет к тому, что по каждой паре символов R и S, стоящих рядом в сентенциальной форме, R – слева, S – справа, принять решение по ряду вопросов.

1. Стоят ли они рядом в каком-либо правиле, то есть эквивалентны (R ≡ S)?

2. Входит ли R в одну основу (является ее хвостом), а S – в другую (является ее головой) (R •>S)?

3. R не является частью основы, а S – является головой возможной основы (R<•S)?

4. Символы R и S никогда не стоят рядом в правильной программе?

Отношения <•, •>, ≡ называют отношениями предшествования, и они однозначно определяют ответ на поставленные вопросы.

Грамматику G называют грамматикой (простого) предшествования или (1,1)-предшествования, если:

а) между любыми двумя символами словаря *V* определено не более чем одно отношение предшествования.

в) ни у каких двух продукций нет одинаковых правых частей.

Алгоритм разбора, работа которого базируется на отношениях простого предшествования, производит замену основы, определяемой отношениями предшествования левой частью соответствующего правила.

На наличие основы указывает выполнение пары граничных отношений $<\bullet$ и $\bullet>$, между которыми присутствует цепочка символов, связанных отношениями ≡.

Если основу найти не удаётся, то фиксируется ошибка.

Алгоритм построения отношений простого предшествования

Для определения отношений нам необходимо построить серию булевых матриц размерностью $n \times n$, где n – мощность словаря грамматики $V. = (V_N \cup V_T)$ Индексами строк выступают “левые” символы R, а индексами столбцов – “правые” S.

1. Построим матрицу $EQ(\equiv)$. Для этого необходимо просмотреть все правые части продукций, и, для каждой пары символов, стоящих рядом, поставить единицу в соответствующей позиции матрицы.

2. Построим вспомогательную матрицу F отношения $FIRST$, которое определяется следующим образом: $F(\mathbf{R}, S) = 1$ для всех правил вида $\mathbf{R} \to s\omega$, $\mathbf{R} \in V_N, s \in V, \omega \in V^*$.

3. Построим вспомогательную матрицу L отношения $LAST$, которое определяется по правилу $L(\mathbf{R}, S)=1$, $\mathbf{R} \to \omega s$, $\mathbf{R} \in V_N, s \in V, \omega \in V^*$.

4. Проведём построение транзитивного замыкания отношений F^+ и L^+. Указанные отношения вычисляются по рекуррентному алгоритму

$A^+ = \mathrm{A}^1 \cup A^2 \cup A^3 \cup \ldots;$

$A^1 = A, A^2 = A \times A, A^{\mathrm{n}} = A^{n-1} \times A$

вручную, либо, используя ЭВМ по алгоритму Воршалла (Warshall S.)

```
for (i=0; i<n-1; i++)
    for (j=0; j<n-1; j++)
        if (A[j][i] = = 1)
            for (k = 0; k<n-1; k++)
                A[j][kj] || = A[i][k];
```

Где || – оператор булева суммирования.

5. Вычисляем отношение предшествования $<\bullet$ как матричное выражение $EQ \times F^+$.

6. Вычисляем отношение предшествования $\bullet>$ как матричное выражение $(L^+)^T \times EQ \times (I + F^+)$, где I – единичная матрица.

7. На основании сопоставления всех трех матриц $<\bullet$, $\bullet>$ и ≡ определяем единственность отношений для каждой пары символов.

Иногда удается построить функции предшествования f и g. Это ведет к уменьшению размерности матрицы с $n \times n$ до пары одномерных массивов (векторов) $n \times 1$ и $1 \times n$.

Функции предшествования обладают следующими свойствами:

$$R \equiv S \Rightarrow f(R) = g(S),$$

$$R <\bullet S \Rightarrow f(R) < g(S),$$

$$R \bullet > S \Rightarrow f(R) > g(S),$$

а их построение называется ***линеаризацией*** матрицы предшествования. Функции f и g, строго говоря, не единственны для матрицы предшествования, если удается отыскать хотя бы пару таких, то существует бесконечное количество функций предшествования. Однако есть матрицы предшествования, для коих линеаризация невозможна.

Отметим, что замена матрицы предшествования функциями ведет к частичной потери информации, что обуславливает более позднее обнаружение ошибки при использовании функциональной зависимости, нежели матрицы.

Алгоритм построения функций предшествования

1. Конструируется блочная матрица вида

$$B = \begin{bmatrix} 0 & GE \\ LE^T & 0 \end{bmatrix},$$

в которой GE – объединённая матрица отношений $\bullet >$ и $\equiv$, LE^T – транспонированная объединённая матрица отношений $< \bullet$ и $\equiv$.

2. Находится транзитивно-рефлексивное замыкание $B^* = I + B^+$, где I – единичная матрица, B^+ – транзитивное замыкание B.

3. Подсчёт единиц в разных частях результирующей матрицы B^* позволяет определить значения табличных функций предшествования $f(R_i)$ и $g(S_i)$.

$$F_i = f(R_i) = \sum_{j=1}^{2n} B^*_{i,j}; \quad i = \overline{1,n};$$

$$G_i = g(S_i) = \sum_{i=1}^{2n} B^*_{n+i,j}; \; i = \overline{1,n}.$$

4. Выполняется проверка непротиворечивости построенных функций исходным отношениям предшествования. Если противоречий нет, то функции предшествования построены правильно, в противном случае, функций предшествования не существует.

Продемонстрируем эти алгоритмы на примере. Построим матрицу и функции предшествования для грамматики:

$$\mathbf{S} \rightarrow (\mathbf{R}|a$$
$$\mathbf{R} \rightarrow \mathbf{S}a).$$

1. Построим матрицы отношений:

EQ

	S	R	*a*	(	)
S			1		
R					
a					1
(		1			
)					

FIRST

	S	R	*a*	(	)
S			1	1	
R	1				
a					
(					
)					

и LAST

	S	R	*a*	(	)
S		1	1		
R					1
a					
(					
)					

2. Находим транзитивные замыкания отношений FIRST и LAST.

$$F^2 = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$F^3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$L^2 = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$L^3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Так как кубы матриц нулевые, то дальнейшее возведение в степень лишено смысла. Окончательно имеем

$$F^+ = \begin{bmatrix} 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \; L^+ = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

3. Вычисляем отношение предшествования $<\bullet$.

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

4. Вычисляем отношение предшествования $\bullet>$.

$$(L^+)^T \times EQ = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix},$$

$$I + F^{+} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

В результате имеем

$$(L^{+})^{T} \times EQ \times (I + F^{+}) = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}.$$

5. Конструируем общую матрицу отношений

	S	R	*a*	*(*	*)*
S			≡		
R			●>		
a			●>		≡
(	<●	≡	<●	<●	
)			●>		

Видно, что грамматика является грамматикой предшествования. Далее сделаем попытку построить функции предшествования для данной матрицы отношений.

1. Построим блочную матрицу *B*. Составляющие её блоки суть

$$LE = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \cup \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

$$GE = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \cup \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}.$$

2. Найдём транзитивно-рефлексивное замыкание B^* блочной матрицы.

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, B^2 = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix},$$

$$B^3 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, B^4 = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}.$$

Начиная с итерации B^5, результаты вычислений будут чередоваться и равны, соответственно, B^3 и B^4. Поэтому, транзитивно-рефлексивное замыкание определиться объединением матриц $B \ldots B^4$.

$$B^* = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}.$$

3. По полученной матрице B^* рассчитаем функции $f(R_i)$ и $g(S_i)$.

$$B^* = \left.\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}\begin{matrix} 4 \\ 5 \\ 6 \\ 2 \\ 5 \end{matrix}\right\} F_i$$

$$\left.\begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \end{bmatrix}\begin{matrix} 3 \\ 2 \\ 4 \\ 3 \\ 6 \end{matrix}\right\} G_i$$

4. Проверим полученные табличные функции не непротиворечивость

F_i	S	R	*a*	*(*	*)*
G_i	3	2	4	3	6
S	4			≡		
R	5			●>		
a	6			●>		≡
(	2	<●	≡	<●	<●	
)	5			●>		

Упражнения для закрепления навыков

1. Определить является ли данная грамматика грамматикой предшествования.

$\mathbf{S} \to a b \mathbf{A} c$
$\mathbf{A} \to b\mathbf{A} | b$

Ответ: нет.

2. Построить отношения предшествования для грамматик

а) $\mathbf{P} \to \mathbf{A}$
$\mathbf{A} \to \mathbf{A} + \mathbf{T} | \mathbf{T}$
$\mathbf{E} \to (\mathbf{A}) | i$

б) $\mathbf{P} \to \mathbf{B}$
$\mathbf{B} \to \mathbf{A}$
$\mathbf{A} \to \mathbf{A} + \mathbf{M} | \mathbf{T}$
$\mathbf{M} \to \mathbf{T}$
$\mathbf{T} \to \mathbf{T} * \mathbf{E} | \mathbf{E}$
$\mathbf{E} \to (\mathbf{A}) | i$

3. Построить отношения предшествования и функции предшествования для грамматики

$\mathbf{S} \to \mathbf{A} | \mathbf{D}$
$\mathbf{A} \to ab | a\mathbf{B} | \mathbf{A}b$
$\mathbf{D} \to a | \mathbf{D}b$
$\mathbf{B} \to c$

Ответ: отношения предшествования определены, функций не существует.

4. Покажите, что следующие грамматики не являются грамматиками предшествования:

а) $\mathbf{Z} \to b\mathbf{E}b$
$\mathbf{E} \to \mathbf{E} + \mathbf{T} | \mathbf{T}$

б) $\mathbf{Z} \to b\mathbf{D}b$
$\mathbf{D} \to \mathbf{E} | \mathbf{E} + \mathbf{T} | \mathbf{T} | i$
$\mathbf{T} \to i | (\mathbf{D})$

5. Построить отношения предшествования для грамматики

$\mathbf{Z} \to b\mathbf{M}b$
$\mathbf{M} \to (\mathbf{L} | a$
$\mathbf{L} \to \mathbf{M}a)$

6. Построить функции предшествования по таблице отношений

●>		●>	●>
●>		●>	
<●	÷	<●	
≡		≡	

6. ЭКВИВАЛЕНТНЫЕ ПРЕОБРАЗОВАНИЯ ГРАММАТИК

Грамматики $G_1[S]$ и $G_2[Z]$ называются ***эквивалентными***, если они порождают один и тот же язык, то есть $L\{G_1[S]\} = L\{G_2[Z]\}$.

Очевидно, что необходимым условием эквивалентности является совпадение словарей терминальных символов. Совпадение множеств порождаемых цепочек – достаточное и необходимое условие эквивалентности грамматик.

Пример: Нижеследующие пары грамматик эквивалентны. Показать этот факт предоставляем читателю.

1. $\mathbf{S} \to \mathbf{A}|\mathbf{B}$
   $\mathbf{A} \to 1\mathbf{A}|1$
   $\mathbf{B} \to 0\mathbf{B}|0$
2. $\mathbf{F} \to 1\mathbf{E}|0\mathbf{Z}|0|1$
   $\mathbf{E} \to 1\mathbf{E}|1$
   $\mathbf{Z} \to 0\mathbf{Z}|0$
3. $\mathbf{S} \to b\mathbf{A}|a\mathbf{B}$
   $\mathbf{A} \to a|a\mathbf{S}|b\mathbf{AA}$
   $\mathbf{B} \to b|b\mathbf{S}|a\mathbf{BB}$
4. $\mathbf{S} \to ab|b\mathbf{A}|b\mathbf{AS}$
   $\mathbf{A} \to b\mathbf{AA}|a$
   $\mathbf{B} \to a\mathbf{BB}|b$

Одним из вынуждающих к преобразованию грамматики факторов, является наличие в ней ε-продукций, сильно усложняющих процедуру разбора. Грамматика, из правил которой устранены продукции вида $\mathbf{A} \to \varepsilon$, называется ε-свободной (e-free). Очевидно, что механическое устранение указанных продукций не выполняет задачу эквивалентного преобразования, так как требуется обеспечить тождество: $L\{G`[S]\} = L\{G[S]\} \setminus \{\varepsilon\}$.

Алгоритм построения ε-свободной грамматики

Пусть $G[S] = (V_T, V_N, S, R)$ - КС-грамматика, имеющая ε-продукции.

1. Объединим ***все*** ε-продукции (непосредственного вывода) из исходного множества правил R в множество R_ε.

2. Отыщем все нетерминальные символы $\boldsymbol{N_i} \in V_N$, такие что $N_i \Rightarrow^+ \varepsilon$ или $N_i \Rightarrow^* \varepsilon$. Указанные символы будем называть ***ε-порождающими***.

3. Каждой продукции $p \in R$, у которой в ***правой*** части находится один или несколько ε-порождающих символов, поставим в соответствие продукцию, в правой части которой, по сравнению с исходной, опущены один или более ε-порождающих символов. Вновь образованные продукции объединим в множество $\mathrm{R_o}$.

4. Конструируем новое множество продукций грамматики по правилу

$$R` = \{R \setminus R_\varepsilon\} \cup \{R_\mathrm{o}\}.$$

Рассмотрим применения алгоритма на пример.

Пусть грамматика $G[S]$ представима набором правил R:

$\mathbf{S} \to [\mathbf{E}]|\mathbf{E}$

$\mathbf{E} \to \mathbf{D}|\mathbf{D} + \mathbf{E}|\mathbf{D} - \mathbf{E}$

$\mathbf{D} \to \mathbf{F}|\mathbf{D}\ *\ \mathbf{F}|\mathbf{D}\ /\ \mathbf{F}$

$\mathbf{F} \to a|b|c|\varepsilon$

1. Шаг 1 даёт $R_\varepsilon = \{\mathbf{F} \to \varepsilon\}$.

2. На шаге 2 необходимо попытаться построить синтаксические выводы. Имеем следующие выводы $\mathbf{F} \Rightarrow \varepsilon$, $\mathbf{S} \Rightarrow^* \varepsilon$, $\mathbf{E} \Rightarrow^+ \varepsilon$, $\mathbf{D} \Rightarrow^+ \varepsilon$.

3 Шаг 3 обеспечил построение множества Ro.

$\mathbf{S} \to [\ ]$

$\mathbf{E} \to +\ \mathbf{E}|\ \mathbf{D}\ +\ |\ +\ |-\ \mathbf{E}|\mathbf{D}\ -\ |\ -$

$\mathbf{D} \to *\ \mathbf{F}|\ \mathbf{D}\ *\ |\ *\ |\ /\ \mathbf{F}|\mathbf{D}\ /\ |\ /$

4. Окончательно, на шаге 4 получаем набор продукций для *G`[S]*.

$\mathbf{S} \to [\mathbf{E}]|\mathbf{E}|[\ ]$

$\mathbf{E} \to \mathbf{D}|\mathbf{D}\ +\ \mathbf{E}|\mathbf{D}\ -\ \mathbf{E}|+\ \mathbf{E}|\ \mathbf{D}\ +\ |\ +\ |-\ \mathbf{E}|\mathbf{D}\ -\ |\ -$

$\mathbf{D} \to \mathbf{F}|\mathbf{D}\ *\ \mathbf{F}|\mathbf{D}\ /\ \mathbf{F}|*\ \mathbf{F}|\ \mathbf{D}\ *\ |\ *\ |\ /\ \mathbf{F}|\mathbf{D}\ /\ |\ /$

$\mathbf{F} \to a|b|c$

Можно убедиться, что языки, порождаемые грамматиками тождественны. Этот факт эквивалентности грамматик студент может проверить самостоятельно.

В ходе разработки формальной грамматики возникают закономерные вопросы о том, порождает ли грамматика цепочки терминальных символов (предложения языка), либо смешанные цепочки, и все ли продукции грамматики используются. Одним из компонентов исследования непустоты языка является преобразования грамматики в нормальную форму Хомского (Chomsky normal form), продукции которой имеют вид:

$\mathbf{A} \to \mathbf{BC}$ или $\mathbf{A} \to \alpha$, где $\mathbf{A, B, C} \in V_N$, $\alpha \in V_T$.

Алгоритм построения грамматики в нормальной форме Хомского

1. Среди продукций грамматики отыщем все ***первичные продукции*** вида

$$\mathbf{A}_i \to \mathbf{B}_j,\ \mathbf{A}_i,\ \mathbf{B}_j \in V_N,$$

правая часть которых содержит единственный нетерминальный символ.

То есть, один нетерминал (слева) определяется через другой (справа). При этом всё множество продукций с нетерминалом $\mathbf{A}_i$ в левой части разбивается на подмножества: $U\{\mathbf{A}_i\}$ – первичных продукций и $N\{\mathbf{A}_i\}$ – не являющихся таковыми.

Для $\forall \mathbf{A}_i$, где $U\{\mathbf{A}_i\} \neq \{\}$, ставится в соответствие множество правил

$$\{\mathbf{A}_i \Rightarrow^+ \alpha |\ \mathbf{A}_i \to \mathbf{B}_j,\ \mathbf{B}_j \Rightarrow^+ \alpha,\ \alpha \in V^+,\ \mathbf{B}_j \in N(\mathbf{B}_j)\}.$$

Последняя запись обозначает, что правая часть первичной продукции с нетерминалом $\mathbf{A}_i$ в левой части заменяется правыми частями непервичных продукций нетерминала $\mathbf{B}_j$, обеспечивающими вывод α из $\mathbf{A}_i$.

После выполнения этого шага алгоритма в формальной грамматике не останется первичных продукций.

2. Отыщем все ***вторичные*** продукции. Правые части таких продукций представляют собой ***смешанные цепочки*** терминальных и нетерминальных символов.

Каждый терминальный символ такой цепочки η заменяется нетерминалом $\mathbf{A}_{\eta}$. Одновременно множество продукций пополняется правилом $\mathbf{A}_{\eta} \to \eta$, определяющим данный нетерминал.

Для примера, пусть правая часть вторичной продукции $x_1x_2...x_m,\ x_i \in (V_T \cup V_N), i = \overline{1,m}$. Тогда после преобразования она примет вид $y_1y_2...y_m,\ y_i \in V_N,\ i = \overline{1,m}$ при соответствии между цепочками

$$y_i = \begin{cases} x_i, & x_i \in V_N, \\ X_{x_i}, & X_{x_i} \in V_N, x_i \in V_T, \\ R^* = R \cup \left\{ X_{x_i} ::= x_i \right\} \end{cases}$$

В результате данного шага преобразования из грамматики будут устранены все вторичные продукции, и грамматика претерпит изменение.

3. В модифицированном, в ходе п.п. 2 и 3, множестве продукций, отыщем ***третичные*** продукции, в правых частях которых содержится ***не менее трёх*** нетерминалов.

Для каждой найденной третичной продукции строится разложение, которым указанная продукция заменяется. Разложение строится следующим образом. Пусть исходная третичная продукция имеет вид:

$$A_r \to S_1S_2...S_m,\ S_i \in V_N,\ i = \overline{1,m},\ m \geq 2.$$

Её разложение будет выглядеть так:

$$\begin{aligned} A_r &\to S_1N_{B1}, \\ N_{B1} &\to S_2N_{B2}, \\ &\dots \\ N_{Bm-1} &\to S_{m-1}S_m. \end{aligned}$$

Нетерминалы $\boldsymbol{N_{Bi}}$, $i=1,m-1$ суть новые, нетерминалы, ***не содержащиеся более ни в одной продукции***, добавленные к словарю $\boldsymbol{V_N}$ в ходе разложения.

Пример. Пусть грамматика определяется совокупностью правил:

$$\begin{aligned} &\mathbf{S} \to \mathbf{ABA}|\mathbf{A} \\ &\mathbf{A} \to a|a\mathbf{A}|\mathbf{B} \\ &\mathbf{B} \to b|b\mathbf{B} \end{aligned}$$

1. Первичные продукции суть

$$\begin{aligned} &\mathbf{S} \to \mathbf{A} \\ &\mathbf{A} \to \mathbf{B} \end{aligned}$$

Имеем следующие множества первичных и непервичных продукций.

$$\begin{aligned} &\mathrm{U}(\mathbf{A})=\{\mathbf{A} \to \mathbf{B}\}, && \mathrm{N}(\mathbf{A})=\{\mathbf{A} \to a\mathbf{A}|a\}, \\ &\mathrm{U}(\mathbf{B})=\{\}, && \mathrm{N}(\mathbf{B})=\{\mathbf{B} \to b\mathbf{B}|b\}, \\ &\mathrm{U}(\mathbf{S})=\{\mathbf{S} \to \mathbf{A}\}. && \mathrm{N}(\mathbf{S})=\{\mathbf{S} \to \mathbf{ABA}\}. \end{aligned}$$

Правилу $\mathbf{A} \to \mathbf{B}$ поставим в соответствие заменяющие правила $\mathbf{A} \to b\mathbf{B}|b$, а продукции $\mathbf{S} \to \mathbf{A}$ – правила $\mathbf{S} \to a\mathbf{A}|b\mathbf{B}|a|b$, а правилу. Получим новый набор правил изменённой грамматики.

$$\mathbf{S} \to \mathbf{ABA}|a\mathbf{A}|b\mathbf{B}|a|b$$
$$\mathbf{A} \to a\mathbf{A}|b\mathbf{B}|a|b$$
$$\mathbf{B} \to b|b\mathbf{B}$$

2. К вторичным продукциям относятся все продукции с правыми частями $a\mathbf{A}$ и $b\mathbf{B}$. Введением правил: $\mathbf{C} \to a$, $\mathbf{D} \to b$ эти продукции преобразуются к виду **CA** и **DB** соответственно. В результате имеем грамматику, представленную следующим набором правил:

$$\mathbf{S} \to \mathbf{ABA}|\mathbf{CA}|\mathbf{DB}|a|b,$$

$$\mathbf{A} \to \mathbf{CA}|\mathbf{DB}|a|b$$

$$\mathbf{B} \to \mathbf{DB}|b$$

$$\mathbf{C} \to a$$

$$\mathbf{D} \to b$$

3. В последнем множестве продукций имеется единственная в своём роде третичная продукция: $\mathbf{S} \to \mathbf{ABA}$. Разложим и заменим её парой продукций $\mathbf{S} \to \mathbf{AE}$, $\mathbf{E} \to \mathbf{BA}$. Таким образом, нами получена грамматика в нормальной форме Хомского:

$$\mathbf{S} \to \mathbf{AE}|\mathbf{CA}|\mathbf{DB}|a|b,$$
$$\mathbf{A} \to \mathbf{CA}|\mathbf{DB}|a|b,$$
$$\mathbf{B} \to \mathbf{DB}|b,$$
$$\mathbf{E} \to \mathbf{BA},$$
$$\mathbf{C} \to a,$$
$$\mathbf{D} \to b.$$

Как не сложно заметить, сия грамматика является эквивалентной исходной.

Для построения грамматики, свободной от левой рекурсии в правилах, и правые части продукций начинаются с терминала, её приводят в нормальную форму Грейбах (Greibach S.A. normal form), продукции которой суть:

$$\mathbf{A} \to a\beta,\ a \in V_T,\ \beta \in (V_N \cup V_T)^*$$

Алгоритм построения нормальной формы Грейбах

Алгоритм основывается на применении теорем.

Теорема 1. Если $\mathbf{A} \to \alpha\mathbf{B}\gamma$, $\mathbf{A}$, $\mathbf{B} \in V_N$, $\alpha, \gamma \in \mathrm{V}^*$ правило КС – грамматики, а

$$\mathbf{B} \to \beta_1|\beta_2|\ldots|\beta_\mathrm{m}$$

все продукции данной грамматики с нетерминалом **B** в своих левых частях,

то указанные продукции можно заменить правилами вида

$$\mathbf{A}\to\alpha\beta_1\gamma|\alpha\beta_2\gamma|\ldots|\alpha\beta_m\gamma,$$

причём произведённая замена никак не отражается на языке, порождаемом этой грамматикой.

Теорема 2. Если продукции некоторой контекстно-свободной грамматики являются рекурсивными слева продукциями

$$\mathbf{A}\to\mathbf{A}\alpha_1|\mathbf{A}\alpha_2|\ldots\mathbf{A}\alpha_m,\ \mathbf{A}\in V_N,\ \alpha_i\in(V\setminus\mathbf{A})^*,$$

а остальные продукции с нетерминалом **A** в левой части суть

$$\mathbf{A}\to\beta_1|\beta_2|\ldots|\beta_k,$$

то новая грамматика, эквивалентная исходной, может быть построена путём добавления нетеминала **A`** к множеству V_N и заменой этих продукций исходной грамматики продукциями вида

$$\mathbf{A}\to\beta_1|\beta_2|\ldots|\beta_k|\beta_1\ \mathbf{A`}|\beta_2\ \mathbf{A`}\ldots|\beta_k\ \mathbf{A`}\text{ и}$$
$$\mathbf{A`}\to\alpha_1|\alpha_2|\ldots|\alpha_m|\beta_1\ \mathbf{A`}|\beta_2\ \mathbf{A`}\ldots|\beta_k\ \mathbf{A`}.$$

Пример. Пусть грамматика задана множеством продукций

$$\mathbf{S}\to\mathbf{AB}|\mathbf{A}$$
$$\mathbf{A}\to a\mathbf{A}|b|\mathbf{B}$$
$$\mathbf{B}\to b|\mathbf{B}b$$

Имеем леворекурсивную (прямую левую) продукцию $\mathbf{B}\to\mathbf{B}b$. Используя теорему 2, избавимся от явной леворекурсивной продукции. Получим пару продукций, эквивалентных заменённым:

$$\mathbf{B}\to b\mathbf{C}|b$$
$$\mathbf{C}\to b\mathbf{C}|b$$

Продукции грамматики примут вид:

$$\mathbf{S}\to\mathbf{AB}|\mathbf{A}$$
$$\mathbf{A}\to a\mathbf{A}|b|\mathbf{B}$$
$$\mathbf{B}\to b|\mathbf{C}b$$
$$\mathbf{C}\to b\mathbf{C}|b$$

Последовательное применение теоремы 1 для правил

$$\mathbf{S}\to\mathbf{AB}|\mathbf{A}\text{ и}$$
$$\mathbf{A}\to\mathbf{B}$$

даёт следующее:

$$\mathbf{S}\to\mathbf{AB}\quad\Rightarrow\quad\mathbf{S}\to a\mathbf{AB}|b\mathbf{B}|\mathbf{BB}$$
$$\mathbf{S}\to\mathbf{A}\quad\Rightarrow\quad\mathbf{S}\to a\mathbf{A}|b|\mathbf{B}$$
$$\mathbf{A}\to\mathbf{B}\quad\Rightarrow\quad\mathbf{A}\to b|b\mathbf{C}$$

При этом возникают новые правила, правые части которых начинаются с нетерминалов, преобразуем их аналогично, в соответствии с теоремой 1.

$$\mathbf{S}\to\mathbf{BB}\quad\Rightarrow\quad\mathbf{S}\to b\mathbf{B}|b\mathbf{CB}$$
$$\mathbf{S}\to\mathbf{B}\quad\Rightarrow\quad\mathbf{S}\to b|b\mathbf{C}$$

Окончательно, объединив правила, имеем грамматику в форме Грейбах:

S→*a***AB**|*b***B**|*a***A**|*b***CB**|*b***C**|*b*
A→*a***A**|*b***C**|*b*
B→*b*|*b***C**
C→*b***C**|*b*

Упражнения для закрепления навыков

1. Построить ε-свободную грамматику, эквивалентную заданной

1.	$\mathbf{S} \to \mathbf{A}$	2.	$\mathbf{S} \to \mathbf{AB}$	3.	$\mathbf{S} \to \varepsilon \mid a\mathbf{B} \mid a\mathbf{C}$
	$\mathbf{A} \to a\mathbf{A}a$		$\mathbf{A} \to \mathbf{G} \mid ab$		$\mathbf{C} \to a\mathbf{C} \mid a\mathbf{B} \mid \varepsilon$
	$\mathbf{A} \to a$		$\mathbf{G} \to c \mid \varepsilon$		$\mathbf{B} \to b\mathbf{B} \mid b \mid \varepsilon$
	$\mathbf{A} \to \varepsilon$				

2. Привести полученные ε-свободные грамматики в нормальные формы Хомского и Грейбах.

3. Привести грамматику к нормальной форме Грейбах.

1. $\mathbf{A} \to \mathbf{BC}$
   $\mathbf{B} \to \mathbf{CA} \mid b$
   $\mathbf{C} \to \mathrm{AB} \mid a$

2. $\mathbf{S} \to \mathbf{AB} \mid \mathbf{AS}$
   $\mathbf{A} \to \mathbf{BS} \mid a$
   $\mathbf{B} \to \mathbf{AA} \mid b$

4. Упражнение для «продвинутых» студентов. Постройте грамматику в форме Грейбах, порождающую арифметические выражения над словарем терминалов {идентификатор (i), константа явная (d), константа описанная в разделе констант dd, как в Pascal'е или C'и}.

7. ВАРИАНТЫ КОНТРОЛЬНЫХ ЗАДАНИЙ ПО ТЕОРИИ КОНЕЧНЫХ АВТОМАТОВ

Задание.

1. Построить минимальный детерминированный конечный автомат, принимающий цепочки, соответствующие выражению, заданному по варианту.

2. Построить леволинейную и праволинейную регулярные грамматики, соответствующие построенному МДКА.

3. Исследовать полученные грамматики на принадлежность к классу *LL*(1).

1. $\{a \vee b \vee c\}\ aa\ \{b \vee c\}$
2. $\{ab \vee bc\} \vee \{a \vee c\}\ b$
3. $\{b\ \{a \vee c\}\} \vee \{a \vee b\}$

4. $b \vee c\{b \vee bc\}\ ba$
5. $ab \vee a\{c \vee b\}\ b$
6. $\{b\} \vee \{a\} \vee c\{ab \vee c\}$
7. $\{\{a\} \vee bc\}\ ab\{a\}$
8. $a\{cb\} \vee b\{a \vee b\}$
9. $c \vee b \vee ab\{b \vee \{c\}\}$
10. $f\{g\} \vee k\{fg \vee k\}$
11. $\{a \vee b\}\{ab\}a \vee b$
12. $c \vee a \vee b\{ab\}ca\{b\}$
13. $\{a\}bc \vee \{c\}a \vee b$
14. $\{b \vee a\} \vee \{ba\}c \vee c$
15. $\{bc \vee c \vee a\}aac\{b\}$
16. $\{ab \vee b\{c \vee ba\}\}\{c\}$
17. $b\{a\{c\}\} \vee ab\{c \vee b\}$
18. $a\{f \vee ka\} \vee \{k\}\{f\}$
19. $ff\{a\} \vee kk\{a \vee f\}$
20. $gh\{a\} \vee \{a \vee gha\} \vee h$
21. $ab\{c \vee bc\} \vee \{b\}ca$
22. $cb \vee ca\{a\{bc\}\}$
23. $ak\{f \vee kf\} \vee a\{k\}$
24. $ca \vee b\{a\} \vee a\{b \vee c\}$
25. $bb \vee a\{bc\}c\{a \vee ab\}$
26. $\{fa \vee f \vee a\}ff\{a\}$
27. $ca \vee c\{c\} \vee ca\{c\{a\}\}$
28. $\{gh\{h\} \vee g\}hh\{g\}$
29. $\{cb \vee b\{c\}\}bc \vee b\{cc\}$
30. $ah\{h \vee ah\}ah\{a\}$

ЗАКЛЮЧЕНИЕ

Освещённые в настоящем методическом указании вопросы ни в коей мере не претендуют на всемерный охват области информационных технологий, посвящённых синтаксическому анализу. Желающие, по мере необходимости, могут пополнять и углублять свои знания, пользуясь специальной литературой. Выражаем надежду и уверенность, что данная методическая разработка не только послужит хорошим начальным подспорьем для дальнейшего освоения дисциплины, но и поможет сделать первые шаги, которые, как известно, порой нелегки…

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Ахо А. Теория синтаксического анализа, перевода и компиляции. Синтаксический анализ / А. Ахо, Дж. Ульман. – М.: Мир, 1978. – 619 с.
2. Ахо А. Теория синтаксического анализа, перевода и компиляции. Компиляция / А. Ахо, Дж. Ульман. – М.: Мир, 1978. – 560 с.
3. Ахо А. Компиляторы: принципы, технологии, инструменты / А. Ахо, Дж. Ульман. – М.: Мир, 1978. – 619 с.
4. Ахо А. Компиляторы: принципы, технологии и инструменты/ А. Ахо, Р. Сети, Дж. Ульман. – М.: Издательский дом «Вильямс», 2002. – 512 с.
5. Басс Л. Архитектура программного обеспечения на практике / Л. Басс, П. Клеменм, Р. Кауфман. – СпБ.: Питер, 2006. – 576 с.
6. Бекхауз Р.Ч. Синтаксис языков программирования / Р.Ч. Бекхауз. – М.: Мир, 1986. – 281 с.
7. Вайнгартен Ф. Трансляция языков программирования / Ф. Вайнгартен. – М.: Мир, 1977. – 190 с.
8. Гинзбург С. Математическая теория контекстно-свободных языков / С. Гинзбург. – М.: Мир, 1970. – 336 с.
9. Гордеев А.В. Системное программное обеспечение. / А.В. Гордеев, А.Ю. Молчанов. – СпБ.: Питер, 2003. – 736 с.
10. Грис Д. Конструирование компиляторов для ЦВМ / Д. Грис. – М.: Мир, 1975. - 544 с.
11. Гросс М. Теория формальных грамматик / М. Гросс. – М.: Мир, 1971.
12. Землянский А.А. Разработка трансляторов. Учебное пособие / А.А. Землянский. – М.: Издательство МЭСИ, 1974. – 191 с.
13. Зиглер К. Методы проектирования программных систем / Зиглер К. – М.: Мир, 1985. – 328 с.
14. Ингерман Л. Системы построения трансляторов / Л. Ингерман. – М.: Мир, 1969. – 336 с.
15. Карпов Ю.Г. Теория автоматов / Ю.Г. Карпов. – СПб.: Питер, 2002. – 224 с.
16. Касаткин А.И. Профессиональное программирование на языке Си. Системное программирование / А.И. Касаткин А.И. – Минск: Вышейшая школа, 1993. – 301 с.
17. Компаниец Р.И. Основы построения трансляторов / Р.И. Компаниец, Маньков Е.В., Н.Е. Филатов. – Спб.: Корона-принт, 2000, – 256 с.
18. Льюис Ф. Теоретические основы проектирования компиляторов / Ф. Льис, Д. Розенкранц, Р. Стирнз. – М.: Мир, 1979. – 564 с.
19. Молчанов А.Ю. Системное программное обеспечение: учебник для вузов / А.Ю. Молчанов. – СпБ.: Питер, 2006. – 396 с.

20. Молчанов А.Ю. Системное программное обеспечение: лабораторный практикум / А.Ю. Молчанов. – СпБ.: Питер, 2005. – 284 с.
21. Оллонгрен А. Определение языков программирования интерпретирующими автоматами / А. Оллонгрен. – М.: Мир, 1972. - 288 с.
22. Рейуорд-Смит В.Дж. Теория формальных языков. Вводный курс / В.Дж. Рейуорд-Смит. – М.: Мир, 1986. – 128 с.
23. Саломаа А. Жемчужины теории формальных языков / А. Саломаа. – М.: Мир, 1986. – 159 с.
24. Серебряков В.И. Лекции по конструированию компиляторов / В.И. Серебряков. – М.: МГУ, 1997. – 171 с.
25. Фельдман Дж. Системы построения трансляторов / Дж. Фельдман., Д. Грис. – М.: Мир, 1971. – 436 с.
26. Фокс Дж. Программное обеспечение и его разработка / Дж. Фокс. – М.: Мир, 1985. – 368 с.
27. Хантер Р. Проектирование и конструирование компиляторов / Р. Хантер. – М.: Финансы и статистика, 1984. – 232 с.
28. Хантер Р. Основные концепции компиляторов / Р. Хантер. – М.: Издательский дом “Вильямс”, 2002. - 256 с.
29. Хопгуд Ф. Методы компиляции / Ф. Хопгуд. – М.: Мир, 1972. – 158 с.
30. Хопкрофт Дж. Е. Формальные языки и автоматы / Дж. Е. Хопкрофт, Дж. Д. Ульман. – М.: Мир, 1982. – 346 с.
31. Хопкрофт Дж. Введение в теорию автоматов, языков и вычислений / Дж. Хопркрофт, Р. Мотвани, Дж. Ульман. – М.: Издательский дом “Вильямс”, 2002. – 528 с.

Заказ №		от “		“		2015		. Тираж	*50*	экз.
				Изд-во СевГУ						