

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ОБРАЗОВАНИЯ

Государственное образовательное учреждение
высшего профессионального образования
“Оренбургский государственный университет”

Кафедра программного обеспечения вычислительной техники
и автоматизированных систем

Е.Н. ИШАКОВА

ТЕОРИЯ ФОРМАЛЬНЫХ ЯЗЫКОВ, ГРАММАТИК И АВТОМАТОВ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
К ЛАБОРАТОРНОМУ ПРАКТИКУМУ

Рекомендовано к изданию Редакционно-издательским советом
государственного образовательного учреждения
высшего профессионального образования
“Оренбургский государственный университет”

Оренбург 2005

УДК 004.4'4(075.8)
ББК 32.973.26-018.1я73
И 97

Рецензент

кандидат технических наук, доцент Бахарева Н.Ф.

И 97 **Ишакова Е.Н.**
Теория формальных языков, грамматик и автоматов: Методи-
ческие указания к лабораторному практикуму. - Оренбург:
ГОУ ОГУ, 2005. – 54 с.

В методических указаниях содержатся материалы, необходимые для самостоятельной подготовки студентов к выполнению лабораторных работ по разделу «Теория формальных языков, грамматик и автоматов». В описание лабораторных работ включены цель работы, порядок ее выполнения, рассмотрены теоретические вопросы, связанные с реализацией поставленных задач, приведена необходимая литература. В приложениях представлены правила оформления результатов лабораторной работы.

Методические указания предназначены для выполнения лабораторного практикума по дисциплине «Теория языков программирования и методов трансляции» для студентов специальности 220400 – «Программное обеспечение вычислительной техники и автоматизированных систем».

И 1404000000
6Л9-04

ББК 32.973.26-018.1я73

© Ишакова Е.Н., 2005
© ГОУ ОГУ, 2005

Введение

Предлагаемый материал посвящен основам классической теории формальных языков, грамматик и автоматов – одной из важнейших составных частей инженерного образования по информатике и вычислительной технике.

Теория формальных языков, грамматик и автоматов составляет фундамент синтаксических методов. Основы этой теории были заложены Н. Хомским в 40–50-е годы XX столетия в связи с его лингвистическими работами, посвященными изучению естественных языков. Но уже в следующем десятилетии синтаксические методы нашли широкое практическое применение в области разработки и реализации языков программирования.

В настоящее время искусственные языки, использующие для описания предметной области текстовое представление, широко применяются не только в программировании, но и в других областях. С их помощью описывается структура всевозможных документов, трехмерных виртуальных миров, графических интерфейсов пользователя и многих других объектов, используемых в моделях и в реальном мире. Для того чтобы эти текстовые описания были корректно составлены, а затем правильно распознаны и интерпретированы, применяются специальные методы их анализа и преобразования. В основе данных методов лежит теория формальных языков, грамматик и автоматов.

Теория формальных языков, грамматик и автоматов дала новый стимул развитию математической лингвистики и методам искусственного интеллекта, связанных с естественными и искусственными языками. Кроме того, ее элементы успешно применяются, например, при описании структур данных, файлов, изображений, представленных не в текстовом, а двоичном формате. Эти методы полезны при разработке своих трансляторов даже там, где уже имеются соответствующие аналоги.

В методических указаниях содержатся материалы, необходимые для самостоятельной подготовки студентов к выполнению лабораторных работ по разделу «Теория формальных языков, грамматик и автоматов». В описание лабораторных работ включены цель работы, порядок ее выполнения, рассмотрены теоретические вопросы, связанные с реализацией поставленных задач, приведена необходимая литература. В приложениях представлены правила оформления результатов лабораторной работы.

1 Лабораторная работа № 1. Распознавание типов формальных языков и грамматик

Цель: - закрепить понятия «алфавит», «цепочка», «формальная грамматика» и «формальный язык», «выводимость цепочек», «эквивалентная грамматика»;

- сформировать умения и навыки распознавания типов формальных языков и грамматик по классификации Хомского, построения эквивалентных грамматик.

Основы теории

Определение 1.1. Алфавитом V называется конечное множество символов.

Определение 1.2. Цепочкой α в алфавите V называется любая конечная последовательность символов этого алфавита.

Определение 1.3. Цепочка, которая не содержит ни одного символа, называется пустой цепочкой и обозначается ε .

Определение 1.4. Формальное определение цепочки символов в алфавите V :

- 1) ε - цепочка в алфавите V ;
- 2) если α - цепочка в алфавите V и a - символ этого алфавита, то αa - цепочка в алфавите V ;
- 3) β - цепочка в алфавите V тогда и только тогда, когда она является таковой в силу утверждений 1) и 2).

Определение 1.5. Длиной цепочки α называется число составляющих ее символов (обозначается $|\alpha|$).

Обозначим через V^* множество, содержащее все цепочки в алфавите V , включая пустую цепочку ε , а через V^+ - множество, содержащее все цепочки в алфавите V , исключая пустую цепочку ε .

Пример 1.1. Пусть $V = \{1, 0\}$, тогда $V^* = \{\varepsilon, 0, 1, 00, 01, 10, 11, 000, \dots\}$, а $V^+ = \{0, 1, 00, 01, 10, 11, 000, \dots\}$.

Определение 1.6. Формальной грамматикой называется четверка вида:

$$G = (V_T, V_N, P, S), \quad (1.1)$$

где V_N - конечное множество нетерминальных символов грамматики (обычно прописные латинские буквы);

V_T - множество терминальных символов грамматики (обычно строчные латинские буквы, цифры, и т.п.), $V_T \cap V_N = \emptyset$;

P - множество правил вывода грамматики, являющееся конечным подмножеством множества $(V_T \cup V_N)^+ \times (V_T \cup V_N)^*$; элемент (α, β) множества P называется правилом вывода и записывается

ся в виде $\alpha \rightarrow \beta$ (читается: «из цепочки α выводится цепочка β »);

S - начальный символ грамматики, $S \in V_N$.

Для записи правил вывода с одинаковыми левыми частями вида $\alpha \rightarrow \beta_1, \alpha \rightarrow \beta_2, \dots, \alpha \rightarrow \beta_n$ используется сокращенная форма записи $\alpha \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$.

Пример 1.2. Грамматика $G_1 = (\{0, 1\}, \{A, S\}, P_1, S)$, где множество P_1 состоит из правил вида: 1) $S \rightarrow 0A1$; 2) $0A \rightarrow 00A1$; 3) $A \rightarrow \varepsilon$.

Определение 1.7. Цепочка $\beta \in (V_T \cup V_N)^*$ непосредственно выводима из цепочки $\alpha \in (V_T \cup V_N)^+$ в грамматике $G = (V_T, V_N, P, S)$ (обозначается: $\alpha \Rightarrow \beta$), если $\alpha = \xi_1 \gamma \xi_2$ и $\beta = \xi_1 \delta \xi_2$, где $\xi_1, \xi_2, \delta \in (V_T \cup V_N)^*$, $\gamma \in (V_T \cup V_N)^+$ и правило вывода $\gamma \rightarrow \delta$ содержится во множестве P .

Определение 1.8. Цепочка $\beta \in (V_T \cup V_N)^*$ выводима из цепочки $\alpha \in (V_T \cup V_N)^+$ в грамматике $G = (V_T, V_N, P, S)$ (обозначается $\alpha \Rightarrow^* \beta$), если существует последовательность цепочек $\gamma_0, \gamma_1, \dots, \gamma_n$ ($n \geq 0$) такая, что $\alpha = \gamma_0 \Rightarrow \gamma_1 \Rightarrow \dots \Rightarrow \gamma_n = \beta$.

Пример 1.3. В грамматике G_1 $S \Rightarrow^* 000111$, т.к. существует вывод $S \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000111$.

Определение 1.9. Языком, порожденным грамматикой $G = (V_T, V_N, P, S)$, называется множество всех цепочек в алфавите V_T , которые выводимы из начального символа грамматики S с помощью правил множества P , т.е. множество $L(G) = \{\alpha \in V_T^* \mid S \Rightarrow^* \alpha\}$.

Пример 1.4. Для грамматики G_1 $L(G_1) = \{0^n 1^n \mid n > 0\}$.

Определение 1.10. Цепочка $\alpha \in (V_T \cup V_N)^*$, для которой существует вывод $S \Rightarrow^* \alpha$, называется сентенциальной формой в грамматике $G = (V_T, V_N, P, S)$.

Определение 1.11. Грамматики G_1 и G_2 называются эквивалентными, если $L(G_1) = L(G_2)$.

Пример 1.5. Для грамматики G_1 эквивалентной будет грамматика $G_2 = (\{0, 1\}, \{S\}, P_2, S)$, где множество правил вывода P_2 содержит правила вида $S \rightarrow 0S1 \mid 01$.

Классификация грамматик по Хомскому

Тип 0. Грамматика $G = (V_T, V_N, P, S)$ называется грамматикой типа 0, если на ее правила вывода не наложено никаких ограничений, кроме тех, которые указаны в определении грамматики.

Тип 1. Грамматика $G = (V_T, V_N, P, S)$ называется контекстно-зависимой грамматикой (КЗ-грамматикой), если каждое правило вывода из множества P имеет вид $\alpha \rightarrow \beta$, где $\alpha \in (V_T \cup V_N)^+$, $\beta \in (V_T \cup V_N)^*$ и $|\alpha| \leq |\beta|$.

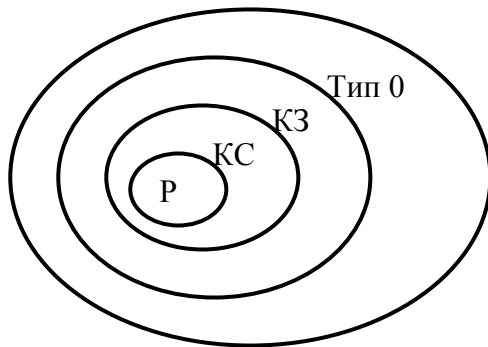
Тип 2. Грамматика $G = (V_T, V_N, P, S)$ называется контекстно-свободной грамматикой (КС-грамматикой), если ее правила вывода имеют вид: $A \rightarrow \beta$, где $A \in V_N$ и $\beta \in V^*$.

Тип 3. Грамматика $G = (V_T, V_N, P, S)$ называется регулярной грамматикой (Р-грамматикой) выровненной вправо, если ее правила вывода имеют вид $A \rightarrow aB \mid a$, где $a \in V_T$; $A, B \in V_N$.

Грамматика $G = (V_T, V_N, P, S)$ называется регулярной грамматикой (Р-грамматикой) выровненной влево, если ее правила вывода имеют вид $A \rightarrow Ba \mid a$, где $a \in V_T$; $A, B \in V_N$.

Определение 1.12. Язык $L(G)$ называется языком типа k , если его можно описать грамматикой типа k , где k – максимально возможный номер типа грамматики.

Соотношение типов грамматик и языков представлено на рисунке 1.1.



Р – регулярная грамматика;

КС – контекстно-свободная грамматика;

КЗ – контекстно-зависимая грамматика;

Тип 0 – грамматика типа 0.

Рисунок 1.1 – Соотношение типов формальных языков и грамматик

Пример 1.6. Примеры различных типов формальных языков и грамматик по классификации Хомского. Терминалы будем обозначать строчными символами, нетерминалы – прописными буквами, начальный символ грамматики – S .

а) Язык типа 0 $L(G) = \{a^2b^{n^2-1} \mid n \geq 1\}$ определяется грамматикой с правилами вывода:

- | | |
|----------------------------------|------------------------------------|
| 1) $S \rightarrow aaCFD$; | 2) $AD \rightarrow D$; |
| 3) $F \rightarrow AFB \mid AB$; | 4) $Cb \rightarrow bC$; |
| 5) $AB \rightarrow bBA$; | 6) $CB \rightarrow C$; |
| 7) $Ab \rightarrow bA$; | 8) $bCD \rightarrow \varepsilon$. |

б) Контекстно-зависимый язык $L(G) = \{a^n b^n c^n \mid n \geq 1\}$ определяется грамматикой с правилами вывода:

- | | |
|------------------------------------|--------------------------|
| 1) $S \rightarrow aSBC \mid abc$; | 2) $bC \rightarrow bc$; |
|------------------------------------|--------------------------|

$$3) CB \rightarrow BC;$$

$$4) cC \rightarrow cc;$$

$$5) BB \rightarrow bb.$$

в) Контекстно-свободный язык $L(G) = \{(ab)^n(cb)^n \mid n > 0\}$ определяется грамматикой с правилами вывода:

$$1) S \rightarrow aQb \mid accb;$$

$$2) Q \rightarrow cSc.$$

г) Регулярный язык $L(G) = \{\omega \perp \mid \omega \in \{a, b\}^+, \text{ где нет двух рядом стоящих } a\}$ определяется грамматикой с правилами вывода:

$$1) S \rightarrow A\perp \mid B\perp;$$

$$2) A \rightarrow a \mid Ba;$$

$$3) B \rightarrow b \mid Bb \mid Ab.$$

Постановка задачи к лабораторной работе № 1

При выполнении лабораторной работы следует реализовать следующие действия:

1) составить грамматику, порождающую формальный язык, заданный в соответствии с вариантом;

2) определить тип формальной грамматики и языка по классификации Хомского;

3) разработать программное средство, распознающее тип введенной пользователем грамматики по классификации Хомского.

Варианты индивидуальных заданий представлены в таблице 1.1.

Таблица 1.1 – Варианты индивидуальных заданий к лабораторной работе № 1

Вариант	Формальный язык
1	$L(G) = \{a^n b^m c^k \mid n, m, k > 0\}$
2	$L(G) = \{(ab)^n (cb)^m \mid n, m \geq 0\}$
3	$L(G) = \{0^n (10)^m \mid n, m \geq 0\}$
4	$L(G) = \{wcw cw \mid w \in \{a, b\}^+\}$
5	$L(G) = \{c^{2n} d^n \mid n > 0\}$
6	$L(G) = \{l+l-l \mid l \in \{a, b\}^+\}$
7	$L(G) = \{(10)^{n-1} (01)^{n+1} \mid n > 0\}$
8	$L(G) = \{(ac)^n \mid n > 0, a \in \{b, d\}, c \in \{+, -\}\}$
9	$L(G) = \{\perp (010)^n \perp \mid n > 0\}$
10	$L(G) = \{a_1 a_2 \dots a_n a_n \dots a_2 a_1 \mid a_i \in \{0, 1\}\}$
11	$L(G) = \{a_1 a_2 \dots a_n a_1 a_2 \dots a_n \mid a_i \in \{c, d\}\}$
12	$L(G) = \{ab.b \mid a_i \in \{+, -\}, b \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}^+\}$

2 Лабораторная работа № 2. Тема: «Построение конечного автомата по регулярной грамматике»

Цель: - закрепить понятия «регулярная грамматика», «недетерминированный и детерминированный конечный автомат»;

- сформировать умения и навыки построения конечного автомата по регулярной грамматике и преобразования недетерминированного конечного автомата к детерминированному конечному автомату.

Основы теории

Распознавателем для регулярной грамматики является конечный автомат (КА).

Определение 2.1. Детерминированным конечным автоматом (ДКА) называется пятерка объектов:

$$M = (Q, T, F, H, Z), \quad (2.1)$$

где Q - конечное множество состояний автомата;

T - конечное множество допустимых входных символов;

F - функция переходов, отображающая множество $Q \times T$ во множество Q ;

H - конечное множество начальных состояний автомата;

Z - множество заключительных состояний автомата, $Z \subseteq Q$.

Определение 2.2. Недетерминированным конечным автоматом (НКА) называется конечный автомат, в котором в качестве функции переходов используется отображение $Q \times T$ во множество всех подмножеств множества состояний автомата $P(Q)$, т.е. функция переходов неоднозначна, так как текущей паре (q, t) соответствует множество очередных состояний автомата $q' \in P(Q)$.

Способы представления функции переходов

Командный способ. Каждую команду КА записывают в форме $F(q, t) = p$, где $q, p \in Q, t \in T$.

Табличный способ. Строки таблицы переходов соответствуют входным символам автомата $t \in T$, а столбцы – состояниям Q . Ячейки таблицы заполняются новыми состояниями, соответствующими значению функции $F(q, t)$. Неопределенным значениям функции переходов соответствуют пустые ячейки таблицы.

Графический способ. Строится диаграмма состояний автомата – неупорядоченный ориентированный помеченный граф. Вершины графа помечены именами состояний автомата. Дуга ведет из состояния q в состояние p и помечается списком всех символов $t \in T$, для которых $F(q, t) = p$. Вершина, соответствующая входному состоянию автомата, снабжается стрелкой. Заключи-

тельное состояние на графе обозначается двумя концентрическими окружностями.

Алгоритм 2.1. Построение КА по регулярной грамматике

Вход: регулярная грамматика $G = (V_T, V_N, P, S)$.

Выход: КА $M = (Q, T, F, H, Z)$.

Шаг 1. Пополнить грамматику правилом $A \rightarrow aN$, где $A \in V_N, a \in V_T$ и N - новый нетерминал, для каждого правила вида $A \rightarrow a$, если в грамматике нет соответствующего ему правила $A \rightarrow aB$, где $B \in V_N$.

Шаг 2. Начальный символ грамматики S принять за начальное состояние КА H . Из нетерминалов образовать множество состояний автомата $Q = V_N \cup \{N\}$, а из терминалов – множество символов входного алфавита $T = V_T$.

Шаг 3. Каждое правило $A \rightarrow aB$ преобразовать в функцию переходов $F(A, a) = B$, где $A, B \in V_N, a \in V_T$.

Шаг 4. Во множество заключительных состояний включить все вершины, помеченные символами $B \in V_N$ из правил вида $A \rightarrow aB$, для которых имеются соответствующие правила $A \rightarrow a$, где $A, B \in V_N, a \in V_T$.

Шаг 5. Если в грамматике имеется правило $S \rightarrow \varepsilon$, где S - начальный символ грамматики, то поместить S во множество заключительных состояний.

Шаг 6. Если получен НКА, то преобразовать его в ДКА.

Алгоритм 2.2. Преобразование НКА в ДКА

Вход: НКА $M = (Q, T, F, H, Z)$.

Выход: ДКА $M' = (Q', T, F', H, Z')$.

Шаг 1. Пометить первый столбец таблицы переходов M' ДКА начальным состоянием (множеством начальных состояний) НКА M .

Шаг 2. Заполняем очередной столбец таблицы переходов M' , помеченный символами D , для этого определяем те состояния M , которые могут быть достигнуты из каждого символа строки D при каждом входном символе x . Поместить каждое найденное множество R (в том числе \emptyset) в соответствующие позиции столбца D таблицы M' , т.е.:

$$F'(D, x) = \{s \mid s \in F(t, x) \text{ для некоторого } t \in D\}.$$

Шаг 3. Для каждого нового множества R (кроме \emptyset), полученного в столбце D таблицы переходов M' , добавить новый столбец в таблицу, помеченный R .

Шаг 4. Если в таблице переходов КА M' есть столбец с незаполненными позициями, то перейти к шагу 2.

Шаг 5. Во множество Z' ДКА M' включить каждое множество, помечающее столбец таблицы переходов M' и содержащее $q \in Z$ НКА M .

Шаг 6. Составить таблицу новых обозначений множеств состояний и определить ДКА M' в этих обозначениях.

Пример 2.1. Дана регулярная грамматика $G = (\{a, b\}, \{S, A, B\}, P, S)$ с правилами P : 1) $S \rightarrow aB \mid aA$; 2) $B \rightarrow bB \mid a$; 3) $A \rightarrow aA \mid b$. Построить по регулярной грамматике КА и преобразовать полученный автомат к детерминированному виду.

Решение задачи включает следующую последовательность действий.

1 Построим по регулярной грамматике КА.

1.1 Пополним грамматику правилами $A \rightarrow bN$ и $B \rightarrow aN$, где N - новый нетерминал.

1.2 Начальное состояние конечного автомата $H = S$. Множество состояний автомата $Q = V_N = \{S, A, B, N\}$, множество символов входного алфавита $T = V_T = \{a, b\}$.

1.3 Значения сформированной функции переходов даны в таблице 2.1.

Таблица 2.1 – Функция переходов автомата M

F	S	A	B	N
a	A, B	A	N	\emptyset
b	\emptyset	N	B	\emptyset

1.4 Множество заключительных состояний $Z = \{N\}$.

1.5 Для начального символа грамматики ϵ -правила отсутствуют.

Конечный автомат M - недетерминированный, граф НКА представлен на рисунке 2.1 слева.

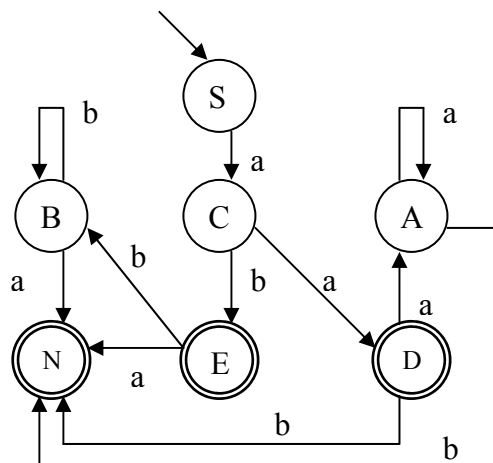
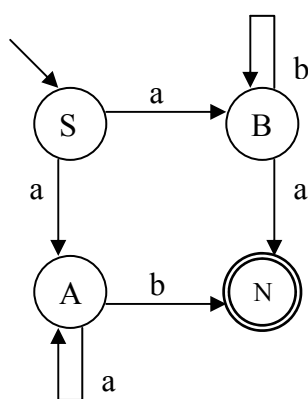


Рисунок 2.1 - Граф НКА (слева) и ДКА (справа) для P-грамматики

2 Построим по НКА M ДКА M' .

2.1 Строим таблицу переходов для ДКА M' (таблица 2.2).

Таблица 2.2 – Построение функции переходов для ДКА M'

Шаг	1	2	3	4	5	6	7
F	S	A, B	A, N	B, N	A	N	B
a	A, B	A, N	A	N	A	\emptyset	N
b	\emptyset	B, N	N	B	N	\emptyset	B

2.2 Во множество заключительных состояний автомата M' включим элементы $Z' = \{(A, N), (B, N), N\}$.

2.3 Введем следующие новые обозначения состояний автомата M' : $(A, B)=C$, $(A, N)=D$, $(B, N)=E$.

2.4 Искомый ДКА определяется следующей пятеркой объектов: $Q' = \{S, A, B, C, D, E, N\}$, $T = \{a, b\}$, функция переходов задана таблицей 2.3, $H = \{S\}$, $Z' = \{N, D, E\}$.

Граф полученного ДКА представлен на рисунке 2.1 справа.

Таблица 2.3 – Функция переходов для ДКА M'

F'	S	A	B	C	D	E	N
a	C	A	N	D	A	N	\emptyset
b	\emptyset	N	B	E	N	B	\emptyset

Постановка задачи к лабораторной работе № 2

Разработать программное средство, реализующее следующие функции:

- 1) ввод произвольной формальной грамматики с клавиатуры и проверка ее на принадлежность к классу регулярных грамматик;
- 2) построение по заданной регулярной грамматике конечного автомата;
- 3) преобразование недетерминированного конечного автомата к детерминированному конечному автомату;
- 4) вывод графа результирующего конечного автомата на экран.

Варианты индивидуального задания представлены в таблице 2.4.

Таблица 2.4 – Варианты индивидуального задания к лабораторной работе № 2

Вариант	Регулярная грамматика
1	$G = (\{S, C, D\}, \{0, 1\}, P, S)$, где P : 1) $S \rightarrow 1C \mid 0D$; 2) $C \rightarrow 0D \mid 0S \mid 1$; 3) $D \rightarrow 1C \mid 1S \mid 0$.
2	$G = (\{S, A, B, C\}, \{a, b, c\}, P, S)$, где P : 1) $S \rightarrow aA \mid bB \mid aC$; 2) $A \rightarrow bA \mid bB \mid c$; 3) $B \rightarrow aA \mid cC \mid b$; 4) $C \rightarrow bB \mid bC \mid a$.

Продолжение таблицы 2.4 – Варианты индивидуального задания к лабораторной работе № 2

Вариант	Регулярная грамматика
3	$G=(\{K, L, M, N\}, \{a, b, +, -, \perp\}, P, K)$, где P : 1) $K \rightarrow aL \mid bM$; 2) $L \rightarrow -N \mid -M$; 3) $M \rightarrow +N$; 4) $N \rightarrow aL \mid bM \mid \perp$.
4	$G=(\{X, Y, Z, W, V\}, \{0, 1, \sim, \#, \&\}, P, X)$, где P : 1) $X \rightarrow 0Y \mid 1Z \mid \varepsilon$; 2) $Y \rightarrow 0Z \mid \sim W \mid \#$; 3) $Z \rightarrow 1Y \mid 1W \mid 0V$; 4) $W \rightarrow 0W \mid 1W \mid \#$; 5) $V \rightarrow \&Z$.
5	$G=(\{K, L, M, N, Q, P, R, S\}, \{0, 1, *, \$, /\}, V, K)$, где V : 1) $K \rightarrow 1L \mid 0N$; 2) $L \rightarrow 0M \mid 0P \mid /Q$; 3) $N \rightarrow 1R \mid 1M \mid *S$; 4) $Q \rightarrow 1P$; 5) $P \rightarrow *L \mid \$$; 6) $M \rightarrow \$$; 7) $S \rightarrow 0R$; 8) $R \rightarrow /N \mid \$$.
6	$G=(\{E, A, B, C, D\}, \{0, 1, a, b, c\}, P, E)$, где P : 1) $E \rightarrow 0A \mid \varepsilon$; 2) $A \rightarrow aB \mid aD$; 3) $B \rightarrow bB \mid 1C \mid c$; 4) $D \rightarrow aD \mid 0C \mid c$.
7	$G=(\{X, Y, Z, V, W\}, \{0, 1, x, y, z\}, P, X)$, где P : 1) $X \rightarrow yY \mid zZ$; 2) $Y \rightarrow 1V$; 3) $Z \rightarrow 0W \mid 0Y$; 4) $V \rightarrow xZ \mid xW \mid 1$; 5) $W \rightarrow 1Y \mid 0$.
8	$G=(\{S, A, B, C, D\}, \{a, b, c, d, \perp\}, P, S)$, где P : 1) $S \rightarrow aA \mid bB$; 2) $A \rightarrow cC \mid \perp$; 3) $C \rightarrow cC \mid cA$; 4) $B \rightarrow dD \mid \perp$; 5) $D \rightarrow dD \mid dB$.
9	$G=(\{K, L, M, N, P\}, \{0, 1, \&, \%, a, b\}, C, K)$, где C : 1) $K \rightarrow 1M \mid \varepsilon$; 2) $M \rightarrow 0L \mid \&N \mid \&P$; 3) $L \rightarrow 1L \mid 0L \mid \%P$; 4) $N \rightarrow aN \mid bN \mid \%P$; 5) $P \rightarrow 1P \mid aP \mid 0$.
10	$G=(\{I, J, K, M, N\}, \{0, 1, \sim, !\}, P, I)$, где P : 1) $I \rightarrow 0J \mid 1K \mid 0M$; 2) $J \rightarrow \sim K \mid 0M$; 3) $K \rightarrow \sim M \mid 0J \mid 0N$; 4) $M \rightarrow 1K \mid !$; 5) $N \rightarrow 0I \mid 1I \mid !$.
11	$G=(\{S, A, B, C, D, E\}, \{a, b, c, d, e, \$, \perp\}, P, S)$, где P : 1) $S \rightarrow aA \mid bB \mid cC$; 2) $A \rightarrow dD$; 3) $B \rightarrow \#D \mid \$E$; 4) $D \rightarrow dD \mid dB \mid \perp$; 5) $C \rightarrow cE$; 6) $E \rightarrow eE \mid eB \mid \perp$.
12	$G=(\{X, Y, Z, V\}, \{(,), y, z, v\}, P, X)$, где P : 1) $X \rightarrow (Y \mid \varepsilon$; 2) $Y \rightarrow yY \mid zY \mid zZ$; 3) $Z \rightarrow zZ \mid vZ \mid vV$; 4) $V \rightarrow vV \mid)$.

3 Лабораторная работа № 3. Минимизация конечных автоматов

Цель: - закрепить понятия «недостижимые состояния автомата», «эквивалентные состояния автомата», «минимальный конечный автомат»;
- сформировать умения и навыки минимизации детерминированного конечного автомата.

Основы теории

Конечный автомат может содержать лишние состояния двух типов: недостижимые и эквивалентные состояния.

Определение 3.1. Два различных состояния q и q' в конечном автомате $M = (Q, T, F, H, Z)$ называются n -эквивалентными, $n \in N \cup \{0\}$, если, находясь в одном из этих состояний и получив на вход любую цепочку символов ω : $\omega \in V_T^*$, $|\omega| \leq n$, автомат может перейти в одно и то же множество конечных состояний.

Определение 3.2. Состояние q КА называется недостижимым, если к нему нет пути из начального состояния автомата.

Определение 3.3. КА, не содержащий недостижимых и эквивалентных состояний, называется приведенным или минимальным КА.

Алгоритм 3.1. Устранение недостижимых состояний КА

Вход: КА $M = (Q, T, F, H, Z)$.

Выход: КА $M' = (Q', T, F', H, Z')$.

Шаг 1. Поместить начальное состояние КА в список достижимых состояний Q_∂ , т.е. $Q_\partial^0 = H$.

Шаг 2. Для новых элементов списка достижимых состояний пополнить список группой их состояний-приемников, отсутствующих в нем, т.е. $Q_\partial^i = Q_\partial^{i-1} \cup \{p \mid \forall q \in Q_\partial^{i-1} \exists F(q, t) = p\}$.

Шаг 3. Повторить шаг 2, пока список достижимых состояний не перестанет меняться. То есть, если $Q_\partial^i \neq Q_\partial^{i-1}$, то $i := i + 1$, иначе $Q_\partial = Q_\partial^i$.

Шаг 4. Исключить из множества Q состояний КА все состояния, отсутствующие в списке Q_∂ достижимых состояний, т.е. $Q' = Q \setminus Q_\partial$.

Шаг 5. Исключить недостижимые заключительные состояния и пары функции переходов, содержащие недостижимые состояния, т.е. $Z' = Z \cap Q_\partial$, $F' = F - \{F(q, t) = p \mid q \in (Q - Q_\partial)\}$.

Пример 3.1. Устранить недостижимые состояния КА $M = (Q, T, F, H, Z)$, где $Q = \{A, B, C, D, E, F, G\}$, $T = \{a, b\}$, $H = \{A\}$, $Z = \{D, E\}$ и функция переходов задана таблицей 3.1. Граф исходного КА M представлен на рисунке 3.1.

Таблица 3.1 – Функция переходов конечного автомата M

F	A	B	C	D	E	F	G
a	B			C	B	D	F
b	C	D	E	E	D	G	E

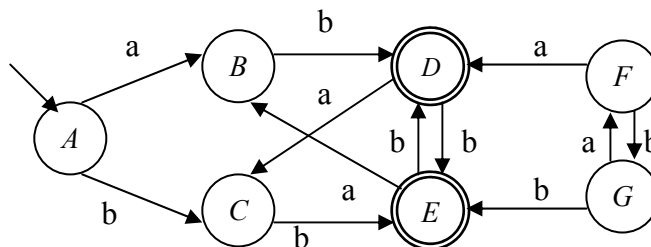


Рисунок 3.1 – Граф исходного конечного автомата M

Последовательность устранения недостижимых состояний КА имеет вид:

$$Q_0 = \{A\};$$

$$Q_1 = \{A, B, C\};$$

$$Q_2 = \{A, B, C, D, E\};$$

$$Q_3 = \{A, B, C, D, E\}; \text{ т.к. } Q_2 = Q_3, \text{ то } Q_d = \{A, B, C, D, E\}.$$

$$Q_n = \{F, G\}; Q' = \{A, B, C, D, E\}; Z' = \{D, E\}.$$

Функция переходов автомата M' представлена в таблице 3.2.

Таблица 3.2 - Функция переходов автомата M'

F	A	B	C	D	E
a	B			C	B
b	C	D	E	E	D

Граф КА M' после устранения недостижимых состояний представлен на рисунке 3.2.

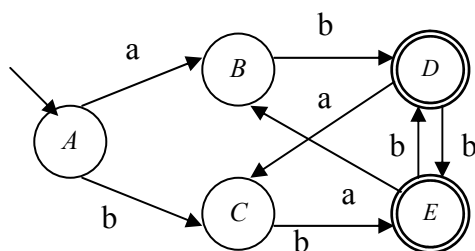


Рисунок 3.2 - Граф КА M' после устранения недостижимых состояний

Алгоритм 3.2. Объединение эквивалентных состояний КА

Вход: КА $M' = (Q', T, F', H, Z')$ без недостижимых состояний.

Выход: минимальный КА $M'' = (Q'', T, F'', H, Z'')$.

Шаг 1. На первом шаге строим нулевое разбиение $R(0)$, состоящее из двух классов эквивалентности: заключительные состояния КА - Z и не заключительные - $Q-Z$.

Шаг 2. На очередном шаге построения разбиения $R(n)$ в классы эквивалентности включить те состояния, которые по одинаковым входным символам переходят в $n-1$ эквивалентные состояния, т.е.

$$R(n) = \{r_i(n) : \{q_{ij} \in Q : \forall t \in T F(q_{ij}, t) \subseteq r_j(n-1)\} \forall i, j \in N\}.$$

Шаг 3. До тех пор, пока $R(n) \neq R(n-1)$ полагаем $n:=n+1$ и идем к шагу 2.

Шаг 4. Переобозначить оставшиеся неразбитые группы состояний и включить их в таблицу новых обозначений состояний автомата.

Шаг 5. Определить эквивалентный КА M'' в новых обозначениях.

Пример 3.2. Минимизировать конечный автомат из примера 3.1.

Последовательность построения разбиений будет иметь вид:

$$R(0) = \{\{A, B, C\}, \{D, E\}\}, n = 0;$$

$$R(1) = \{\{A\}, \{B, C\}, \{D, E\}\}, n = 1;$$

$$R(2) = \{\{A\}, \{B, C\}, \{D, E\}\}, n=2.$$

Т.к. $R(1) = R(2)$, то искомое разбиение построено.

Переобозначим оставшиеся неразбитые группы состояний:

$$X=\{B, C\}, Y=\{D, E\}.$$

Получим минимальный автомат M'' , где $Q''=\{A, X, Y\}$, $Z''=\{Y\}$.

Функция переходов автомата M'' представлена в таблице 3.3.

Таблица 3.3 - Функция переходов автомата M''

F''	A	X	Y
a	X		X
b	X	Y	Y

Граф переходов конечного автомата после его минимизации показан на рисунке 3.3.

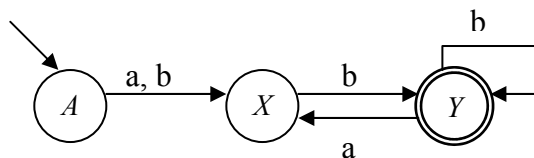


Рисунок 3.3 – Граф минимального КА M''

Постановка задачи к лабораторной работе № 3

Разработать программное средство, реализующее следующие функции:

- 1) ввод исходного конечного автомата и вывод на экран его графа;
- 2) устранение недостижимых состояний конечного автомата;

3) исключение эквивалентных состояний конечного автомата;

4) вывод на экран графа минимального конечного автомата.

Разработать серию контрольных примеров для тестирования реализованных алгоритмов.

Варианты индивидуальных заданий к лабораторной работе № 3 представлены на рисунке 3.4.

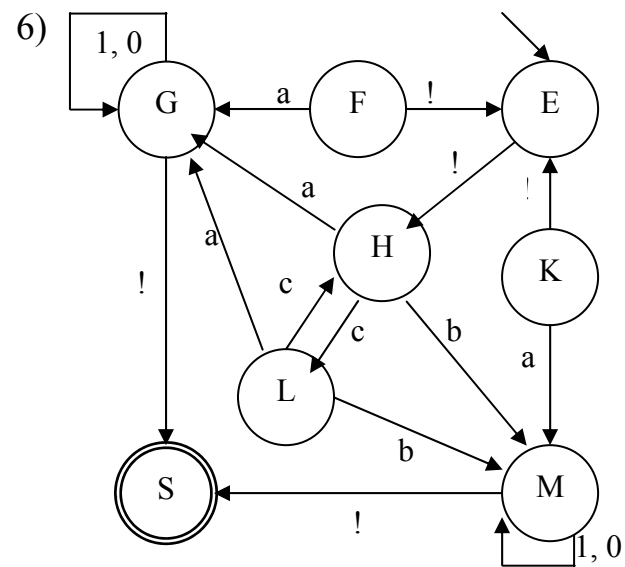
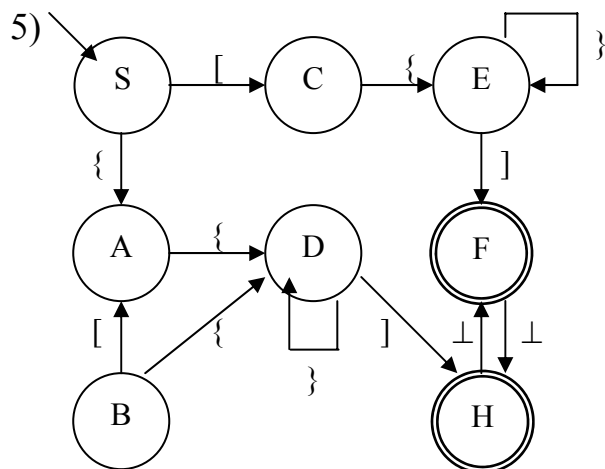
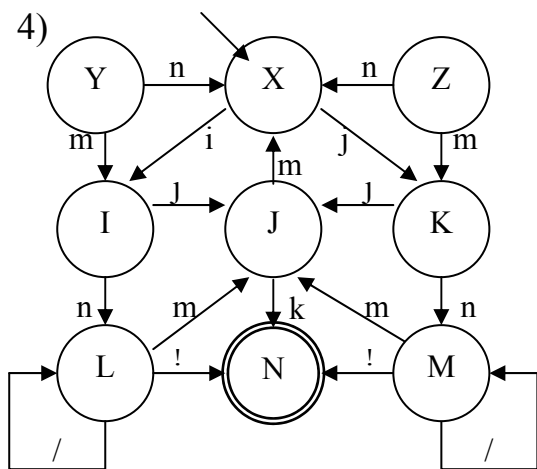
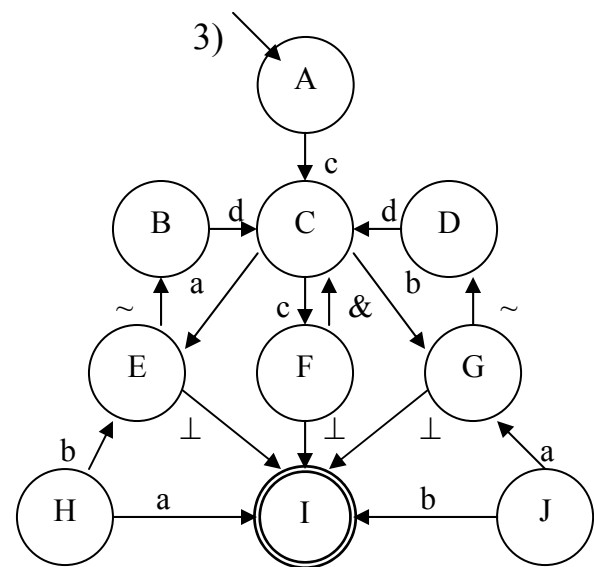
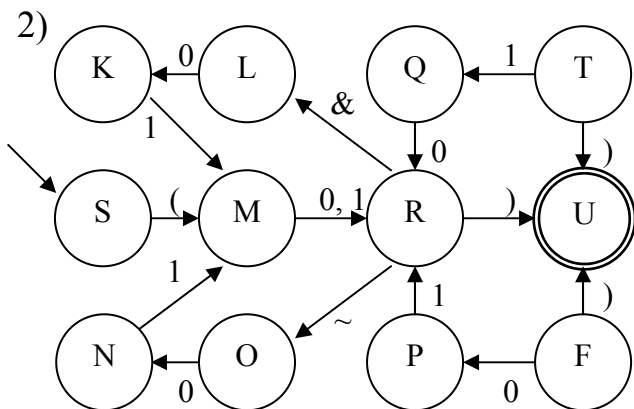
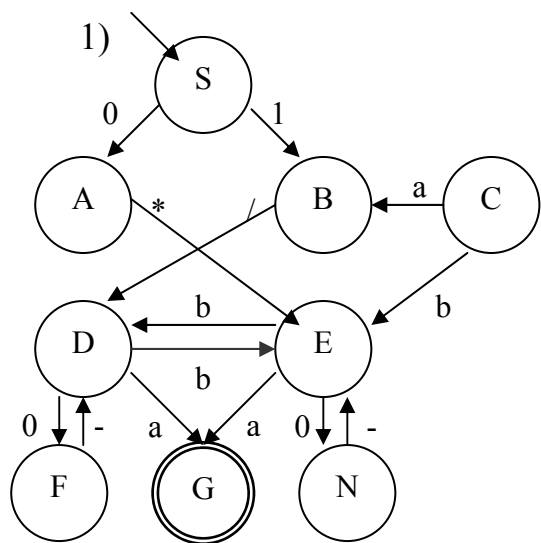


Рисунок - 3.4 – Варианты индивидуальных заданий к лабораторной работе № 3

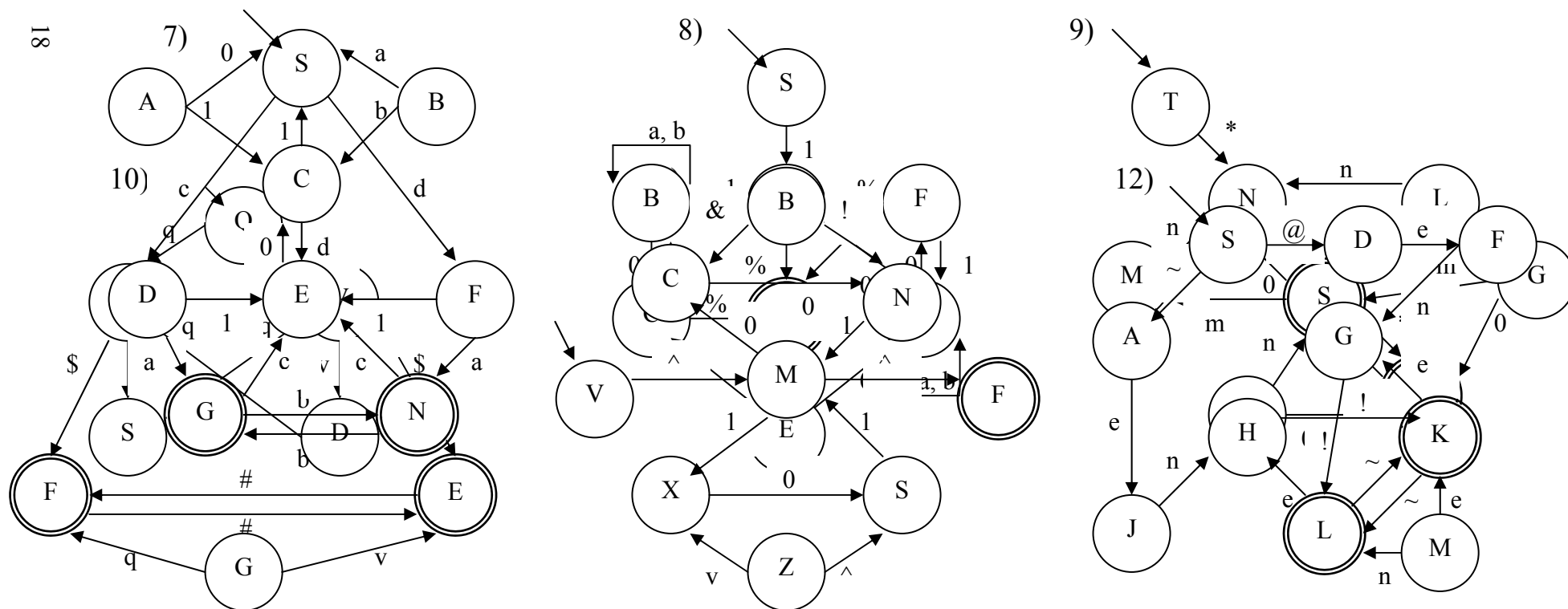


Рисунок - 3.4 – Варианты индивидуальных заданий к лабораторной работе № 3, лист 2

4 Лабораторная работа № 4. Эквивалентные преобразования контекстно-свободных грамматик

Цель: - закрепить понятия «эквивалентные грамматики», «приведенная КС-грамматика»;

- сформировать умения и навыки эквивалентных преобразований контекстно-свободных грамматик.

Основы теории

Определение 4.1. КС-грамматика называется приведенной, если она не имеет циклов, ε -правил и бесполезных символов.

Рассмотрим основные алгоритмы приведения КС-грамматик.

Перед всеми другими исследованиями и преобразованиями КС-грамматик выполняется проверка существования языка грамматики.

Алгоритм 4.1. Проверка существования языка грамматики

Вход: КС-грамматика $G = (V_T, V_N, P, S)$.

Выход: заключение о существовании или отсутствии языка грамматики.

Определим множество нетерминалов, порождающих терминальные строки $N = \{Z \mid Z \in V_N, Z \Rightarrow^* x, x \in V_T^*\}$.

Шаг 1. Положить $N_0 = \emptyset$.

Шаг 2. Вычислить $N_i = N_{i-1} \cup \{A \mid (A \rightarrow \alpha) \in P \text{ и } \alpha \in (N_{i-1} \cup V_T)^*\}$.

Шаг 3. Если $N_i \neq N_{i-1}$, то положить $i=i+1$ и перейти к пункту 2, иначе считать $N = N_i$.

Если $S \in N$, то выдать сообщение о том, что язык грамматики существует, иначе сообщить об отсутствии языка.

Пример 4.1. Дана грамматика $G = (\{0, 1\}, \{S, A, B\}, P, S)$, где множество правил P : 1) $S \rightarrow AB$; 2) $A \rightarrow 0A$; 3) $A \rightarrow 0$; 4) $B \rightarrow 1$. Построим последовательность приближений множества N :

$$N_0 = \emptyset;$$

$$N_1 = \{A, B\};$$

$$N_2 = \{S, A, B\};$$

$$N_3 = \{S, A, B\}.$$

Т.к. $N_2 = N_3$, то $N = \{S, A, B\}$, следовательно, язык грамматики существует, потому что начальный символ $S \in N$.

Определение 4.2. Бесполезными символами грамматики называют:

а) нетерминалы, не порождающие терминальных строк, т.е. множество символов

$$\{X \mid X \in V_N, \neg \exists (X \Rightarrow^* x), x \in V_T^*\};$$

б) недостижимые нетерминалы, порождающие терминальные строки, т.е. множество символов

$$\{X \mid X \in V_N, \neg \exists (S \Rightarrow * \alpha X \beta), \exists (X \Rightarrow * x); \alpha, \beta \in V^*; x \in V_T^*\};$$

в) недостижимые терминалы, т.е. множество символов

$$\{X \mid X \in V_T, \neg \exists (S \Rightarrow * \alpha X \beta); \alpha, \beta \in V^*\}.$$

Алгоритм 4.2. Устранение нетерминалов, не порождающих терминальных строк

Вход: КС-грамматика $G = (V_T, V_N, P, S)$.

Выход: КС-грамматика $G' = (V_T, V'_N, P', S)$, такая, что $L(G') = L(G)$ и для всех $Z \in V'_N$ существуют выводы $Z \Rightarrow * x$, где $x \in V_T^*$.

Шаг 1. Определить множество нетерминалов, порождающих терминальные строки, с помощью алгоритма 4.1.

Шаг 2. Вычислить $V'_N = V_N \cap N$, $N_B = V_N - V'_N$, $P' = P - P_B$, где $P_B \subseteq P$ - это множество правил, содержащих бесполезные нетерминалы $X \in N_B$.

Пример 4.2. Дана грамматика $G = (\{a, b, c\}, \{S, A, B, C\}, P, S)$ с правилами P : 1) $S \rightarrow ab$; 2) $S \rightarrow AC$; 3) $A \rightarrow AB$; 4) $B \rightarrow b$; 5) $C \rightarrow cb$.

Преобразуем ее в эквивалентную грамматiku G' по алгоритму 4.2:

$$N_0 = \emptyset;$$

$$N_1 = \{S, B, C\};$$

$$N_2 = \{S, B, C\}.$$

Т.к. $N_1 = N_2$, то $N = \{S, B, C\}$. После удаления бесполезных нетерминалов и правил вывода, получим грамматiku $G' = (\{a, b, c\}, \{S, B, C\}, P', S)$ с правилами P' : 1) $S \rightarrow ab$; 2) $B \rightarrow b$; 5) $C \rightarrow cb$.

Алгоритм 4.3. Устранение недостижимых символов

Вход: КС-грамматика $G = (V_T, V_N, P, S)$.

Выход: КС-грамматика $G' = (V'_T, V'_N, P', S)$, такая, что $L(G') = L(G)$ и для всех $Z \in V'$ существует вывод $S \Rightarrow * \alpha Z \beta$, где $\alpha, \beta \in (V')^*$.

Определим множество достижимых символов Z грамматики G , т.е. множество

$$W = \{Z \mid Z \in V, \exists (S \Rightarrow * \alpha Z \beta); \alpha, \beta \in V^*\}.$$

Шаг 1. Положить $W_0 = S$.

Шаг 2. Вычислить очередное приближение следующим образом:

$$W_i = W_{i-1} \cup \{X \mid X \in V, (A \rightarrow \alpha X \beta) \in P, A \in W_{i-1}; \alpha, \beta \in V^*\}.$$

Шаг 3. Если $W_i \neq W_{i-1}$, то положить $i:=i+1$ и перейти к шагу 2, иначе считать $W = W_i$.

Шаг 4. Вычислить $V'_N = V_N \cap W, V'_T = V_T \cap W, V_B = V - W, P' = P - P_B$, где $P_B \subseteq P$ - это множество правил, содержащих недостижимые символы $X \in V_B$.

Пример 4.3. Дана грамматика $G = (\{a, b, c\}, \{S, B, C\}, P, S)$ с правилами P' : 1) $S \rightarrow ab$; 2) $B \rightarrow b$; 5) $C \rightarrow cb$.

Преобразуем ее в эквивалентную грамматику G' по алгоритму 4.3:

$$W_0 = \{S\};$$

$$W_1 = \{S, a, b\};$$

$$W_2 = \{S, a, b\}.$$

Т.к. $W_1=W_2$, то $W=\{S, a, b\}$. Множество недостижимых символов $V_B = \{B, C, c\}$. Тогда после удаления недостижимых символов, получим грамматику $G' = (\{a, b\}, \{S\}, P, S)$ с правилом P' : $S \rightarrow ab$.

Алгоритм 4.4. Устранение ε -правил

Вход: КС-грамматика $G = (V_T, V_N, P, S)$.

Выход: Эквивалентная КС-грамматика $G' = (V_T, V'_N, P', S')$ без ε -правил для всех нетерминальных символов, кроме начального, который не должен встречаться в правых частях правил грамматики.

Шаг 1. В исходной грамматике G найти ε -порождающие нетерминальные символы $A \in V_N$, такие, что $A \Rightarrow^* \varepsilon$.

1.1 Положить $N_0 = \{A \mid (A \rightarrow \varepsilon) \in P\}$.

1.2 Вычислить $N_i = N_{i-1} \cup \{B \mid (B \rightarrow \alpha) \in P, \alpha \in N_{i-1}^*\}$.

1.3 Если $N_i \neq N_{i-1}$, то положить $i:=i+1$ и перейти к пункту 1.2, иначе считать $N = N_i$.

Шаг 2. Из множества P правил исходной грамматики G перенести во множество P' все правила, за исключением ε -правил, т.е. $P' = P - \{(A \rightarrow \varepsilon) \in P \text{ для всех } A \in V_N\}$.

Шаг 3. Пополнить множество P' правилами, которые получаются из каждого правила этого множества путем исключения всевозможных комбинаций ε -порождающих нетерминалов в правой части. Полученные при этом ε -правила во множество P' не включать.

Шаг 4. Если $S \in N$, то $P' = P \cup \{S' \rightarrow \varepsilon, S' \rightarrow S\}, V'_N = V_N \cup S'$, где $V \cap \{S'\} = \emptyset$; иначе $V'_N = V_N, S' = S$.

Пример 4.4. Дана грамматика $G = (\{0, 1\}, \{S, A, B\}, P, S)$ с правилами P :
 1) $S \rightarrow AB$; 2) $A \rightarrow 0A \mid \varepsilon$; 3) $B \rightarrow 1B \mid \varepsilon$. Преобразуем ее в эквивалентную грамматику по алгоритму 4.4.

Шаг 1. $N_0 = \{A, B\}$;

$N_1 = \{S, A, B\}$;

$N_2 = \{S, A, B\}$.

Т.к. $N_1 = N_2$, то искомое множество построено и $N = \{S, A, B\}$.

Шаг 2, 3. Множество P' : 1) $S \rightarrow AB \mid A \mid B$; 2) $A \rightarrow 0A \mid 0$; 3) $B \rightarrow 1B \mid 1$.

Шаг 4. Т.к. $S \in N$, то введем новый нетерминал C и пополним множество P' правилом вида $C \rightarrow S \mid \varepsilon$. Результирующая грамматика будет иметь вид: $G' = (\{0, 1\}, \{S, A, B, C\}, P', C)$ с правилами P' : 1) $C \rightarrow S \mid \varepsilon$; 2) $S \rightarrow AB \mid A \mid B$; 3) $A \rightarrow 0A \mid 0$; 4) $B \rightarrow 1B \mid 1$.

Алгоритм 4.5. Устранение цепных правил

Вход: КС-грамматика $G = (V_T, V_N, P, S)$.

Выход: Эквивалентная КС-грамматика $G' = (V_T, V'_N, P', S')$ без цепных правил, т.е. правил вида $A \rightarrow B$, где $A, B \in V_N$.

Шаг 1. Для каждого нетерминала A вычислить множество выводимых из него нетерминалов, т.е. множество $N^A = \{B \mid A \Rightarrow^* B, \text{ где } B \in V_N\}$.

1.1 Положить $N_0^A = \{A\}$.

1.2 Вычислить $N_i^A = N_{i-1}^A \cup \{C \mid (B \rightarrow C) \in P, B \in N_{i-1}^A, C \in V_N\}$.

1.3 Если $N_i^A \neq N_{i-1}^A$, то положить $i := i + 1$ и перейти к пункту 1.2, иначе считать $N^A = N_i^A$.

Шаг 2. Построить множество P' так: если $(B \rightarrow \alpha) \in P$ не является цепным правилом ($\alpha \notin V_N$), то включить в P' правило $A \rightarrow \alpha$ для каждого A , такого, что $B \in N^A$.

Пример 4.5. Грамматика $G = (\{+, n\}, \{L, M, N\}, P, L)$ с правилами P :
 1) $L \rightarrow M$; 2) $M \rightarrow N$; 3) $N \rightarrow N+ \mid n$. Преобразуем ее в эквивалентную грамматику G' по алгоритму 4.5.

Шаг 1. $N_0^L = \{L\}$;

$N_1^L = \{L, M\}$;

$N_2^L = \{L, M, N\}$;

$N_3^L = \{L, M, N\}$.

Т.к. $N_2^L = N_3^L$, то $N^L = \{L, M, N\}$.

$N_0^M = \{M\}$;

$$N_1^M = \{M, N\};$$

$$N_2^M = \{M, N\}.$$

Т.к. $N_1^M = N_2^M$, то $N^M = \{M, N\}$.

$$N_0^N = \{N\};$$

$$N_1^N = \{N\}.$$

Т.к. $N_1^N = N_0^N$, то $N^N = \{N\}$.

Шаг 2. Преобразовав правила вывода грамматики, получим грамматику $G' = (\{+, n\}, \{L, M, N\}, P', L)$ с правилами P' :

$$1) L \rightarrow N+|n; \quad 2) M \rightarrow N+|n; \quad 3) N \rightarrow N+|n.$$

Алгоритм 4.6. Устранение левой факторизации правил

Вход: КС-грамматика $G = (V_T, V_N, P, S)$.

Выход: Эквивалентная КС-грамматика $G' = (V_T, V'_N, P', S')$ без одинаковых префиксов в правых частях правил, определяющих нетерминалы.

Шаг 1. Записать все правила для нетерминала X , имеющие одинаковые префиксы $\alpha \in V^*$, в виде одного правила с альтернативами: $X \rightarrow \alpha\beta_1 | \alpha\beta_2 | \dots | \alpha\beta_n$; $\beta_1, \beta_2, \dots, \beta_n \in V^*$.

Шаг 2. Вынести за скобки влево префикс α в каждой строке-альтернативе: $X \rightarrow \alpha(\beta_1 | \beta_2 | \dots | \beta_n)$.

Шаг 3. Обозначить новым нетерминалом Y выражение, оставшееся в скобках: $X \rightarrow \alpha Y, Y \rightarrow \beta_1 | \beta_2 | \dots | \beta_n$.

Шаг 4. Пополнить множество нетерминалов новым нетерминалом Y и заменить правила, подвергшиеся факторизации, новыми правилами для X и Y .

Шаг 5. Повторить шаги 1-4 для всех нетерминалов грамматики, для которых это возможно и необходимо.

Пример 4.6. Дана грамматика $G = (\{k, l, m, n\}, \{S\}, P, S)$ с правилами P : 1) $S \rightarrow kSl$; 2) $S \rightarrow kSm$; 3) $S \rightarrow n$. Преобразуем ее в эквивалентную грамматику G' по алгоритму 4.6:

Шаг 1. $S \rightarrow kSl | kSm | n$.

Шаг 2. $S \rightarrow kS(l | m) | n$.

Шаг 3,4. Пополнив множество нетерминалов новым нетерминалом C и заменив правила, подвергшиеся факторизации, получим грамматику $G' = (\{k, l, m, n\}, \{S, C\}, P', S)$ с правилами P' : 1) $S \rightarrow kSC$; 2) $S \rightarrow n$; 3) $C \rightarrow l$; 4) $C \rightarrow m$.

Алгоритм 4.7. Устранение прямой левой рекурсии

Вход: КС-грамматика $G = (V_T, V_N, P, S)$.

Выход: Эквивалентная КС-грамматика $G' = (V_T, V'_N, P', S')$ без прямой левой рекурсии, т.е. без правил вида $A \rightarrow A\alpha$, $A \in V_N$, $\alpha \in V^*$.

Шаг 1. Вывести из грамматики все правила для рекурсивного нетерминала X :

$$X \rightarrow X\alpha_1 \mid X\alpha_2 \mid \dots \mid X\alpha_m \quad (X \in V_N; \alpha_1, \alpha_2, \dots, \alpha_m \in V^*)$$

$$X \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n \quad (\beta_1, \beta_2, \dots, \beta_n \in V^*).$$

Шаг 2. Внести новый нетерминал Y так, чтобы он описывал любой «хвост» строки, порождаемой рекурсивным нетерминалом X :

$$Y \rightarrow \alpha_1 Y \mid \alpha_2 Y \mid \dots \mid \alpha_m Y$$

$$Y \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_m.$$

Шаг 3. Заменить в рекурсивном правиле для X правую часть, используя новый нетерминал и все нерекурсивные правила для X так, чтобы генерируемый язык не изменился:

$$X \rightarrow \beta_1 Y \mid \beta_2 Y \mid \dots \mid \beta_n Y$$

$$X \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$$

$$Y \rightarrow \alpha_1 Y \mid \alpha_2 Y \mid \dots \mid \alpha_m Y$$

$$Y \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_m.$$

Шаг 4. Пополнить множество нетерминалов грамматики новым нетерминалом Y . Пополнить множество правил грамматики правилами, полученными на шаге 3.

Шаг 5. Повторить действия шагов 1-4 для всех рекурсивных нетерминалов грамматики, после чего полученные множества нетерминалов и правил принять в качестве V'_N и P' .

Пример 4.7. Дана грамматика $G = (\{a, b, c, d, z\}, \{S, A, B, C\}, P, S)$ с правилами P : 1) $S \rightarrow Aa$; 2) $A \rightarrow Bb$; 3) $B \rightarrow Cc \mid d$; 4) $C \rightarrow Ccbz \mid dbz$. После устранения прямой левой рекурсии получим эквивалентную грамматику $G' = (\{a, b, c, d, z\}, \{S, A, B, C, Z\}, P', S)$ с правилами P' :

$$1) S \rightarrow Aa; \quad 2) A \rightarrow Bb; \quad 3) B \rightarrow Cc \mid d; \quad 4) C \rightarrow dbzZ \mid dbz; \quad 5) Z \rightarrow cbzZ \mid cbz.$$

Постановка задачи к лабораторной работе № 4

Разработать программное средство, автоматизирующее процесс эквивалентного преобразования КС-грамматик. Программное средство должно выполнять следующие функции:

- 1) организация ввода грамматики и проверка ее на принадлежность к классу КС-грамматик;
- 2) проверка существования языка КС-грамматики;

3) реализация эквивалентных преобразований грамматики, направленных на удаление:

- а) бесполезных символов;
- б) недостижимых символов;
- в) ε -правил;
- г) цепных правил;
- д) левой факторизации правил;
- е) прямой левой рекурсии.

Варианты индивидуальных заданий представлены в таблице 4.1.

Таблица 4.1 – Варианты индивидуальных заданий к лабораторной работе № 4 и 5

Вариант	Контекстно-свободная грамматика
1	$G=(\{S, A, B, D, E\}, \{a, b, c, e\}, P, S)$, где P : 1) $S \rightarrow AB \mid \varepsilon$; 2) $A \rightarrow Aa \mid S \mid a$; 3) $B \rightarrow bD \mid bS \mid b$; 4) $D \rightarrow ccD$; 5) $E \rightarrow eE \mid e$.
2	$G=(\{E, T, F, G, H\}, \{+, -, *, /, n, m, h\}, P, E)$, где P : 1) $E \rightarrow T \mid E+T \mid E-T \mid \varepsilon$; 2) $T \rightarrow F \mid F*T \mid F/T \mid \varepsilon$; 3) $F \rightarrow G \mid Fn \mid n$; 4) $G \rightarrow Gm$; 5) $H \rightarrow Hh \mid h$.
3	$G=(\{S, R, T, X, Y\}, \{a, b, p, g, y\}, P, S)$, где P : 1) $S \rightarrow R \mid T$; 2) $R \rightarrow pX \mid paR \mid paT \mid \varepsilon$; 3) $T \rightarrow Tg \mid g$; 4) $X \rightarrow aXb$; 5) $Y \rightarrow aYa \mid y$.
4	$G=(\{Q, A, B, C, D\}, \{a, b, c, d\}, P, Q)$, где P : 1) $Q \rightarrow acA \mid acB \mid \varepsilon$; 2) $B \rightarrow A \mid Cb \mid \varepsilon$; 3) $A \rightarrow Aa \mid Ab \mid a$; 4) $C \rightarrow dCc$; 5) $D \rightarrow dc$.
5	$G=(\{R, T, F, G, K\}, \{m, i, j, k, ^, \sim, \perp\}, P, R)$, где P : 1) $R \rightarrow R \sim T \perp \mid R^{\wedge} T \perp \mid \varepsilon$; 2) $T \rightarrow F \mid Fi \mid Fj \mid Gk \mid \varepsilon$; 3) $G \rightarrow GkG$; 4) $K \rightarrow Ki \mid Km \mid m$.
6	$G=(\{S, X, Y, Z, K\}, \{x, y, z, k, \#, \$\}, P, S)$, где P : 1) $S \rightarrow X \mid Y \mid Z$; 2) $X \rightarrow x\#X \mid x\#Y \mid \varepsilon$; 3) $Y \rightarrow Yy\$ \mid Yz\$ \mid \$ \mid \varepsilon$; 4) $Z \rightarrow Zz\$$; 5) $K \rightarrow Kk\$ \mid k\$$.
7	$G=(\{S, L, M, P, N\}, \{n, m, l, p, @, \perp\}, V, S)$, где V : 1) $S \rightarrow @nL \mid @mM \mid P$; 2) $L \rightarrow M \mid Ll \perp \mid Lm \perp \mid \varepsilon$; 3) $M \rightarrow L \mid Mm \mid mm$; 4) $N \rightarrow pN@ \mid @$; 5) $P \rightarrow nmP$.
8	$G=(\{X, Y, Z, K, L\}, \{a, b, l, =, <, >, \wedge, \vee, \neg\}, V, X)$, где V : 1) $X \rightarrow Y \mid Y=Y \mid Y<Y \mid Y>Y \mid K$; 2) $Y \rightarrow Y \wedge Z \mid Y \vee Z \mid \varepsilon$; 3) $Z \rightarrow \neg a \mid \neg b \mid \varepsilon$; 4) $K \rightarrow \neg K$; 5) $L \rightarrow l \mid a \mid b$.
9	$G=(\{Q, A, B, C, D\}, \{0, 1, -\}, P, Q)$, где P : 1) $Q \rightarrow 01A \mid 01B \mid A$; 2) $A \rightarrow 0B1 \mid B \mid 1 \mid \varepsilon$; 3) $B \rightarrow BA0 \mid B1 \mid C \mid \varepsilon$;

	4) $C \rightarrow 0C11$; 5) $D \rightarrow -D1 \mid -0 \mid -1$.
10	$G = (\{R, T, U, W, V\}, \{0, 1, +, -, *, /\}, P, R)$, где P : 1) $R \rightarrow T1T \mid T1U \mid W \mid \varepsilon$; 2) $T \rightarrow U \mid T01 \mid T10 \mid \varepsilon$; 3) $U \rightarrow +U \mid +0 \mid +1$ 4) $W \rightarrow W-W \mid W+W$; 5) $V \rightarrow *0 \mid /1$.

Продолжение таблицы 4.1 – Варианты индивидуальных заданий к лабораторной работе № 4 и 5

Вариант	Контекстно-свободная грамматика
11	$G = (\{S, R, T, F, E\}, \{a, b, k, \{, [, \},], \perp\}, P, S)$, где P : 1) $S \rightarrow \{R \mid [R$; 2) $R \rightarrow Ra\} \mid Ra \mid a \mid T \mid F \mid \varepsilon$; 3) $F \rightarrow \{F\} \mid bb$; 4) $T \rightarrow [T$; 5) $E \rightarrow k\perp$.
12	$G = (\{Y, K, M, L, S\}, \{a, b, *, /, ^\}, P, Y)$, где P : 1) $Y \rightarrow KS \mid KM$; 2) $K \rightarrow K^* \mid K/ \mid S$; 3) $S \rightarrow Sa/ \mid Sb/ \mid \varepsilon$; 4) $M \rightarrow *M^*$; 5) $L \rightarrow L^{\wedge} \mid ^{\wedge}a$.

5 Лабораторная работа № 5. Построение автомата с магазинной памятью по контекстно-свободной грамматике

Цель: - закрепить понятия «автомат с магазинной памятью (МП-автомат)», «расширенный МП-автомат», «конфигурация МП-автомата»; «строка и язык, допускаемые МП-автоматом»;

- сформировать умения и навыки построения МП-автомата и расширенного МП-автомата по КС-грамматике, разбора входной строки с помощью МП-автомата.

Основы теории

КС-языки можно распознавать с помощью автомата с магазинной памятью (МП-автомата).

Определение 5.1. МП-автомат можно представить в виде семерки:

$$M = (Q, T, N, F, q_0, N_0, Z), \quad (5.1)$$

где Q – конечное множество состояний автомата;

T – конечный входной алфавит;

N – конечный магазинный алфавит;

F – магазинная функция, отображающая множество $(Q \times (T \cup \{\varepsilon\}) \times N)$

во множество всех подмножеств множества $Q \times N^*$, т.е.

$$F : (Q \times (T \cup \{\varepsilon\}) \times N) \rightarrow P(Q \times N^*);$$

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

N_0 – начальный символ магазина, $N_0 \in T$;

Z – множество заключительных состояний автомата, $Z \subseteq Q$.

Определение 5.2. Конфигурацией МП-автомата называется тройка вида:

$$(q, \omega, \alpha) \in (Q \times T^* \times N^*), \quad (5.2)$$

где q – текущее состояние автомата, $q \in Q$;

ω – часть входной строки, первый символ которой находится под входной головкой, $\omega \in T^*$;

α – содержимое магазина, $\alpha \in N^*$.

Общая схема МП-автомата представлена на рисунке 5.1.

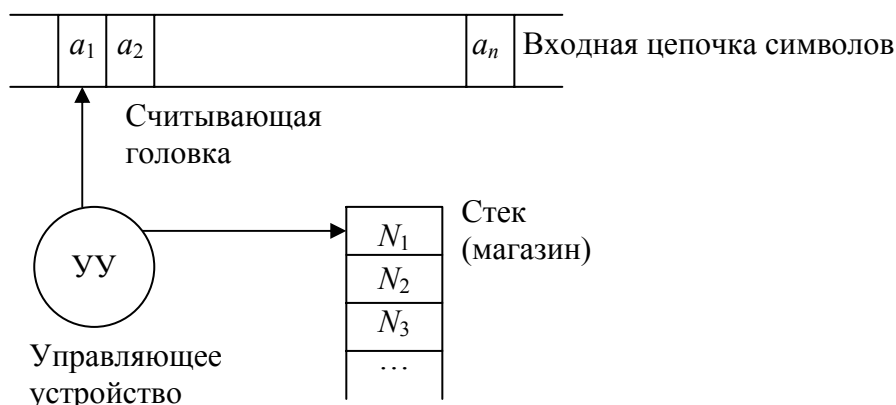


Рисунок 5.1 – Схема МП-автомата

Алгоритм 5.1. Функционирование МП-автомата

Начальной конфигурацией МП-автомата является конфигурация (q_0, ω, N_0) .

Шаг работы МП-автомата будем представлять в виде отношения непосредственного следования конфигураций (обозначается « $|\Rightarrow$ ») и отношения достижимости конфигураций (обозначается « $|\equiv^*$ »). Если одним из значений магазинной функции $F(q \in Q, t \in (T \cup \{\varepsilon\}), S \in N)$ является $(q' \in Q, \gamma \in N^*)$, то записывается $(q, t\omega, S\alpha) |\Rightarrow (q', \omega, \gamma\alpha)$. При этом возможны следующие варианты.

1) Случай $t \in T$. Автомат находится в текущем состоянии q , читает входной символ t , имеет в вершине стека символ S . Он переходит в очередное состояние q' , сдвигает входную головку на ячейку вправо и заменяет верхний символ S строкой γ магазинных символов. Вариант $\gamma = \varepsilon$ означает, что S удаляется из стека.

2) Случай $t = \varepsilon$. Отличается от первого случая тем, что входной символ t просто не принимается во внимание, и входная головка не сдвигается. Такой шаг работы МП-автомата называется ε -шагом, который может выполняться даже после завершения чтения входной строки.

Заключительной конфигурацией МП-автомата является конфигурация (q, ε, α) , где $q \in Z$.

Определение 5.3. МП-автомат допускает входную строку ω , если существует путь по конфигурациям $(q_0, \omega, N_0) |\equiv^* (q, \varepsilon, \alpha)$ для некоторых $q \in Z$ и $\alpha \in N^*$.

Определение 5.4. Язык L , распознаваемый (принимаемый) МП-автоматом M определяется как множество вида:

$$L(M) = \{\omega \mid \omega \in T^* \text{ и } (q_0, \omega, N_0) |\equiv^* (q, \varepsilon, \alpha) \text{ для некоторых } q \in Z \text{ и } \alpha \in N^*\}.$$

Определение 5.5. МП-автомат с магазинной функцией $F : (Q \times (T \cup \{\varepsilon\}) \times N^*) \rightarrow P(Q \times N^*)$ называется расширенным МП-автоматом, т.е. автоматом, который может заменять цепочку символов конечной длины в верхушке стека на другую цепочку символов конечной длины.

Существуют КС-языки, МП-автоматы и расширенные МП-автоматы, определяющие один и тот же язык.

Алгоритм 5.2. Построение МП-автомата по КС-грамматике

Построим МП-автомат, выполняющий левосторонний разбор. Данный автомат обладает только одним состоянием и принимает входную строку опустошением магазина. Стек используется для размещения текущей sentenции, первоначально это начальный символ грамматики. Очередная sentenция получается заменой верхнего нетерминала стека.

Вход: КС-грамматика $G = (V_T, V_N, P, S)$.

Выход: МП-автомат $M = (Q, T, N, F, q_0, n_0, Z)$ такой, что $L(M) = L(G)$.

Шаг 1. Положить $Q = \{q\}$, $q_0 = q$, $Z = \emptyset$, $N = V_T \cup V_N$, $T = V_T$, $N_0 = S$.

Шаг 2. Для каждого правила вида $(A \rightarrow \beta) \in P$, где $\beta \in V^*$, сформировать магазинную функцию вида $F(q, \varepsilon, A) = (q, \beta)$. Эти функции предписывают замещать нетерминал в вершине стека по правилу грамматики.

Шаг 3. Для каждого $t \in V_T$ сформировать магазинную функцию вида $F(q, t, t) = (q, \varepsilon)$, которая выталкивает из стека символ, совпадающий с входным, и перемещает читающую головку. Эти функции обеспечивают опустошение стека.

Пример 5.1. Дана КС-грамматика:

$G(\{+, (,), a\}, \{S, A\}, \{S \rightarrow S+A \mid A, A \rightarrow (S) \mid a\}, \{S\})$. Последовательность построения МП-автомата будет иметь вид.

1) $Q = \{q\}$, $q_0 = q$, $T = \{+, (,), a\}$, $N = \{+, (,), a, S, A\}$, $N_0 = S$, $Z = \emptyset$.

2) $F(q, \varepsilon, S) = (q, S+A)$, $F(q, \varepsilon, S) = (q, A)$, $F(q, \varepsilon, A) = (q, (S))$; $F(q, \varepsilon, A) = (q, a)$.

3) $F(q, t, t) = (q, \varepsilon)$ для каждого $t \in \{+, (,), a\}$.

Распознавание строки (a) построенным МП-автоматом представлено в таблице 5.1. Полученный МП-автомат является недетерминированным.

Таблица 5.1 – Распознавание МП-автоматом строки (a)

Номер конфигурации	Текущее состояние	Входная строка	Содержимое магазина
1	q	(a)	S
2	q	(a)	A
3	q	(a)	(S)
4	q	$a)$	$S)$
5	q	$a)$	$A)$
6	q	$a)$	$a)$
7	q	$)$	$)$
8	q	ε	ε

Алгоритм 5.3. Построение расширенного МП-автомата по КС-грамматике

Построим МП-автомат, выполняющий правосторонний разбор. Данный автомат имеет единственное текущее состояние и одно заключительное состояние, в котором стек пуст. Стек содержит левую часть текущей сентенции. Первоначально в стек помещается специальный магазинный символ, маркер пустого стека $\#$. На каждом шаге автомат по правилу грамматики замещает нетерминалом строку верхних символов стека или дописывает в вершину входной символ.

Вход: КС-грамматика $G = (V_T, V_N, P, S)$.

Выход: расширенный МП-автомат $M = (Q, T, N, F, q_0, N_0, Z)$ такой, что $L(M) = L(G)$.

Шаг 1. Положить $Q = \{q, r\}$, $q_0 = q$, $Z = \{r\}$, $N = V_T \cup V_N \cup \{\#\}$, $T = V_T$, $N_0 = \#$.

Шаг 2. Для каждого правила вида $(A \rightarrow \beta) \in P$, где $\beta \in V^*$, сформировать магазинную функцию вида $F(q, \varepsilon, \beta) = (q, A)$, предписывающую заменять правую часть правила в вершине стека нетерминалом из левой части, независимо от текущего символа входной строки.

Шаг 3. Для каждого терминала $t \in T$ сформировать магазинную функцию вида $F(q, t, \varepsilon) = (q, t)$, которая помещает символ входной строки в вершину стека, если там нет правой части правила, и перемещает читающую головку.

Шаг 4. Предусмотреть магазинную функцию для перевода автомата в заключительное состояние $F(q, \varepsilon, \#S) = (r, \varepsilon)$.

Пример 5.2. Для грамматики из примера 5.1 построить расширенный МП-автомат. Последовательность построения МП-автомата будет иметь вид.

1) $Q = \{q, r\}$, $q_0 = q$, $T = \{+, (,), a\}$, $N = \{+, (,), a, S, A\}$, $N_0 = \#$, $Z = r$.

2) $F(q, \varepsilon, S+A) = (q, S)$, $F(q, \varepsilon, A) = (q, S)$, $F(q, \varepsilon, (S)) = (q, A)$, $F(q, \varepsilon, a) = (q, A)$.

3) $F(q, t, \varepsilon) = (q, t)$ для каждого $t \in \{+, (,), a\}$.

4) $F(q, \varepsilon, \#S) = (r, \varepsilon)$.

Распознавание строки (a) расширенным МП-автоматом представлено в таблице 5.2. Полученный МП-автомат является детерминированным.

Таблица 5.2 – Распознавание расширенным МП-автоматом строки (a)

Номер конфигурации	Текущее состояние	Входная строка	Содержимое магазина
1	q	(a)	$\#$
2	q	$a)$	$\#($
3	q	$)$	$\#(a$
4	q	$)$	$\#(A$
5	q	$)$	$\#(S$
6	q	ε	$\#(S)$
7	q	ε	$\#A$
8	q	ε	$\#S$
9	r	ε	ε

Постановка задачи к лабораторной работе № 5

Разработать программное средство, реализующее следующие функции:

а) ввод произвольной формальной грамматики и проверка ее на принадлежность к классу КС-грамматик;

б) построение МП-автомата по КС-грамматике;

в) построение расширенного МП-автомата по КС-грамматике.

Продемонстрировать разбор некоторой входной строки с помощью построенных автоматов для случая:

а) входная строка принадлежит языку исходной КС-грамматики и допускается МП-автоматом;

б) входная строка не принадлежит языку исходной КС-грамматики и не принимается МП-автоматом.

Индивидуальные варианты заданий представлены в таблице 4.1.

6 Лабораторная работа № 6. Моделирование функционирования распознавателя для $LL(1)$ -грамматик

Цель: - закрепить понятие « $LL(k)$ –грамматика», необходимые и достаточные условия $LL(k)$ –грамматики;

- сформировать умения и навыки построения множеств $FIRST(k, \alpha)$ и $FOLLOW(k, \alpha)$, распознавателя для $LL(1)$ -грамматик.

Основы теории

Определение 6.1. КС-грамматика обладает свойством $LL(k)$ для некоторого $k>0$, если на каждом шаге вывода для однозначного выбора очередной альтернативы МП-автомату достаточно знать символ на вершине стека и рассмотреть первые k символов от текущего положения считывающей головки во входной строке.

Определение 6.2. КС-грамматика называется $LL(k)$ -грамматикой, если она обладает свойством $LL(k)$ для некоторого $k>0$.

В основе распознавателя $LL(k)$ -грамматик лежит левосторонний разбор строки языка. Исходной сентенциальной формой является начальный символ грамматики, а целевой – заданная строка языка. На каждом шаге разбора правило грамматики применяется к самому левому нетерминалу сентенции. Данный процесс соответствует построению дерева разбора цепочки сверху вниз (от корня к листьям). Отсюда и произошла аббревиатура $LL(k)$: первая « L » (от слова «*left*») означает левосторонний ввод исходной цепочки символов, вторая « L » - левосторонний вывод в процессе работы распознавателя.

Определение 6.3. Для построения распознавателей для $LL(k)$ -грамматик используются два множества:

- $FIRST(k, \alpha)$ – множество терминальных цепочек, выводимых из цепочки $\alpha \in (V_T \cup V_N)^*$, укороченных до k символов;

- $FOLLOW(k, A)$ – множество укороченных до k символов терминальных цепочек, которые могут следовать непосредственно за $A \in V_N$ в цепочках вывода.

Формально эти множества можно определить следующим образом:

- $FIRST(k, \alpha) = \{\omega \in V_T^* \mid \exists \text{ вывод } \alpha \Rightarrow^* \omega \text{ и } |\omega| \leq k \text{ или } \exists \text{ вывод } \alpha \Rightarrow^* \omega x \text{ и } |\omega| = k; x, \alpha \in (V_T \cup V_N)^*, k > 0\}$;

- $FOLLOW(k, A) = \{\omega \in V_T^* \mid \exists \text{ вывод } S \Rightarrow^* \alpha A \gamma \text{ и } \omega \in FIRST(k, \gamma); \alpha, \gamma \in V^*, A \in V_N, k > 0\}$.

Теорема 6.1. Необходимое и достаточное условие $LL(1)$ -грамматики

Для того чтобы грамматика $G(V_N, V_T, P, S)$ была $LL(1)$ -грамматикой необходимо и достаточно, чтобы для каждого символа $A \in V_N$, у которого в грамматике существует более одного правила вида $A \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$, выполнялось требование:

$$FIRST(1, \alpha_i FOLLOW(1, A)) \cap FIRST(1, \alpha_j FOLLOW(1, A)) = \emptyset,$$

$$\forall i \neq j, 0 < i \leq n, 0 < j \leq n.$$

Т.е. если для символа A отсутствует правило вида $A \rightarrow \varepsilon$, то все множества $FIRST(1, \alpha_1), FIRST(1, \alpha_2), \dots, FIRST(1, \alpha_n)$ должны попарно не пересекаться, если же присутствует правило $A \rightarrow \varepsilon$, то они не должны также пересекаться с множеством $FOLLOW(1, A)$.

Для построения распознавателей для $LL(1)$ -грамматик необходимо построить множества $FIRST(1, x)$ и $FOLLOW(1, A)$. Причем, если строка x будет начинаться с терминального символа a , то $FIRST(1, x) = a$, и если она будет начинаться с нетерминального символа A , то $FIRST(1, x) = FIRST(1, A)$. Следовательно, достаточно рассмотреть алгоритмы построения множеств $FIRST(1, A)$ и $FOLLOW(1, A)$ для каждого нетерминального символа A .

Алгоритм 6.1. Построение множества $FIRST(1, A)$

Для выполнения алгоритма необходимо предварительно преобразовать исходную грамматику G в грамматику G' , не содержащую ε -правил (см. лабораторную работу № 4). Алгоритм построения множества $FIRST(1, A)$ использует грамматику G' .

Шаг 1. Первоначально внести во множество первых символов для каждого нетерминального символа A все символы, стоящие в начале правых частей правил для этого нетерминала, т.е.

$$\forall A \in V_N \text{ } FIRST(1, A) = \{X \mid A \rightarrow X\alpha \in P, X \in (V_T \cup V_N), \alpha \in (V_T \cup V_N)^*\}.$$

Шаг 2. Для всех $A \in V_N$ положить:

$$FIRST_{i+1}(1, A) = FIRST_i(1, A) \cup FIRST_i(1, B), \forall B \in (FIRST(1, A) \cap V_N).$$

Шаг 3. Если существует $A \in V_N$, такой что $FIRST_{i+1}(1, A) \neq FIRST_i(1, A)$, то присвоить $i = i + 1$ и вернуться к шагу 2, иначе перейти к шагу 4.

Шаг 4. Исключить из построенных множеств все нетерминальные символы, т.е.

$$\forall A \in V_N \text{ } FIRST(1, A) = FIRST_i(1, A) \setminus V_N.$$

Алгоритм 6.2. Построение множества $FOLLOW(1, A)$

Алгоритм основан на использовании правил вывода грамматики G .

Шаг 1. Первоначально внести во множество последующих символов для каждого нетерминального символа A все символы, которые в правых частях правил вывода встречаются непосредственно за символом A , т.е.

$$\forall A \in V_N \text{ } FOLLOW_0(1, A) = \{X \mid \exists B \rightarrow \alpha A X \beta \in P, B \in V_N, X \in (V_T \cup V_N), \alpha, \beta \in (V_T \cup V_N)^*\}.$$

Шаг 2. Внести пустую строку во множество $FOLLOW(1, S)$, т.е.

$$FOLLOW(1, S) = FOLLOW_0(1, S) \cup \{\varepsilon\}.$$

Шаг 3. Для всех $A \in V_N$ вычислить:

$$FOLLOW'_i(1, A) = FOLLOW_i(1, A) \cup FIRST(1, B), \forall B \in (FOLLOW_i(1, A) \cap V_N).$$

Шаг 4. Для всех $A \in V_N$ положить:

$$FOLLOW''_i(1, A) = FOLLOW'_i(1, A) \cup FOLLOW'_i(1, B), \\ \forall B \in (FOLLOW'_i(1, A) \cap V_N), \text{ если } \exists \text{ правило } B \rightarrow \varepsilon.$$

Шаг 5. Для всех $A \in V_N$ определить:

$$FOLLOW_{i+1}(1, A) = FOLLOW''_i(1, A) \cup FOLLOW''_i(1, B), \\ \text{для всех нетерминальных символов } B \in V_N, \text{ имеющих правило вида } \\ B \rightarrow \alpha A, \alpha \in (V_T \cup V_N)^*.$$

Шаг 6. Если существует $A \in V_N$ такой, что $FOLLOW_{i+1}(1, A) \neq FOLLOW_i(1, A)$, то положить $i := i + 1$ и вернуться к шагу 3, иначе перейти к шагу 7.

Шаг 7. Исключить из построенных множеств все нетерминальные символы, т.е. $\forall A \in V_N FOLLOW(1, A) = FOLLOW_i(1, A) \setminus V_N$.

Алгоритм 6.3. Функционирование распознавателя цепочек для $LL(1)$ -грамматик

Шаг 1. Помещаем в стек начальный символ грамматики S , а во входной буфер исходную цепочку символов.

Шаг 2. До тех пор пока в стеке и во входном буфере останется только пустая строка ε либо будет обнаружена ошибка в алгоритме разбора, выполняем одно из следующих действий:

- если на верхушке стека находится нетерминальный символ A и очередной символ входной строки символ a , то выполняем операцию «свертка» по правилу $A \rightarrow x$ при условии, что $a \in FIRST(1, x)$, т.е. извлекаем из стека символ A и заносим в стек строку x , не меняя содержимого входного буфера;
- если на верхушке стека находится нетерминальный символ A и очередной символ входной строки символ a , то выполняем операцию «свертка» по правилу $A \rightarrow \varepsilon$ при условии, что $a \in FOLLOW(1, A)$, т.е. извлекаем из стека символ A и заносим в стек строку ε , не меняя содержимого входного буфера;
- если на верхушке стека находится терминальный символ a , совпадающий с очередным символом входной строки, то выполняем операцию «выброс», т.е. удаляем из стека и входного буфера данный терминальный символ;
- если содержимое стека и входного буфера пусто, то исходная строка прочитана полностью, и разбор завершен удачно;
- если ни одно из данных условий не выполнено, то цепочка не принадлежит заданному языку, и алгоритм завершает свою работу с ошибкой.

Пример 6.1. Дана грамматика $G (\{S, T, R\}, \{+, -, (,), a, b\}, P, S)$, с правилами P : 1) $S \rightarrow TR$; 2) $R \rightarrow \varepsilon \mid +TR \mid -TR$; 3) $T \rightarrow (S) \mid a \mid b$. Построить распознаватель для строки $(a+(b-a))$ языка грамматики G .

Этап 1. Преобразуем грамматику G в грамматику G' , не содержащую ε -правил:

$$N_0 = \{R\};$$

$N_1 = \{R\}$, т.к. $N_0 = N_1$, то во множество P' войдут правила:

$$1) S \rightarrow TR \mid T; \quad 2) R \rightarrow +TR \mid +T \mid -TR \mid -T; \quad 3) T \rightarrow (S) \mid a \mid b.$$

Этап 2. Построение множеств $FIRST(1, A)$ для каждого нетерминала A представлено в таблице 6.1.

Таблица 6.1 – Построение множеств $FIRST(1, A)$

$FIRST_i(1, A)$	0	1	2	$FIRST(1, A)$
S	T	$T, (, a, b$	$T, (, a, b$	$(, a, b$
R	$+, -$	$+, -$	$+, -$	$+, -$
T	$(, a, b$	$(, a, b$	$(, a, b$	$(, a, b$

Этап 3. Построение множеств $FOLLOW(1, A)$ для каждого нетерминала A представлено в таблице 6.2.

Таблица 6.2 – Построение множеств $FOLLOW(1, A)$

Шаг	Нетерминалы	$FOLLOW_i(1, A)$	$FOLLOW_i'(1, A)$	$FOLLOW_i''(1, A)$
0	S)), ε), ε
	R	\emptyset	\emptyset	\emptyset
	T	R	$R, +, -$	$R, +, -$
1	S), ε), ε), ε
	R), ε), ε), ε
	T	$R, +, -$	$R, +, -$	$R, +, -,), \varepsilon$
2	S), ε), ε), ε
	R), ε), ε), ε
	T	$R, +, -,), \varepsilon$	$R, +, -,), \varepsilon$	$R, +, -,), \varepsilon$
$FOLLOW(1, S)$), ε		
$FOLLOW(1, R)$), ε		
$FOLLOW(1, T)$		$+, -,), \varepsilon$		

Этап 4. Множества $FIRST(1, A)$ и $FOLLOW(1, A)$ для каждого нетерминала A сведены в таблицу 6.3.

Таблица 6.3 – Множества $FIRST(1, A)$ и $FOLLOW(1, A)$

A	$FIRST(1, A)$	$FOLLOW(1, A)$
S	$(, a, b$), ε
R	$+, -$), ε
T	$(, a, b$	$+, -,), \varepsilon$

Грамматика G является $LL(1)$ -грамматикой, т.к. для каждого нетерминала A , имеющего альтернативные выводы, множества $FIRST(1, A)$ попарно не пересекаются, а для нетерминала R они также не пересекаются со множеством $FOLLOW(1, R)$.

Шаг 5. Разбор строки $(a+(b-a))$ для грамматики G показан в таблице 6.4.

Таблица 6.4 - Разбор строки $(a+(b-a))$ для грамматики G

Стек	Входной буфер	Действие
S	$(a+(b-a))$	свертка $S \rightarrow TR$, т.к. $(\in FIRST(1, TR)$
TR	$(a+(b-a))$	свертка $T \rightarrow (S)$, т.к. $(\in FIRST(1, (S))$
$(S)R$	$(a+(b-a))$	выброс
$S)R$	$a+(b-a))$	свертка $S \rightarrow TR$, т.к. $a \in FIRST(1, TR)$
$TR)R$	$a+(b-a))$	свертка $T \rightarrow a$, т.к. $a \in FIRST(1, a)$
$aR)R$	$a+(b-a))$	выброс
$R)R$	$+(b-a))$	свертка $R \rightarrow +TR$, т.к. $+ \in FIRST(1, TR)$
$+TR)R$	$+(b-a))$	выброс
$TR)R$	$(b-a))$	свертка $T \rightarrow (S)$, т.к. $(\in FIRST(1, (S))$
$(S)R)R$	$(b-a))$	выброс
$S)R)R$	$b-a))$	свертка $S \rightarrow TR$, т.к. $b \in FIRST(1, TR)$
$TR)R)R$	$b-a))$	свертка $T \rightarrow b$, т.к. $b \in FIRST(1, b)$
$bR)R)R$	$b-a))$	выброс
$R)R)R$	$-a))$	свертка $R \rightarrow -TR$, т.к. $- \in FIRST(1, -TR)$
$-TR)R)R$	$-a))$	выброс
$TR)R)R$	$a))$	свертка $T \rightarrow a$, т.к. $a \in FIRST(1, a)$
$aR)R)R$	$a))$	выброс
$R)R)R$	$)$	свертка $R \rightarrow \varepsilon$, т.к. $) \in FOLLOW(1, R)$
$)R)R$	$)$	выброс
$R)R$	$)$	свертка $R \rightarrow \varepsilon$, т.к. $) \in FOLLOW(1, R)$
$)R$	$)$	выброс
R	ε	свертка $R \rightarrow \varepsilon$, т.к. $\varepsilon \in FOLLOW(1, R)$
ε	ε	строка принята полностью

Шаг 6. Получили следующую цепочку вывода:

$$\begin{aligned}
 S &\Rightarrow TR \Rightarrow (S)R \Rightarrow (TR)R \Rightarrow (aR)R \Rightarrow (a+TR)R \Rightarrow (a+(S)R)R \Rightarrow (a+(TR)R)R \Rightarrow \\
 &\Rightarrow (a+(bR)R)R \Rightarrow (a+(b-TR)R)R \Rightarrow (a+(b-aR)R)R \Rightarrow (a+(b-a)R)R \Rightarrow (a+(b-a))R \\
 &\Rightarrow (a+(b-a)).
 \end{aligned}$$

Нисходящее дерево разбора цепочки представлено на рисунке 6.1.

Постановка задачи к лабораторной работе № 6

Разработать программное средство, автоматизирующее процесс разбора цепочек для $LL(1)$ -грамматик. Программное средство должно выполнять следующие функции:

- 1) реализация ввода произвольной КС-грамматики;
- 2) построение множеств $FIRST(1, A)$ и $FOLLOW(1, A)$ для каждого нетерминального символа грамматики;
- 3) проверка необходимого и достаточного условия $LL(1)$ для введенной КС-грамматики;
- 4) моделирование функционирования распознавателя для $LL(1)$ -грамматик.

Составить набор контрольных примеров для случаев:

- а) введенная КС-грамматика не является $LL(1)$ -грамматикой;
- б) исходная КС-грамматика является $LL(1)$ -грамматикой, но входная строка не принадлежит языку грамматики;
- в) заданная КС-грамматика является $LL(1)$ -грамматикой и введенная строка принадлежит языку грамматики.

Разбор цепочек показать с помощью таблицы, строки вывода и дерева вывода. Вариантами индивидуальных заданий к лабораторной работе № 6 являются выходные данные лабораторной работы № 4.

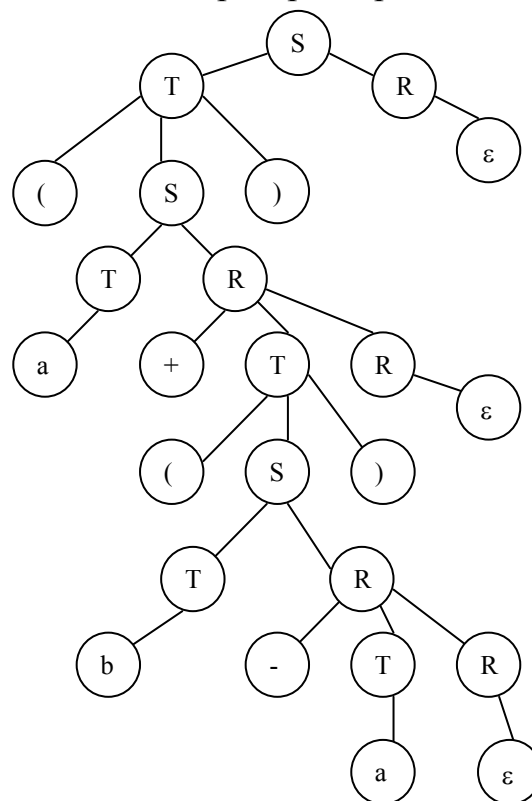


Рисунок 6.1 – Дерево вывода для цепочки $(a+(b-a))$ в грамматике G

7 Лабораторная работа № 7. Моделирование функционирования распознавателя для грамматик простого предшествования

Цель: - закрепить понятие «грамматика простого предшествования»; - сформировать умения и навыки построения множеств $L(A)$ и $R(A)$, матрицы предшествования символов грамматики и распознавателя для грамматик простого предшествования методом «сдвиг-свертка».

Основы теории

Определение 7.1. Приведенная КС-грамматика $G(V_N, V_T, P, S)$ называется грамматикой простого предшествования, если выполняются следующие условия.

1) Для каждой упорядоченной пары терминальных и нетерминальных символов выполняется не более чем одно из трех отношений предшествования:

а) $B_i = \cdot B_j$ ($\forall B_i, B_j \in V$), если и только если существует правило $A \rightarrow xB_iB_jy \in P$, где $x, y \in V^*$;

б) $B_i < \cdot B_j$ ($\forall B_i, B_j \in V$), если и только если существует правило $A \rightarrow xB_iDy \in P$ и вывод $D \Rightarrow^* B_jz$, где $A, D \in V_N, x, y, z \in V^*$;

в) $B_i \cdot > B_j$ ($\forall B_i, B_j \in V$), если и только если существует правило $A \rightarrow xCB_jy$ и вывод $C \Rightarrow^* zB_i$ или существует правило $A \rightarrow xCDy \in P$ и вывод $C \Rightarrow^* zB_i$ и $D \Rightarrow^* B_jw$, где $A, C, D \in V_N, x, y, z, w \in V^*$.

2) Различные правила в грамматике имеют разные правые части.

Определение 7.2. Отношения $=\cdot, <\cdot, \cdot>$ называют отношениями простого предшествования для символов грамматики.

В основе распознавателя для грамматик простого предшествования лежит правосторонний разбор строки языка. Исходной сентенциальной формой является заданная строка языка, а целевой – начальный символ грамматики. На каждом шаге разбора в исходной цепочке символов пытаются выделить подцепочку, совпадающую с правой частью некоторого правила вывода грамматики, и заменить ее нетерминалом, стоящим в левой части этого правила. Данная операция называется сверткой к нетерминалу, а заменяемая подстрока – основой сентенции. Описанный процесс разбора соответствует построению дерева вывода цепочки снизу вверх (от листьев к корню).

Метод предшествования основан на том факте, что отношения между двумя соседними символами распознаваемой строки соответствуют трем следующим вариантам:

- $B_i = \cdot B_{i+1}$, если символы B_i и B_{i+1} принадлежат основе;
- $B_i < \cdot B_{i+1}$, если B_{i+1} – крайний левый символ некоторой основы;
- $B_i \cdot > B_{i+1}$, если B_i – крайний правый символ некоторой основы.

Алгоритм 7.1. Поиск основы сентенции грамматики

Если грамматика является грамматикой простого предшествования, то для поиска основы каждой ее предложения надо просматривать элементы предложения слева направо и найти самую левую пару символов x_j и x_{j+1} , такую что $x_j \cdot > x_{j+1}$. Окончанием основы предложения будет x_j . Далее просматривать элементы предложения справа налево, начиная с символа x_j до тех пор, пока не будет найдена самая правая пара символов x_{i-1} и x_i , такая что $x_{i-1} < \cdot x_i$. Заголовком основы будет символ x_i . Таким образом, будет найдена основа предложения, имеющая вид $x_i x_{i+1} \dots x_{j-1} x_j$. Схема поиска основы предложения грамматики представлена на рисунке 7.1.

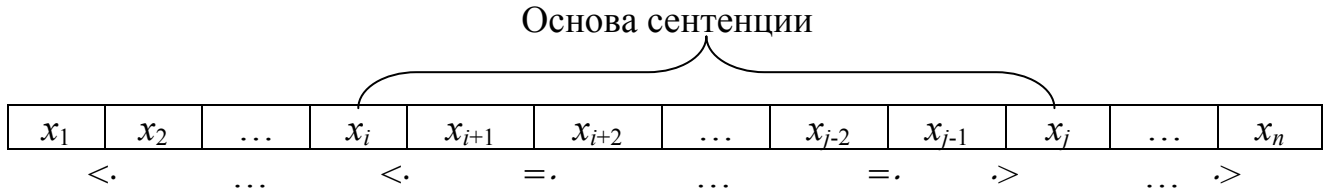


Рисунок 7.1 – Схема поиска основы предложения грамматики

На основе отношений предшествования строят матрицу предшествования грамматики. Строки и столбцы матрицы предшествования помечаются символами грамматики. Пустые клетки матрицы указывают на то, что данные символы не связаны отношением предшествования.

Определение 7.3. Построение матрицы предшествования основано на двух вспомогательных множествах, определяемых следующим образом:

- $L(A) = \{X \mid \exists A \Rightarrow^* Xz\}$, $A \in V_N$, $X \in V$, $z \in V^*$ - множество крайних левых символов относительно нетерминального символа A ;
- $R(A) = \{X \mid \exists A \Rightarrow^* zX\}$, $A \in V_N$, $X \in V$, $z \in V^*$ - множество крайних правых символов относительно нетерминального символа A .

Определение 7.4. Отношения предшествования можно определить с помощью введенных множеств следующим образом:

- $B_i = \cdot B_j$ ($\forall B_i, B_j \in V$), если и только если существует правило $A \rightarrow xB_iB_jy \in P$, где $A \in V_N$, $x, y \in V^*$;
- $B_i < \cdot B_j$ ($\forall B_i, B_j \in V$), если и только если существует правило $A \rightarrow xB_iDy \in P$ и $B_j \in L(D)$, где $A, D \in V_N$, $x, y \in V^*$;
- $B_i \cdot > B_j$ ($\forall B_i, B_j \in V$), если и только если существует правило $A \rightarrow xCB_jy$ и $B_i \in R(C)$ или существует правило $A \rightarrow xCDy \in P$ и $B_i \in R(C)$, $B_j \in L(D)$, где $A, C, D \in V_N$, $x, y \in V^*$.

Матрицу предшествования дополняют символами \perp_n и \perp_k (начало и конец цепочки). Для них определены следующие отношения предшествования:

- $\perp_n < \cdot X$, $\forall X \in V$, если $X \in L(S)$;
- $\perp_k \cdot > X$, $\forall X \in V$, если $X \in R(S)$.

Алгоритм 7.2. Построение множеств $L(A)$ и $R(A)$

Шаг 1. Для каждого нетерминального символа A ищем все правила, содержащие A в левой части. Во множество $L(A)$ включаем самый левый символ из правой части правил, а во множество $R(A)$ – самый крайний правый символ из правой части, т.е.

$$\forall A \in V_N: L_0(A) = \{X \mid A \rightarrow Xu, X \in V, u \in V^*\}, \\ R_0(A) = \{X \mid A \rightarrow uX, X \in V, u \in V^*\}.$$

Шаг 2. Для каждого нетерминального символа A : если множество $L(A)$ содержит нетерминальные символы грамматики A', A'', \dots , то множество $L(A)$ надо дополнить символами, входящими в соответствующие множества $L(A')$, $L(A'')$ и т.д., ... и не входящими в $L(A)$. Аналогичную операцию выполнить для множеств $R(A)$, т.е.

$$\forall A \in V_N: L_i(A) = L_{i-1}(A) \cup L_{i-1}(B), \forall B \in (L_{i-1}(A) \cap V_N), \\ R_i(A) = R_{i-1}(A) \cup R_{i-1}(B), \forall B \in (R_{i-1}(A) \cap V_N).$$

Шаг 3. Если на предыдущем шаге хотя бы одно множество $L(A)$ или $R(A)$ для некоторого символа грамматики изменилось, то вернуться к шагу 2, иначе построение закончено. Т.е. если существует $A \in V_N: R_i(A) \neq R_{i-1}(A)$ или $L_i(A) \neq L_{i-1}(A)$, то положить $i := i + 1$ и вернуться к шагу 2, иначе построение закончено и $R(A) = R_i(A)$ и $L(A) = L_i(A)$.

Алгоритм 7.3. Функционирование распознавателя для грамматик простого предшествования

Шаг 1. Поместить в вершущку стека символ \perp_n , считывающую головку – в начало входной цепочки символов.

Шаг 2. До тех пор, пока не будет обнаружена ошибка, либо успешно завершен алгоритм разбора, сравниваем отношение простого предшествования символа на вершущке стека и очередного символа входной строки. При этом возможны следующие ситуации:

- если самый верхний символ стека имеет меньшее или равное предшествование, чем очередной символ входной строки, то производим операцию «сдвиг» (перенос текущего символа из входной цепочки в стек и перемещение считывающей головки на один символ вправо);
- если самый верхний символ стека имеет большее предшествование, чем очередной символ входной строки, то выполняем операцию «свертка». Для этого находим на вершущке стека «основу» сентенции, т.е. все символы, имеющие равное предшествование или один символ на вершущке стека. Символы основы удаляем из стека, выбираем правило вывода грамматики, имеющее правую часть, совпадающую с основой, и помещаем в стек левую часть выбранного правила. Если такого правила вывода найти не удалось, то выдается сообщение об ошибке, и разбор завершен неудачно;

- если не установлено ни одно отношение предшествования между текущим символом входной цепочки и самым верхним символом в стеке, то алгоритм прерывается сообщением об ошибке;
- если в стеке остаются символы $\perp_n S$, а во входном буфере только символ \perp_k , то входная строка прочитана полностью, и алгоритм разбора завершен успешно.

Пример 7.1. Дана грамматика $G(\{a, (,)\}, \{S, R\}, P, S)$, с правилами P :

1) $S \rightarrow (R \mid a$; 2) $R \rightarrow Sa$). Построить распознаватель для строки $((aa)a)a\perp_k$.

Этап 1. Построим множества крайних левых и крайних правых символов $L(A)$ и $R(A)$ относительно всех нетерминальных символов грамматики (таблица 7.1).

Таблица 7.1 – Построение множеств $L(A)$ и $R(A)$ для грамматики G

Шаг	$L_i(A)$	$R_i(A)$
0	$L_0(S) = \{ (, a \}$ $L_0(R) = \{ S \}$	$R_0(S) = \{ R, a \}$ $R_0(R) = \{ \}$
1	$L_1(S) = \{ (, a \}$ $L_1(R) = \{ S, (, a \}$	$R_1(S) = \{ R, a,) \}$ $R_1(R) = \{ \}$
2	$L_2(S) = \{ (, a \}$ $L_2(R) = \{ S, (, a \}$	$R_2(S) = \{ R, a,) \}$ $R_2(R) = \{ \}$
Результат	$L(S) = \{ (, a \}$ $L(R) = \{ S, (, a \}$	$R(S) = \{ R, a,) \}$ $R(R) = \{ \}$

Этап 2. На основе построенных множеств и правил вывода грамматики составим матрицу предшествования символов (таблица 7.2).

Поясним заполнение матрицы предшествования. В правиле грамматики $S \rightarrow (R$ символ $($ стоит слева от нетерминального символа R . Во множестве $L(R)$ входят символы $S, (, a$. Ставим знак $<\cdot$ в клетках матрицы, соответствующих этим символам, в строке для символа $($.

В правиле грамматики $R \rightarrow Sa$ символ a стоит справа от нетерминального символа S . Во множество $R(S)$ входят символы $R, a,)$. Ставим знак $\cdot >$ в клетках матрицы, соответствующих этим символам, в столбце для символа a .

В строке символа \perp_n ставим знак $<\cdot$ в клетках символов, входящих во множество $L(S)$, т.е. символов $(, a$. В столбце символа \perp_k ставим знак $\cdot >$ в клетках, входящих во множество $R(S)$, т.е. символов $R, a,)$.

В клетках, соответствующих строке символа S и столбцу символа a , ставим знак $=$; т.к. существует правило $R \rightarrow Sa$, в котором эти символы стоят рядом. По тем же соображениям ставим знак $=\cdot$ в клетках строки a и столбца $)$, а также строки $($ и столбца R .

Таблица 7.2 – Матрица предшествования символов грамматики

Символы	S	R	a	$($	$)$	\perp_k
S			$=.$			
R			$.>$			$.>$
a			$.>$		$=.$	$.>$
$($	$<.$	$=.$	$<.$	$<.$		
$)$			$.>$			$.>$
\perp_H			$<.$	$<.$		

Шаг 3. Функционирование распознавателя для цепочки $((a)a)a$ показано в таблице 7.3.

Таблица 7.3 – Алгоритм работы распознавателя цепочки $((a)a)a$

Шаг	Стек	Входной буфер	Действие
1	\perp_H	$((a)a)a\perp_k$	сдвиг
2	$\perp_H($	$((a)a)a\perp_k$	сдвиг
3	$\perp_H(($	$(a)a)a\perp_k$	сдвиг
4	$\perp_H((($	$aa)a)a\perp_k$	сдвиг
5	$\perp_H(((a$	$a)a)a\perp_k$	свертка $S \rightarrow a$
6	$\perp_H(((S$	$a)a)a\perp_k$	сдвиг
7	$\perp_H(((Sa$	$)a)a\perp_k$	сдвиг
8	$\perp_H(((Sa)$	$a)a\perp_k$	свертка $R \rightarrow Sa$
9	$\perp_H(((R$	$a)a\perp_k$	свертка $S \rightarrow (R$
10	$\perp_H((S$	$a)a\perp_k$	сдвиг
11	$\perp_H((Sa$	$)a\perp_k$	сдвиг
12	$\perp_H((Sa)$	$a\perp_k$	свертка $R \rightarrow Sa$
13	$\perp_H((R$	$a\perp_k$	свертка $S \rightarrow (R$
14	$\perp_H(S$	$a\perp_k$	сдвиг
15	$\perp_H(Sa$	$)\perp_k$	сдвиг
16	$\perp_H(Sa)$	\perp_k	свертка $R \rightarrow Sa$
17	$\perp_H(R$	\perp_k	свертка $S \rightarrow (R$
18	\perp_HS	\perp_k	строка принята

Шаг 4. Получили следующую цепочку вывода:

$$S \Rightarrow (R \Rightarrow (Sa) \Rightarrow ((Ra) \Rightarrow ((Sa)a) \Rightarrow (((Ra)a) \Rightarrow (((Sa)a)a) \Rightarrow (((aa)a)a).$$

Восходящее дерево вывода цепочки представлено на рисунке 7.2.

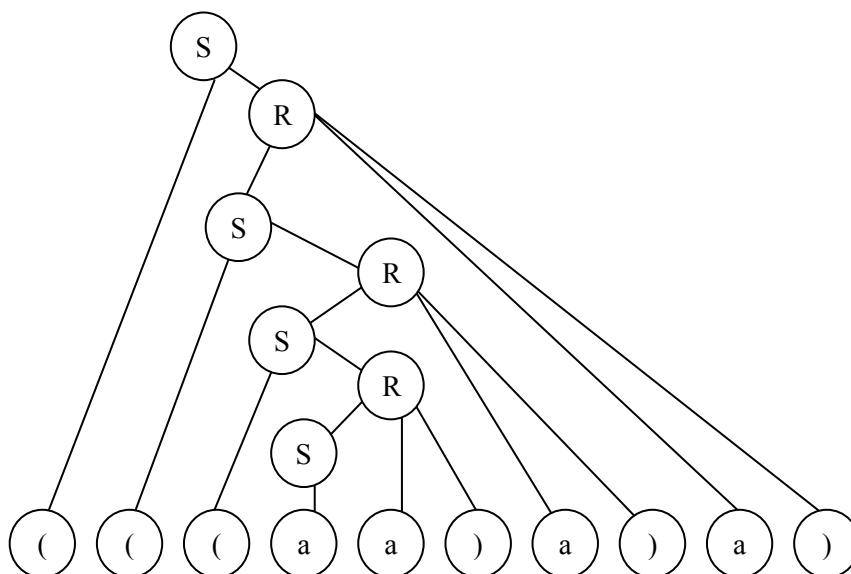


Рисунок 7.2 – Дерево вывода для цепочки $((aa)a)a$ в грамматике G

Постановка задачи к лабораторной работе № 7

Разработать программное средство, автоматизирующее процесс разбора цепочек для грамматик простого предшествования. Программное средство должно выполнять следующие функции:

- 1) ввод произвольной грамматики;
- 2) построение множеств $L(A)$ и $R(A)$ для каждого нетерминального символа грамматики;
- 3) формирование матрицы простого предшествования для введенной грамматики;
- 4) проверка условия простого предшествования для данной грамматики;
- 5) моделирование функционирования распознавателя для грамматик простого предшествования.

Составить набор контрольных примеров для случаев:

- а) введенная грамматика не является грамматикой простого предшествования;
- б) исходная грамматика является грамматикой простого предшествования, но анализируемая строка не принадлежит языку грамматики;
- в) заданная грамматика является грамматикой простого предшествования и входная строка принадлежит языку грамматики.

Разбор цепочек представить в виде таблицы, строки вывода и дерева вывода.

Вариантами индивидуального задания к лабораторной работе № 7 являются выходные данные лабораторной работы № 4.

Список использованных источников

- 1 Афанасьев А.Н. Формальные языки и грамматики: Учебное пособие. – Ульяновск: УлГТУ, 1997. – 84с.
- 2 Ахо А., Сети Р., Ульман Д. Компиляторы: принципы, технологии и инструменты.: Пер. с англ. – М.: Изд. дом «Вильямс», 2001. – 768с.
- 3 Братчиков И.Л. Синтаксис языков программирования / Под ред. С.С. Лаврова. – М.: Наука, 1975. - 262с.
- 4 Вайнгартен Ф. Трансляция языков программирования / Под ред. Мартынюка В.В.- М.: Мир, 1977. - 192с.
- 5 Вильямс А. Системное программирование в Windows 2000 для профессионалов. – СПб.: Питер, 2001. – 624с.
- 6 Волкова И.А., Руденко Т.В. Формальные языки и грамматики. Элементы теории трансляции. – М.: Диалог-МГУ, 1999. – 62с.
- 7 Гордеев А.В., Молчанов А.Ю. Системное программное обеспечение. – СПб: Питер, 2001. – 736с.
- 8 Грис Д. Конструирование компиляторов для цифровых вычислительных машин: Пер. с англ. – М.: Мир, 1975. – 544с.
- 9 Дворянкин А.И. Основы трансляции: Учебное пособие. – Волгоград: ВолгГТУ, 1999. – 80с.
- 10 Жаков В.И., Коровинский В.В., Фильчаков В.В. Синтаксический анализ и генерация кода. – СПб.: ГААП, 1993. – 26с.
- 11 Компаниец Р.И., Маньков Е.В., Филатов Н.Е. Системное программирование. Основы построения трансляторов. – СПб.: Корона принт, 2000. – 256с.
- 12 Льюис Ф., Розенкранц Д., Стирнз Р. Теоретические основы проектирования компиляторов. - М.: Мир, 1979. - 654с.
- 13 Пантелеева И.А. Методы трансляции: Конспект лекций. – Новосибирск: Изд-во НГТУ, 1998. – Ч.2. – 51с.
- 14 Пратт Т., Зелковиц М. Языки программирования: разработка и реализация / Под ред. А. Матросова. – СПб: Питер, 2002. – 688с.
- 15 Рейуорд-Смит В. Теория формальных языков. Вводный курс: Пер. с англ. – М.: Радио и связь, 1988. – 128с.
- 16 Саломеа А. Жемчужины теории формальных языков. - М.: Мир, 1986. – 160с.
- 17 Серебряков В.И. Лекции по конструированию компиляторов. – М.: МГУ, 1997. – 171с.
- 18 Соколов А.П. Системы программирования: теория, методы, алгоритмы: Учеб. пособие. – М.: Финансы и статистика, 2004. – 320с.
- 19 Федоров В.В. Основы построения трансляторов: Учебное пособие. – Обнинск: ИАТЭ, 1995. – 105с.
- 20 Хантер Р. Проектирование и конструирование компиляторов: Пер. с англ. – М.: Финансы и статистика, 1984. – 232с.

Приложение А
(обязательное)
Пример оформления отчета лабораторной работы

Министерство образования и науки Российской Федерации

Федеральное агентство образования

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
“ОРЕНБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ”

Факультет информационных технологий

Кафедра программного обеспечения вычислительной техники
и автоматизированных систем

ЛАБОРАТОРНАЯ РАБОТА №2

по дисциплине “Теория языков программирования и методов
трансляции”

“Построение конечного автомата по регулярной грамматике”

ГОУ ОГУ 220400.9004.18 О

Руководитель работы

_____ Ишакова Е.Н.
" ____ " _____ 2004г.

Исполнитель

студент гр. 01ПО1

_____ Ковальчук С.В.
" ____ " _____ 2004г.

Оренбург 2004

Содержание

1 Тема и цель лабораторной работы	3
2 Постановка задачи	4
3 Формальная модель задачи	5
4 Структурная организация данных	7
5 Спецификация основных процедур и функций	9
6 Алгоритм решения задачи	11
7 Анализ полученных результатов	12
Список использованных источников	13
Приложение А Контрольный пример	14
Приложение Б Текст программы	17

1 Тема и цель лабораторной работы

Лабораторная работа № 2.

Тема: «Построение конечного автомата по регулярной грамматике»

Цель: - закрепить понятия «регулярная грамматика», «недетерминированный и детерминированный конечный автомат»;

- сформировать умения и навыки построения конечного автомата по регулярной грамматике и преобразования недетерминированного конечного автомата к детерминированному конечному автомату.

2 Постановка задачи

Дана регулярная грамматика $G = (\{A, B, C, S\}, \{a, b\}, P, S)$, где

- P :
- 1) $S \rightarrow aB \mid aA$;
 - 2) $A \rightarrow aA \mid bC \mid b$;
 - 3) $B \rightarrow bB \mid aC \mid a$.

Разработать программное средство, реализующее следующие функции:

- 1) ввод произвольной формальной грамматики с клавиатуры и проверка ее на принадлежность к классу регулярных грамматик;
- 2) построение по заданной регулярной грамматике конечного автомата;
- 3) преобразование недетерминированного конечного автомата к детерминированному конечному автомату;
- 4) вывод графа результирующего конечного автомата на экран.

3 Формальная модель задачи

Определение 1. Детерминированным конечным автоматом (ДКА) называется пятерка объектов:

$$M = (Q, T, F, H, Z), \quad (2.1)$$

где Q - конечное множество состояний автомата;

T - конечное множество допустимых входных символов;

F - функция переходов, отображающая множество $Q \times T$ во множество Q ;

H - конечное множество начальных состояний автомата;

Z - множество заключительных состояний автомата, $Z \subseteq Q$.

Определение 2. Недетерминированным конечным автоматом (НКА) называется конечный автомат, в котором в качестве функции переходов используется отображение $Q \times T$ во множество всех подмножеств множества состояний автомата $P(Q)$, т.е. функция переходов неоднозначна, так как текущей паре (q, t) соответствует множество очередных состояний автомата $q' \in P(Q)$.

Способы представления функции переходов

Командный способ. Каждую команду КА записывают в форме $F(q, t) = p$, где $q, p \in Q, t \in T$.

Табличный способ. Строки таблицы переходов соответствуют входным символам автомата $t \in T$, а столбцы – состояниям Q . Ячейки таблицы заполняются новыми состояниями, соответствующими значению функции $F(q, t)$. Неопределенным значениям функции переходов соответствуют пустые ячейки таблицы.

Графический способ. Строится диаграмма состояний автомата – неупорядоченный ориентированный помеченный граф. Вершины графа помечены именами состояний автомата. Дуга ведет из состояния q в состояние p и помечается списком всех символов $t \in T$, для которых $F(q, t) = p$. Вершина, соответствующая входному состоянию автомата, снабжается стрелкой. Заключительное состояние на графе обозначается двумя концентрическими окружностями.

4 Структурная организация данных

Основная часть данных хранится и обрабатывается в двух классах *Grammar* (грамматика) и *FAutomat* (конечный автомат), представленных в таблице 4.1.

Таблица 4.1 – Описание полей классов

Поле	Тип	Назначение
Класс <i>Grammar</i>		
<i>N</i>	<i>charset</i>	множество нетерминалов грамматики
<i>T</i>	<i>charset</i>	множество терминалов грамматики
<i>P</i>	<i>rulemap</i>	множество правил грамматики
<i>S</i>	<i>char</i>	начальный символ грамматики
Класс <i>FAutomat</i>		
<i>*G</i>	<i>Grammar</i>	связанная с данным автоматом грамматика
<i>Q</i>	<i>charset</i>	множество состояний автомата
<i>T</i>	<i>charset</i>	множество входных символов автомата
<i>F</i>	<i>ftable</i>	таблица правил автомата
<i>H</i>	<i>char</i>	начальное состояние автомата
<i>Z</i>	<i>charset</i>	множество конечных состояний автомата
<i>MustPaint</i>	<i>int</i>	флаг отрисовки графа

Для хранения таблицы правил конечного автомата используется структура:

```
struct posit{
    long x, y;
    long double a;
};
```

Для создания отображения используется класс компаратора:

```
struct rulecmp{
    bool operator()(const char c1, const char c2){
        return (c1 < c2);
    }
};
```

5 Спецификации основных процедур и функций программного средства

Основные функции программного средства описаны в виде методов классов *Grammar* и *FAutomat* в таблице 5.1.

Таблица 5.1 – Спецификации основных функций

Название	Входные параметры	Выходные параметры	Назначение
Методы класса Grammar			
int IsRegular()	Нет	Возвращает 1, если грамматика регулярная и 0 в противном случае	Проверка грамматики на принадлежность к классу регулярных грамматик
void InGrammar(char *fname)	Имя файла	Нет	Ввод грамматики из текстового файла
string AsString()	Строка, содержащая грамматику	Нет	Возвращает грамматику в виде строки
void OutGrammar(char *fname)	Имя файла	Нет	Вывод грамматики в текстовый файл
Методы класса FAutomat			
void SetGrammar(Grammar *NG)	Указатель на связанную грамматику	Нет	Связывает грамматику с данным конечным автоматом

Лист

6 Алгоритм решения задачи

Укрупненная схема алгоритма программного средства представлена на рисунке 6.1.

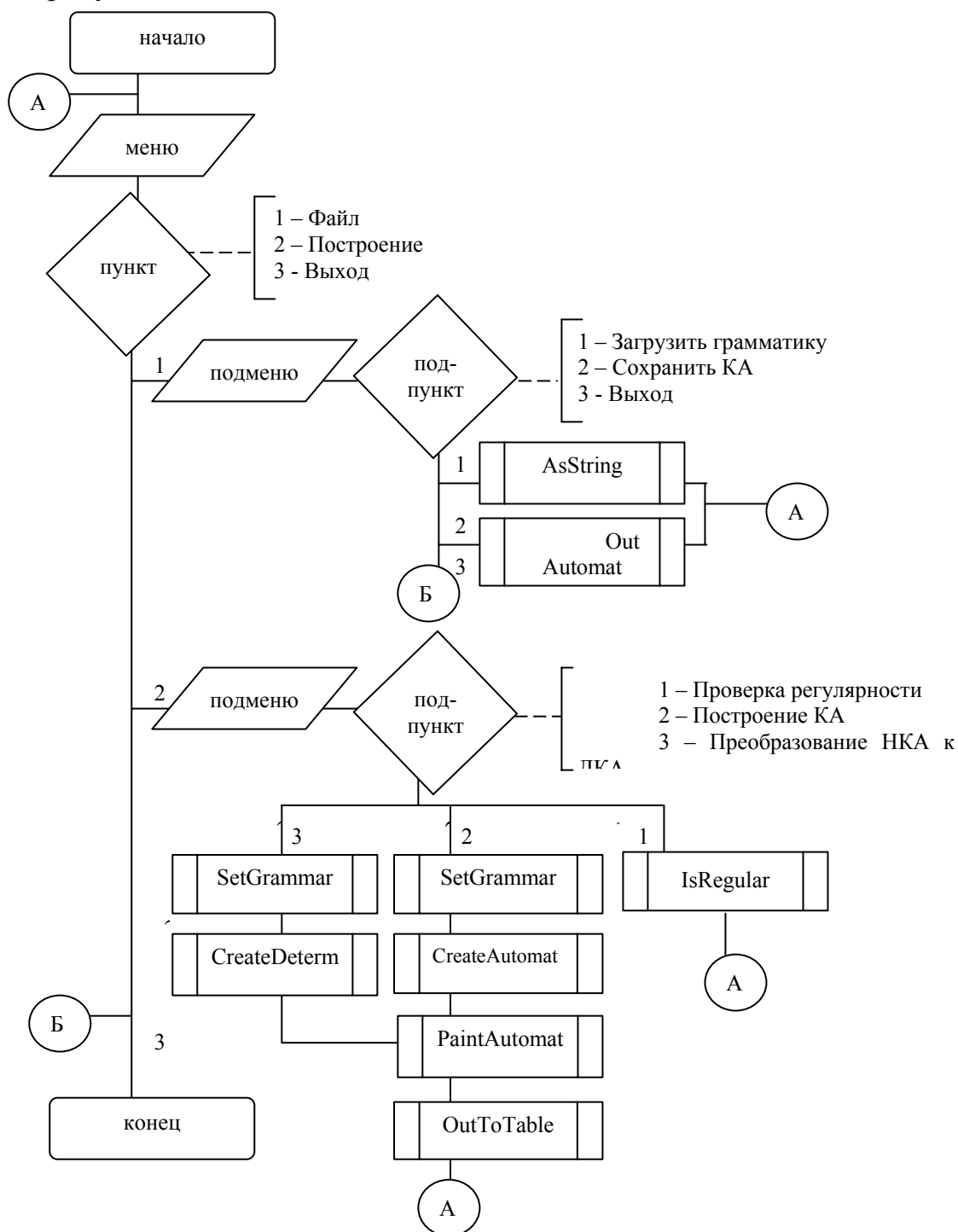


Рисунок 6.1 – Укрупненная схема алгоритма программного средства

Приложение Б

(обязательное)

Пример оформления приложений отчета лабораторной работы

Приложение А

(обязательное)

Контрольный пример

Исходная регулярная грамматика и соответствующий ей недетерминированный конечный автомат представлены на рисунке А.1.

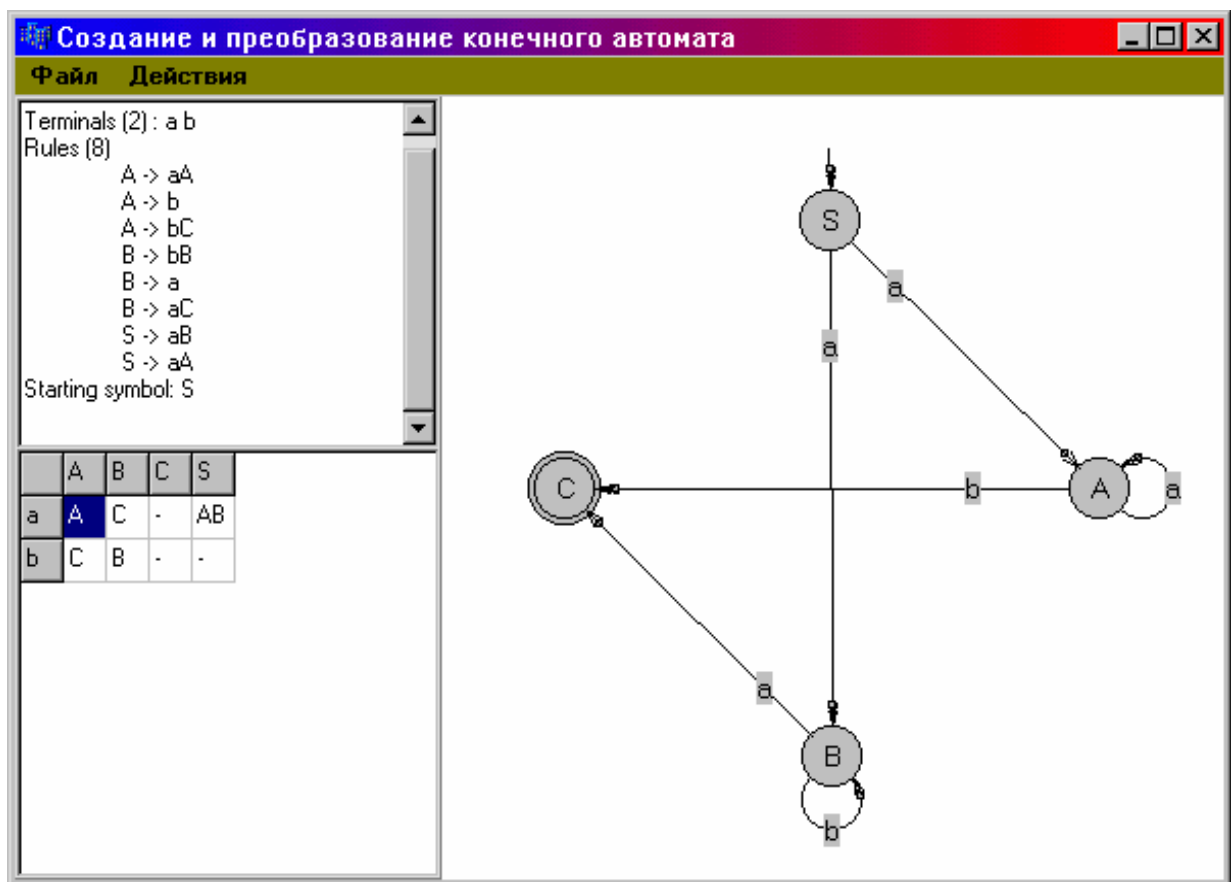


Рисунок А.1 – Исходная КС-грамматика и НКА

Результирующий детерминированный конечный автомат, построенный по заданному недетерминированному конечному автомату, представлен на рисунке А.2.

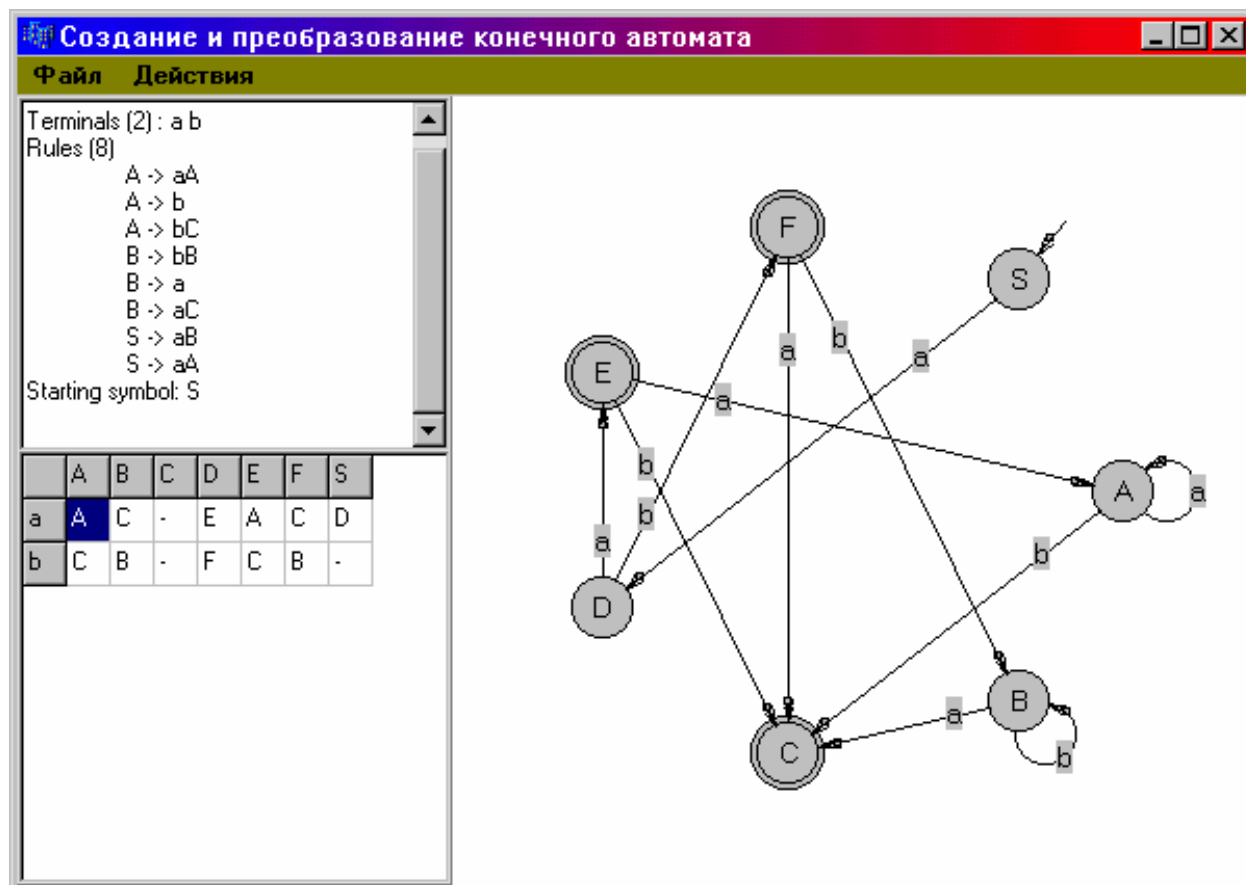


Рисунок А.2 – Результирующий ДКА

Приложение Б

(обязательное)

Текст программы

Unit2.h

```
#include <fstream.h>
#include <math.h>
#include <sysutils.hpp>
#include <graphics.hpp>
#include <grids.hpp>
#include <map>
#include <set>
#include <string>
#include <list>
#include <vector>

using std::string;
using std::map;
using std::multimap;
using std::set;
using std::pair;

// компаратор для работы в отображениях
struct rulecmp {
    bool operator()(const char c1, const char c2){
        return (c1 < c2);
    }
};

typedef string rule_r;
typedef char rule_l;
typedef pair<rule_l, rule_r> rule;
typedef multimap<rule_l, rule_r, rulecmp> rulemap;
typedef set<char> charset;

// класс грамматики
class Grammar{
public:
    charset N; // мн-во нетерминалов
    charset T; // множество терминалов
    rulemap P; // множество правил
    char S; // начальный символ

    int IsRegular(); // проверка грамматики на регулярность
    void InGrammar(char *fname); // ввод грамматики из файла
    string AsString(); // вывод грамматики в строку
    void OutGrammar(char *fname); // вывод грамматики в файл
};

typedef map<char, map<char, charset> > ftable;

// класс конечного автомата
class FAutomat{
public:
    Grammar *G; // связанная грамматика
```

```
    charset Q; // множество состояний
    charset T; // множество входных символов
    ftable F; // таблица правил
    char H; // начальное состояние
    charset Z; // множество конечных состояний
    int MustPaint; // флаг отрисовки графа
    FAutomat(){
        MustPaint = 0;
    }
    void SetGrammar(Grammar *NG); // связывание с грамматикой
    void CreateAutomat(); // создание автомата из грамматики
    void PaintAutomat(TCanvas * Canvas, long w, long h); // отрисовка графа
    void OutToTable(TStringGrid * Grid); // вывод таблицы правил в TStringGrid
    void CreateDeterm(); // преобразование в ДКА
};
```

Unit2.cpp

```
#include "Unit2.h"

int Grammar::IsRegular(){
    if (N.empty() || P.empty())
        return 0;
    for(rulemap::iterator i = P.begin(); i != P.end(); i++){
        if (!N.count(i->first))
            return 0;
        //по длине правой части
        switch(i->second.length()){
            case 1:{
                if (!T.count(i->second[0]))
                    return 0;
                break;
            }
            case 2:{
                if (!T.count(i->second[0]) || !N.count(i->second[1]))
                    return 0;
                break;
            }
            default:
                return 0;
        }
    }
    return 1;
}

void Grammar::InGrammar(char *fname){
    long n, i;
    rule r;
    char c;
```

```

ifstream fi(fname);
N.clear();
T.clear();
P.clear();
// ввод нетерминалов
fi >> n;
for (i = 0; i < n; i++){
    fi >> c;
    N.insert(c);
}
// ввод терминалов
fi >> n;
for (i = 0; i < n; i++){
    fi >> c;
    T.insert(c);
}
// ввод правил
fi >> n;
for (i = 0; i < n; i++){
    fi >> r.first >> r.second;
    P.insert(r);
}
// начальный символ
fi >> S;
}

string Grammar::AsString(){
    string res = "";
    charset::iterator i;
    res += "Nonterminals (";
    res += IntToStr(N.size()).c_str();
    res += ") : ";
    for (i = N.begin(); i != N.end(); i++){
        res += *i;
        res += " ";
    }
    res += "\nTerminals (";
    res += IntToStr(T.size()).c_str();
    res += ") : ";
    for (i = T.begin(); i != T.end(); i++){
        res += *i;
        res += " ";
    }
    res += "\nRules (";
    res += IntToStr(P.size()).c_str();
    res += ")\n";
    for (rulemap::iterator j = P.begin(); j != P.end();
j++){
        res += "\t";
        res += j->first;
        res += " -> " + j->second + "\n";
    }

    res += "Starting symbol: ";

```

```

        res += S;
        res += "\n";
        return res;
    }

void Grammar::OutGrammar(char *fname){
    ofstream fo(fname);
    charset::iterator i;
    fo << N.size() << "\n";
    for (i = N.begin(); i != N.end(); i++){
        fo << *i;
        fo << "\n" << T.size() << "\n";
        for (i = T.begin(); i != T.end(); i++){
            fo << *i;
            fo << "\n" << P.size() << "\n";
            for (rulemap::iterator j = P.begin(); j != P.end();
j++){
                fo << j->first << " " << j->second << "\n";
            }
            fo << "\n" << S;
        }
    }

void FAutomat::SetGrammar(Grammar *NG){
    G = NG;
}

void FAutomat::CreateAutomat(){
    rulemap::iterator i, j;
    rule r;
    char c, t;
    int k;
    // поиск незанятого символа
    for(c = 'A'; G->N.count(c); c++);
    // поиск правил вида A -> a без A -> aB
    for(i = G->P.begin(); i != G->P.end(); i++){
        if (i->second.length() == 1 && G->T.count(i-
>second[0])){
            for(j = G->P.lower_bound(i->first), k = G-
>P.count(i->first); k; j++, k--){
                if (j->second.length() == 2 && j-
>second[0] == i->second[0] && G->N.count(j-
>second[1]))
                    break;
            }
            if (!k){
                // добавление правила вида A -> aC
                r.first = i->first;
                r.second = i->second + c;
                G->P.insert(r);
                G->N.insert(c);
            }
        }
    }
    // начальный символ
    H = G->S;
    // состояния
    Q = G->N;

```