

Документ подписан простой электронной подписью

Информация о владельце:

ФИО: Шебзуова Тархан Александровна

Должность: Директор Пятигорского института (филиал) Северо-Кавказского

федерального университета

Дата подписания: 24.04.2024 10:38:38

Уникальный программный ключ:

d74ce93cd40e39275c3ba2f58486412a1c8ef96f

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**  
**Федеральное государственное автономное образовательное учреждение**

**высшего образования**

**«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

**Пятигорский институт (филиал) СКФУ**

**МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ ПО  
ДИСЦИПЛИНЕ  
ТЕОРИЯ ФОРМАЛЬНЫХ ЯЗЫКОВ, ГРАММАТИК И ПОСТРОЕНИЕ  
ТРАНСЛЯТОРОВ**

Направление подготовки

**09.04.02**

Направленность (профиль)

**Информационные системы и технологии**

**«Технологии работы с данными и**

**знаниями, анализ информации»**

Квалификация выпускника

**Магистр**

Пятигорск 2024 г.

## СОДЕРЖАНИЕ

Введение.....	3
1. Наименование лабораторных работ.....	4
Лабораторная работа № 1. Распознавание типов формальных языков и грамматик.....	5
Лабораторная работа № 2. Построение грамматики модельного языка (М-языка).	
Построение конечного автомата по регулярной грамматике.....	8
Лабораторная работа № 3. Построение лексического анализатора (сканера) для М-языка.	
Минимизация конечных автоматов.....	12
Лабораторная работа № 4. Построение синтаксического и семантического анализатора для М-языка. Эквивалентные преобразования контекстно-свободных грамматик.....	17
Лабораторная работа № 5. Генерация внутреннего представления программ. ПОЛИЗ.	
Построение автомата с магазинной памятью по контекстно-свободной грамматике.....	24
Лабораторная работа № 6. Построение объектного кода модуля М-языка. Ассемблеры.	
Моделирование функционирования распознавателя для LL(1)-грамматик.....	28
Лабораторная работа № 7. Автоматизированные методы построения компиляторов.	
Программы LEX и YACC. Моделирование функционирования распознавателя для грамматик простого предшествования.....	33
2. Учебно-методическое и информационное обеспечение дисциплины.....	39
Приложение 1.....	40

## ВВЕДЕНИЕ

Целью дисциплины «Теория формальных языков, грамматик и построение трансляторов» является ознакомление с основными понятиями и методами использования теории формальных грамматик, овладение теоретическими знаниями о фазы грамматического разбора при компиляции и интерпретации формальных текстов и практическими навыками по алгоритмам лексического, грамматического и семантического анализа.

Задачами дисциплины «Теория формальных языков, грамматик и построение трансляторов» являются:

- освоение принципов построения формальных языков программирования, работы компиляторов;
- разработка алгоритма с использованием грамматических структур формальных грамматик;
- знакомство с работой лексического анализатора языка программирования;
- моделирование лексического анализатора математического выражения.

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. В описание лабораторных работ включены цель работы, порядок ее выполнения, рассмотрены теоретические вопросы, связанные с реализацией поставленных задач, приведена необходимая литература.

### Оборудование и материалы

Аппаратные: персональный компьютер;

Программные: ОС Windows; Visual Studio, Microsoft Office.

Учебный класс оснащен IBM PC-совместимыми компьютерами класса, объединенными в локальную сеть. Локальная сеть учебного класса имеет постоянный доступ к сети Internet по выделенной линии. Для проведения лабораторных работ необходимо следующее программное обеспечение: операционная система MS Windows, пакет офисных программ MS Office.

### Указания по технике безопасности

Перед началом работы следует убедиться в исправности электропроводки, выключателей, штепсельных розеток, при помощи которых оборудование включается в сеть, наличии заземления компьютера, его работоспособности.

Для снижения или предотвращения влияния опасных и вредных факторов необходимо соблюдать санитарные правила и нормы, гигиенические требования к персональным электронно-вычислительным машинам.

Во избежание повреждения изоляции проводов и возникновения коротких замыканий не разрешается: вешать что-либо на провода, закрашивать и белить шнуры и провода, закладывать провода и шнуры за газовые и водопроводные трубы, за батареи отопительной системы, выдергивать штепсельную вилку из розетки за шнур, усилие должно быть приложено к корпусу вилки.

Для исключения поражения электрическим током запрещается: часто включать и выключать компьютер без необходимости, прикасаться к экрану и к тыльной стороне блоков компьютера, работать на средствах вычислительной техники и периферийном оборудовании мокрыми руками, работать на средствах вычислительной техники и периферийном оборудовании, имеющих нарушения целостности корпуса, нарушения изоляции проводов, неисправную индикацию включения питания, с признаками электрического напряжения на корпусе, класть на средства вычислительной техники и периферийном оборудовании посторонние предметы.

Запрещается под напряжением очищать от пыли и загрязнения электрооборудование.

Во избежание поражения электрическим током, при пользовании электроприборами нельзя касаться одновременно каких-либо трубопроводов, батарей отопления, металлических конструкций, соединенных с землей.

После окончания работы необходимо обесточить все средства вычислительной техники и периферийное оборудование. В случае непрерывного учебного процесса необходимо оставить включенными только необходимое оборудование.

## 1. НАИМЕНОВАНИЕ ЛАБОРАТОРНЫХ РАБОТ

### **Тема 1. Формальные языки и грамматики.**

Лабораторная работа № 1. Распознавание типов формальных языков и грамматик

### **Тема 2. Регулярные множества и регулярные выражения (РВ), конечные автоматы и автоматы с магазинной памятью. КС-грамматики и алгоритмы разбора.**

Лабораторная работа № 2. Построение грамматики модельного языка (М-языка).

Лабораторная работа № 3. Построение лексического анализатора (сканера) для М-языка.

Лабораторная работа № 4. Построение синтаксического и семантического анализатора для М-языка. Эквивалентные преобразования контекстно-свободных грамматик

Лабораторная работа № 5. Генерация внутреннего представления программ. ПОЛИЗ.

Построение автомата с магазинной памятью по контекстно-свободной грамматике.

### **Тема 3. Элементы теории трансляции. Операционные системы и их роль в процессе трансляции.**

Лабораторная работа № 6. Построение объектного кода модуля М-языка. Ассемблеры.

Моделирование функционирования распознавателя для LL(1)-грамматик

Компьютерные симуляции

### **Тема 4. Автоматизированные методы построения компиляторов.**

Лабораторная работа № 7. Автоматизированные методы построения компиляторов. Программы LEX и YACC. Моделирование функционирования распознавателя для грамматик простого предшествования.

## 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  $\alpha$  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  $\varepsilon$ .

Определение 1.4. Формальное определение цепочки символов в алфавите  $V$ :

- 1)  $\varepsilon$  - цепочка в алфавите  $V$ ;
- 2) если  $\alpha$  - цепочка в алфавите  $V$  и  $a$  - символ этого алфавита, то  $a\alpha$  - цепочка в алфавите  $V$ ;
- 3)  $\beta$  - цепочка в алфавите  $V$  тогда и только тогда, когда она является таковой в силу утверждений 1) и 2).

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

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

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

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  $(\alpha, \beta)$  множества  $P$  называется правилом вывода и записывается в виде  $\alpha \rightarrow \beta$  (читается: «из цепочки  $\alpha$  выводится цепочка  $\beta$ »);

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

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

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  $\gamma_0, \gamma_1, K, \gamma_n$  ( $n \geq 0$ ) такая, что  $\alpha = \gamma_0 \Rightarrow \gamma_1 \Rightarrow K \Rightarrow \gamma_n = \beta$ .

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

Пример 1.4. Для грамматики  $G1$   $L(G1) = \{0^n 1^n \mid n > 0\}$ .

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

Определение 1.11. Грамматики  $G1$  и  $G2$  называются эквивалентными, если  $L(G1) = L(G2)$ .

Пример 1.5. Для грамматики  $G1$  эквивалентной будет грамматика  $G2 = (\{0, 1\}, \{S\}, P2, S)$ , где множество правил вывода  $P2$  содержит правила вида  $S \rightarrow 0S1 \mid 01$ .

Классификация грамматик по Хомскому

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

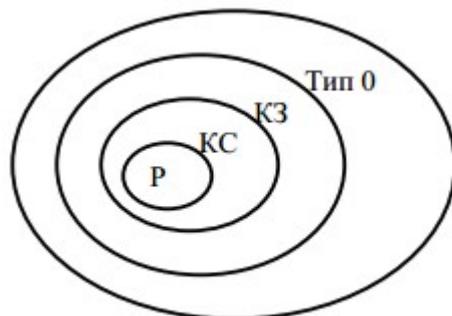
Тип 1. Грамматика  $G = (VT, VN, P, S)$  называется контекстно-зависимой грамматикой (КЗ-грамматикой), если каждое правило вывода из множества  $P$  имеет вид  $\alpha \rightarrow \beta$ , где  $\alpha \in (VT \cup VN)^+$ ,  $\beta \in (VT \cup VN)^*$  и  $|\alpha| \leq |\beta|$ .

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

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

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  $L(G) = \{a^2 b^n 2^{-1} \mid n \geq 1\}$  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  $L(G)=\{anbn^n \mid n \geq 1\}$  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)  $S \rightarrow aQb \mid acsb$ ;
- 2)  $Q \rightarrow cSc$ .

г) Регулярный язык  $L(G)=\{\omega \perp \mid \omega \in \{a, b\}^+\}$ , 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	$L(G) = \{wcwcw \mid w \in \{a, b\}^+\}$
5	$L(G) = \{c^{2^n} d^n \mid n > 0\}$
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	$L(G) = \{ab.b \mid a_i \in \{+, -\}, b \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}^+\}$

### Содержание отчета

По выполненной работе составляется отчет. Отчет выполняется в электронном виде. По выполненному отчету проводится защита лабораторной работы.

### Контрольные вопросы

1. Основные понятия и определения языков и грамматик.
2. Способы задания схем грамматик: форма Бэкуса-Наура; итерационная форма; синтаксические диаграммы.
3. Классификация грамматик и языков по Хомскому.
4. Соотношения между типами грамматик и языков.

## ЛАБОРАТОРНАЯ РАБОТА № 2. ПОСТРОЕНИЕ ГРАММАТИКИ МОДЕЛЬНОГО ЯЗЫКА (М-ЯЗЫКА).

### 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 автомат)  $P(Q)$ , 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.

Алгоритм 2.1. Построение КА по регулярной грамматике

Вход: регулярная грамматика  $G = (VT, VN, P, S)$ .

Выход: КА  $M = (Q, T, F, H, Z)$ .

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

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

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

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

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  $Q = VN = \{S, A, B, N\}$ , множество символов входного алфавита  $T = VT = \{a, b\}$ .

1.3 Значения сформированной функции переходов даны в таблице

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  $\epsilon$ -правила отсутствуют. Конечный автомат  $M$  - недетерминированный, граф НКА представлен на рисунке 2.1 слева.

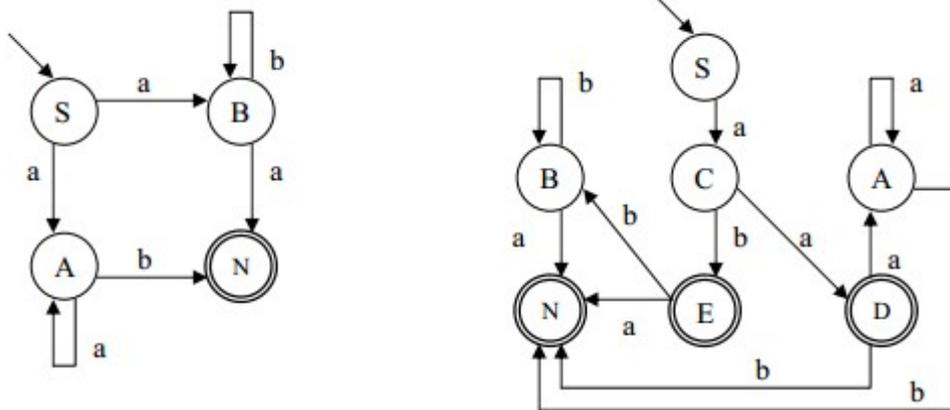


Рисунок 2.1 - Граф НКА (слева) и ДКА (справа) для  $P$ - грамматики

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,  $N = \{S\}$ ,  $Z' = \{N, D, E\}$ .

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.

### Содержание отчета

По выполненной работе составляется отчет. Отчет выполняется в электронном виде. По выполненному отчету проводится защита лабораторной работы.

### Контрольные вопросы

1. Построение грамматик и грамматики, описывающие основные конструкции языков программирования (описание списков, целых чисел без знака и идентификаторов, арифметических выражений, последовательности операторов присваивания, условных операторов и операторов цикла).
2. Автоматные грамматики. Конечные автоматы.
3. Детерминированные и недетерминированные КА. Алгоритм построения детерминированного КА по НКА. Минимизация КА.
4. Определение регулярного множества. Регулярные выражения. Свойства РВ.
5. Взаимосвязь регулярных множеств, регулярных грамматик и конечных автоматов.
6. Три способа задания регулярных языков. Построение КА по грамматике.
7. Связь регулярных выражений и регулярных грамматик. Связь регулярных выражений и конечных автоматов. Связь регулярных грамматик и конечных автоматов.
8. Построение конечного автомата на основе левосторонней грамматики. Построение левосторонней грамматики на основе конечного автомата.
9. Свойства регулярных языков (РЯ). Лемма о разрастании для регулярных языков.
10. Автоматы с магазинной памятью (МП-автоматы). Конечные автоматы с магазинной памятью. Работа магазинного автомата.
11. Язык, допускаемый магазинным автоматом. Построение магазинного автомата, пример. Распознавание цепочек с помощью МП-автоматов.
12. Свойства КС-языков. Лемма о разрастании для КС-языков.
13. Нормальные формы грамматик. Приведенные грамматики.
14. Преобразования грамматик. Удаление бесплодных и недостижимых символов. Удаление  $\lambda$ -правил и цепных правил. Устранение левой рекурсии.
15. Нормальная форма Хомского. Алгоритм преобразования грамматики в нормальную форму Хомского. Грамматики в нормальной форме Грейбах.

16. Универсальные алгоритмы разбора. Принципы работы распознавателей с возвратом. Алгоритмы разбора с возвратами.

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) $C \rightarrow bB \mid bC \mid a$ .
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) $V \rightarrow vV \mid )$ .

### ЛАБОРАТОРНАЯ РАБОТА № 3. ПОСТРОЕНИЕ ЛЕКСИЧЕСКОГО АНАЛИЗАТОРА (СКАНЕРА) ДЛЯ М-ЯЗЫКА. 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,  $n \in \mathbb{N} \cup \{0\}$ , если, находясь в одном из этих состояний и получив на вход любую цепочку символов  $\omega$ :  $\omega \in V_T^*$ ,  $|\omega| \leq n$ , 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: КАМ' =  $(Q', T, F', H, Z')$ .

**Шаг 1.** Поместить начальное состояние КА в список достижимых состояний  $Q_d$ , т.е.  $Q_d^0 = H$ .

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

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

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

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

**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  $M$

<i>F</i>	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>	<i>F</i>	<i>G</i>
<i>a</i>	<i>B</i>			<i>C</i>	<i>B</i>	<i>D</i>	<i>F</i>
<i>b</i>	<i>C</i>	<i>D</i>	<i>E</i>	<i>E</i>	<i>D</i>	<i>G</i>	<i>E</i>

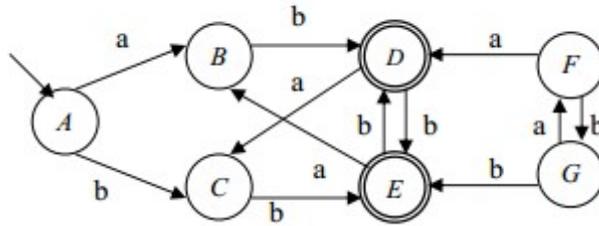


Рисунок 3.1 – Граф исходного конечного автомата М

Последовательность устранения недостижимых состояний КА имеет вид:

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

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

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

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

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

Функция переходов автомата М' представлена в таблице 3.2.

Таблица 3.2 - Функция переходов автомата М'

<i>F</i>	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
<i>a</i>	<i>B</i>			<i>C</i>	<i>B</i>
<i>b</i>	<i>C</i>	<i>D</i>	<i>E</i>	<i>E</i>	<i>D</i>

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

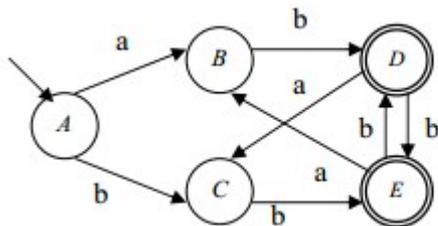


Рисунок 3.2 - Граф КА М' после устранения недостижимых состояний

Алгоритм 3.2. Объединение эквивалентных состояний КА

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

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

Шаг 1. На первом шаге строим нулевое разбиение  $R(0)$ , состоящее из двух классов эквивалентности: заключительные состояния  $KA - Z$  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  $KA M''$  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.

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

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

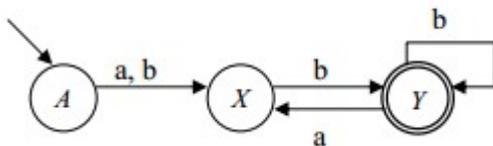


Рисунок 3.3 – Граф минимального  $KA M''$

Постановка задачи к лабораторной работе № 3

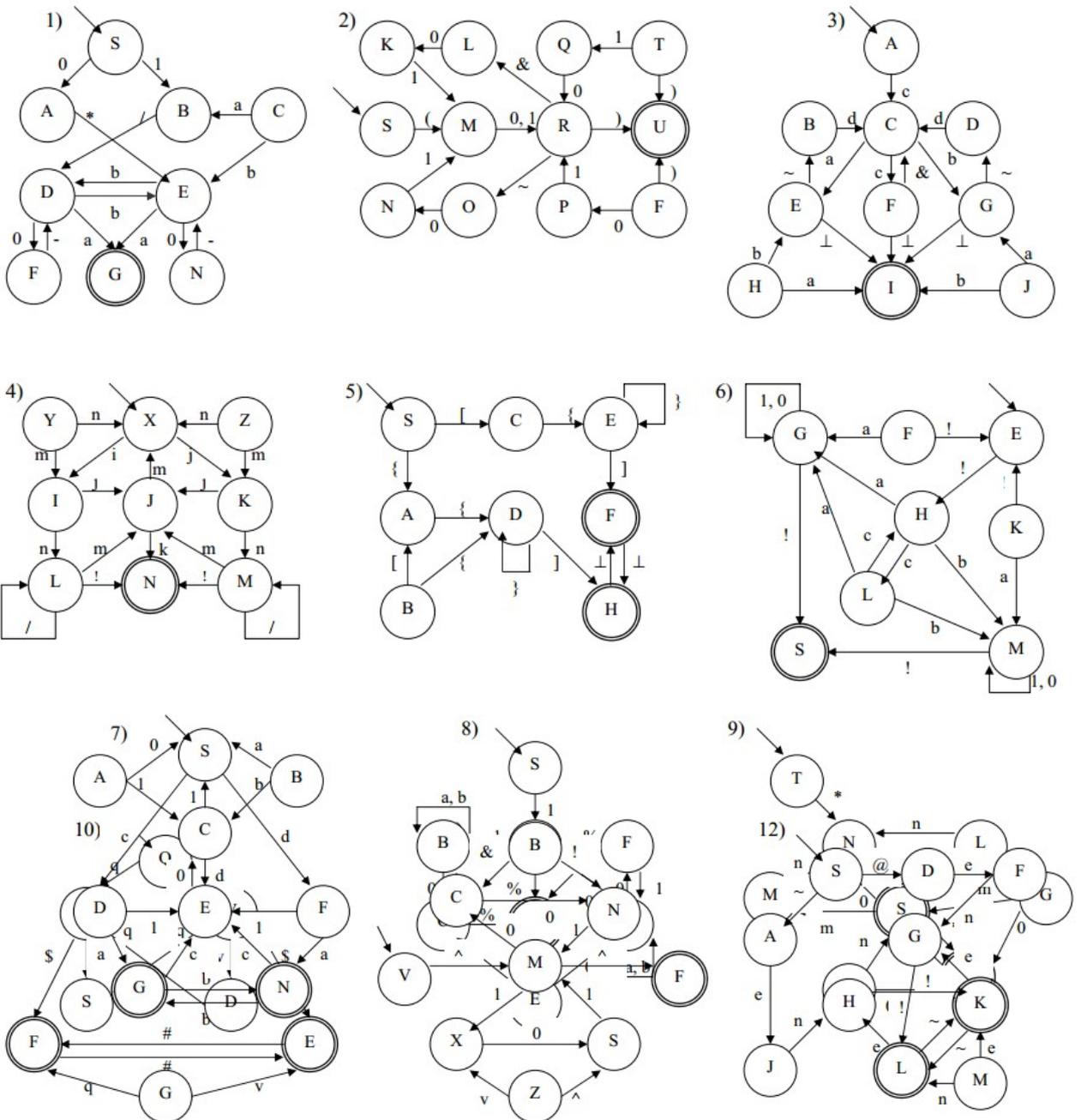
Разработать программное средство, реализующее следующие функции:

- 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.

### Варианты индивидуальных заданий к лабораторной работе № 3



### 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.

### Содержание отчета

По выполненной работе составляется отчет. Отчет выполняется в электронном виде. По выполненному отчету проводится защита лабораторной работы.

### Контрольные вопросы

1. Нисходящий распознаватель с возвратом. Распознаватель на основе алгоритма «сдвиг-свертка». Табличные распознаватели.
2. Алгоритмы Кока-Янгера-Касами и Эрли.
3. Метод рекурсивного спуска. О применимости метода рекурсивного спуска.
4. Алгоритмы разбора частного вида. Принципы построения распознавателей КС-языков без возвратов.
5. LL(k)-грамматики. Алгоритм разбора для LL(k)-грамматик.
6. Построение множеств FIRST(k) и FOLLOW(k). LR(k)-грамматики. Алгоритм разбора для LR(k)-грамматик.
7. Восходящие распознаватели КС-языков без возвратов. Принципы построения
8. Распознавателей для LR(k)-грамматик. Грамматики простого предшествования. Грамматики операторного предшествования. Иерархия классов КС-языков.
9. Элементы теории трансляции. Трансляторы, компиляторы, интерпретаторы; определения и назначение. Способы задания языков.
10. Общая схема работы транслятора. Понятие прохода.
11. Схема работы компилятора. Многопроходные и однопроходные компиляторы.
12. Интерпретаторы, особенности их построения и работы.
13. Ассемблеры. Трансляторы с языка ассемблера.
14. Способы задания языков. Вопросы, решаемые при задании языка программирования.
15. Лексические анализаторы. Задачи лексического анализа. Лексемы. Сканеры. Методы построения сканеров.
16. Таблицы идентификаторов, символов и их организация. Методы организации таблиц символов (бинарные деревья, хэш-функции, цепочки и др.). Методы поиска в таблицах.

### ЛАБОРАТОРНАЯ РАБОТА № 4. ПОСТРОЕНИЕ СИНТАКСИЧЕСКОГО И СЕМАНТИЧЕСКОГО АНАЛИЗАТОРА ДЛЯ М-ЯЗЫКА. 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,  $\epsilon$ -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  $N \cup A \cup P$

$i = i-1 \cup \{ \mid ( \rightarrow \alpha ) \in \text{и} \}. \alpha \in (N_{i-1} \cup V_T)^*$

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)  $B \rightarrow \cdot$ . 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. Беспольными 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 грамматику  $G'$  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 грамматику  $G' = (\{a, b, c\}, \{S, B, C\}, P', S)$  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  $\epsilon$ -правил

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

Выход: Эквивалентная КС-грамматика  $G' = (V_T, V_N', P', S')$  без  $\epsilon$ -правил

для всех нетерминальных символов, кроме начального, который не должен встречаться в правых частях правил грамматики.

Шаг 1. В исходной грамматике  $G$  найти  $\varepsilon$ -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  $\varepsilon$ -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  $\varepsilon$ -порождающих нетерминалов в правой части. Полученные при этом  $\varepsilon$ -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  $V_{N'} = V_N$ ,  $S$ .

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)  $0A \rightarrow 0A \mid \varepsilon$ ; 3)  $B \rightarrow 1B \mid \varepsilon$ .

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)  $A \rightarrow 0A \mid \varepsilon$ ; 4)  $B \rightarrow 1B \mid \varepsilon$ .

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  $N^A_0 = \{A\}$

1.2 Вычислить  $N^A_i = N^A_{i-1} \cup \{C \mid (B \rightarrow C) \in P, B \in N^A_{i-1}, C \in V_N\}$

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

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)  $N \rightarrow N^+ | n$ . 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)  $L \rightarrow N^+ | n$ ; 2)  $M \rightarrow N^+ | n$ ; 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  $\alpha$  в каждой строке альтернативы:  $X \rightarrow \alpha(\beta_1 | \beta_2 | \dots | \beta_n)$

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  $S$  и

заменяя 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  $X$ :

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

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

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

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

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

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:

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

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

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

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

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

#### 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)  $\varepsilon$ -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	$G = (\{R, T, F, G, K\}, \{m, i, j, k, \wedge, \sim, \perp\}, P, R)$ , 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) $V \rightarrow *0 \mid /1$ .

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	$G = (\{Y, K, M, L, S\}, \{a, b, *, /, \wedge\}, P, Y)$ , 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) $L \rightarrow L^{\wedge} \mid \wedge a$ .

### Содержание отчета

По выполненной работе составляется отчет. Отчет выполняется в электронном виде. По выполненному отчету проводится защита лабораторной работы.

### Контрольные вопросы

1. Синтаксический анализ. Назначение синтаксического анализа. Задачи, решаемые синтаксическим анализатором. Взаимосвязь с лексическим анализатором.
2. Семантический анализ. Контекстные условия языков программирования. Задачи, решаемые семантическим анализатором. Обработка описаний. Контроль контекстных условий в операторах.
3. Распределение памяти. Идентификация переменных. Память для типов данных.

### ЛАБОРАТОРНАЯ РАБОТА № 5. ГЕНЕРАЦИЯ ВНУТРЕННЕГО ПРЕДСТАВЛЕНИЯ ПРОГРАММ. ПОЛИЗ. ПОСТРОЕНИЕ АВТОМАТА С МАГАЗИННОЙ ПАМЯТЬЮ ПО КОНТЕКСТНО-СВОБОДНОЙ ГРАММАТИКЕ.

**Цель:** закрепить понятия «автомат с магазинной памятью (МПавтомат)», «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,  $q \in Q$ ;

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

$\alpha$  - содержимое магазина,  $\alpha \in N^*$ .

Общая схема МП-автомата представлена на рисунке 5.1.

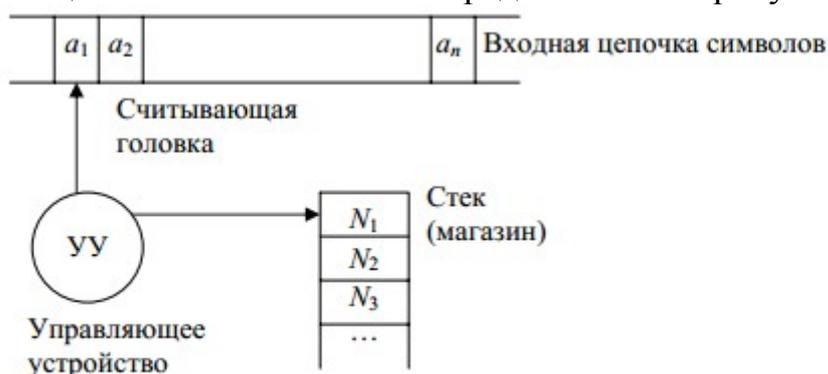


Рисунок 5.1 – Схема МП-автомата

Алгоритм 5.1. Функционирование МП-автомата

Начальной конфигурацией МП-автомата является конфигурация  $(q_0, \omega, N_0)$ .

Шаг работы МП-автомата будем представлять в виде отношения непосредственного следования конфигураций (обозначается « $\Rightarrow$ ») и отношения достижимости конфигураций (обозначается « $\Rightarrow^*$ »). Если одним из значений магазинной функции  $F(q \in Q, t \in (T \cup \{\varepsilon\}), S \in N)$  является  $(q' \in Q, \gamma \in N^*, \alpha \in N)$ , 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  $\gamma$  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  $\varepsilon$ -шагом, который может выполняться даже после завершения чтения входной строки. Заключительной конфигурацией МП-автомата является конфигурация  $(q, \varepsilon, \alpha)$ , где  $q \in Z$ .

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

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

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

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 предложения, первоначально это начальный символ грамматики. Очередная предложение 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)  $F(q, \varepsilon, S) = (q, S+A)$ ,  $F(q, \varepsilon, A) = (q, (S))$ ;  
 $F(q, \varepsilon, A) = (q, a)$ .

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

Распознавание строки (a) построенным МП-автоматом представлено 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	$q$	$\varepsilon$	$\varepsilon$

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 предложения. 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	$q$	$\varepsilon$	#(S)
7	$q$	$\varepsilon$	#A
8	$q$	$\varepsilon$	#S
9	$r$	$\varepsilon$	$\varepsilon$

### Постановка задачи к лабораторной работе № 5

Разработать программное средство, реализующее следующие функции:

- 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.

### Содержание отчета

По выполненной работе составляется отчет. Отчет выполняется в электронном виде. По выполненному отчету проводится защита лабораторной работы.

### Контрольные вопросы

1. Генерация внутреннего представления программ. Назначение этапа. Внутреннее представление программы в виде дерева операций. Польская инверсная запись (ПОЛИЗ).
2. Преобразование дерева операций в ассемблерный код и триады.
3. Использование СУ-перевода для перевода выражений в польскую запись.

### ЛАБОРАТОРНАЯ РАБОТА № 6. ПОСТРОЕНИЕ ОБЪЕКТНОГО КОДА МОДУЛЯ М-ЯЗЫКА. АССЕМБЛЕРЫ. 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 верхушке 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( $k, A$ ) =  $\{\omega \in V_T^* \mid \exists \text{ вывод } S \Rightarrow^* \alpha A \gamma \text{ и } \omega \in \text{FIRST}(k, \gamma); \alpha, \gamma \in V^*, A \in V_N, k > 0\}$ .

### **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:

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

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

Т.е. если для символа  $A$  отсутствует правило вида  $A \rightarrow \varepsilon$ , то все множества FIRST( $1, \alpha_1$ ), FIRST( $1, \alpha_2$ ), ..., 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  $\varepsilon$ -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 $_i$ ( $1, B$ ),  $\forall B \in (\text{FIRST}(1, A) \cap V_N)$ .

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

### **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.  $\forall A \in V_N \text{ FOLLOW}_0(1, A) = \{X \mid \exists B \rightarrow \alpha AX\beta \in P, B \in V_N, X \in (V_T \cup V_N), \alpha, \beta \in (V_T \cup V_N)^*\}$ .

Шаг 2. Внести пустую строку во множество  $\text{FOLLOW}(1, S)$ , т.е.  $\text{FOLLOW}(1, S) = \text{FOLLOW}(1, S) \cup \{\varepsilon\}$ .

Шаг 3. Для всех  $A \in V_N$  вычислить:  $\text{FOLLOW}'_i(1, A) = \text{FOLLOW}_i(1, A) \cup \text{FIRST}(1, B), \forall B \in (\text{FOLLOW}_i(1, A) \cap V_N)$ .

Шаг 4. Для всех  $A \in V_N$  положить:  $\text{FOLLOW}''_i(1, A) = \text{FOLLOW}'_i(1, A) \cup \text{FOLLOW}'_i(1, B), \forall B \in (\text{FOLLOW}'_i(1, A) \cap V_N)$ , если  $\exists$  правило  $B \rightarrow \varepsilon$ .

Шаг 5. Для всех  $A \in V_N$  определить:  $\text{FOLLOW}_{i+1}(1, A) = \text{FOLLOW}''_i(1, A) \cup \text{FOLLOW}''_i(1, B)$ , для всех нетерминальных символов  $B \in V_N$ , имеющих правило вида  $B \rightarrow \alpha A, \alpha \in (V_T \cup V_N)^*$ .

Шаг 6. Если существует  $A \in V_N$  такой, что  $\text{FOLLOW}_{i+1}(1, A) \neq \text{FOLLOW}_i(1, A)$ ,

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

### 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  $\varepsilon$  либо будет обнаружена ошибка в алгоритме разбора, выполняем одно из следующих действий:

- если на вершущке стека находится нетерминальный символ  $A$  и очередной символ входной строки символ  $a$ , то выполняем операцию «свертка» по правилу  $A \rightarrow x$  при условии, что  $a \in \text{FIRST}(1, x)$ , т.е. извлекаем из стека символ  $A$  и заносим в стек строку  $x$ , не меняя содержимого входного буфера;

- если на вершущке стека находится нетерминальный символ  $A$  и очередной символ входной строки символ  $a$ , то выполняем операцию «свертка» по правилу  $A \rightarrow \varepsilon$  при условии, что  $a \in \text{FOLLOW}(1, A)$ , т.е. извлекаем из стека символ  $A$  и заносим в стек строку  $\varepsilon$ , не меняя содержимого входного буфера;

- если на вершущке 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;

**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  $\varepsilon$ -правил:

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

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

1)  $S \rightarrow TR \mid T$ ; 2)  $R \rightarrow +TR \mid +T \mid -TR \mid -T$ ; 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	$S$	)	), $\epsilon$	), $\epsilon$
	$R$	$\emptyset$	$\emptyset$	$\emptyset$
	$T$	$R$	$R, +, -$	$R, +, -$
1	$S$	), $\epsilon$	), $\epsilon$	), $\epsilon$
	$R$	), $\epsilon$	), $\epsilon$	), $\epsilon$
	$T$	$R, +, -$	$R, +, -$	$R, +, -, ), \epsilon$
2	$S$	), $\epsilon$	), $\epsilon$	), $\epsilon$
	$R$	), $\epsilon$	), $\epsilon$	), $\epsilon$
	$T$	$R, +, -, ), \epsilon$	$R, +, -, ), \epsilon$	$R, +, -, ), \epsilon$
$FOLLOW(1, S)$		), $\epsilon$		
$FOLLOW(1, R)$		), $\epsilon$		
$FOLLOW(1, T)$		$+, -, ), \epsilon$		

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  $FOLLOW(1, A)$

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

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
$R$	$\varepsilon$	свертка $R \rightarrow \varepsilon$ , т.к. $\varepsilon \in FOLLOW(1, R)$
$\varepsilon$	$\varepsilon$	строка принята полностью

Шаг 6. Получили следующую цепочку вывода:

$$S \Rightarrow TR \Rightarrow (S)R \Rightarrow (TR)R \Rightarrow (aR)R \Rightarrow (a+TR)R \Rightarrow (a+(S)R)R \Rightarrow (a+(TR)R)R \Rightarrow (a+(bR)R)R \Rightarrow (a+(b-TR)R)R \Rightarrow (a+(b-aR)R)R \Rightarrow (a+(b-a)R)R \Rightarrow (a+(b-a))R \Rightarrow (a+(b-a)).$$

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.

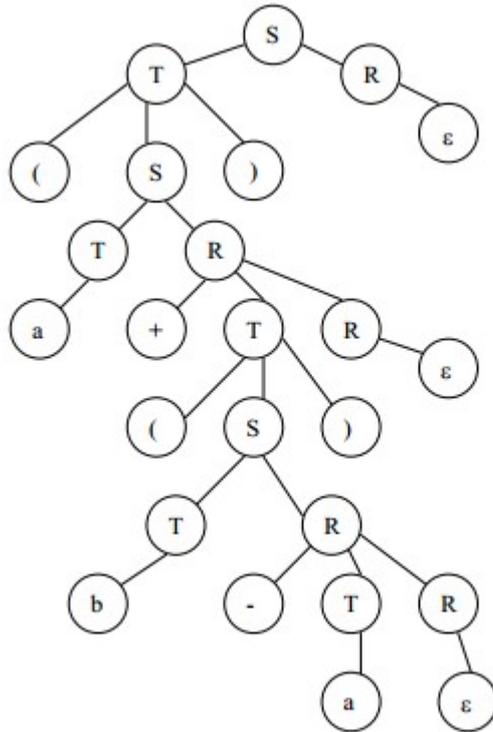


Рисунок 6.1 – Дерево вывода для цепочки  $(a+(b-a))$  в грамматике  $G$

### Содержание отчета

По выполненной работе составляется отчет. Отчет выполняется в электронном виде. По выполненному отчету проводится защита лабораторной работы.

### Контрольные вопросы

1. Генератор внутреннего представления программы на модельном языке. Вычисление выражений в обратной польской записи. Интерпретатор ПОЛИЗа для модельного языка.
2. Оптимизация кода. Назначение, методы и формы оптимизации программ. Оптимизация внутреннего представления программы.
3. Алгоритмы свертки операций. Исключение лишних операций.

### ЛАБОРАТОРНАЯ РАБОТА № 7. АВТОМАТИЗИРОВАННЫЕ МЕТОДЫ ПОСТРОЕНИЯ КОМПИЛЯТОРОВ, ПРОГРАММЫ LEX И YACC. 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)  $V_i = \cdot V_j$  ( $\forall V_i, V_j \in V$ ), если и только если существует правило  $A \rightarrow xV_iV_jy \in P$ , где  $x, y \in V^*$ ;

б)  $V_i < \cdot V_j$  ( $\forall V_i, V_j \in V$ ), если и только если существует правило  $A \rightarrow xV_iDy \in P$  и вывод  $D \Rightarrow^* V_jz$ , где  $A, D \in V_N, x, y, z \in V^*$ ;

в)  $V_i \cdot > V_j$  ( $\forall V_i, V_j \in V$ ), если и только если существует правило  $A \rightarrow xCV_jy$  и вывод  $C \Rightarrow^* zV_i$  или существует правило  $A \rightarrow xCDy \in P$  и вывод  $C \Rightarrow^* zV_i$  и  $D \Rightarrow^* V_jw$ , 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:

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

### Алгоритм 7.1. Поиск основы сентенции грамматики

Если грамматика является грамматикой простого предшествования, то для поиска основы каждой ее сентенции надо просматривать элементы сентенции слева направо и найти самую левую пару символов  $x_j$  и  $x_{j+1}$ , такую что  $x_j \cdot > x_{j+1}$ . Окончанием основы сентенции будет  $x_j$ . Далее просматривать элементы сентенции 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 сентенции, имеющая вид  $x_i x_{i+1} \dots x_{j-1} x_j$ . Схема поиска основы сентенции грамматики представлена на рисунке 7.1.

Рисунок 7.1 – Схема поиска основы сентенции 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:

-  $V_i = \cdot V_j$  ( $\forall V_i, V_j \in V$ ), если и только если существует правило  $A \rightarrow xV_iV_jy \in P$ , где  $A \in V_N$ ,  $x, y \in V^*$ ;

-  $V_i < \cdot V_j$  ( $\forall V_i, V_j \in V$ ), если и только если существует правило  $A \rightarrow xV_iDy \in P$  и  $V_j \in L(D)$ , где  $A, D \in V_N$ ,  $x, y \in V^*$ ;

-  $V_i \cdot > V_j$  ( $\forall V_i, V_j \in V$ ), если и только если существует правило  $A \rightarrow xCB_jy$  и  $V_i \in R(C)$  или существует правило  $A \rightarrow xCDy \in P$  и  $V_i \in R(C)$ ,  $V_j \in L(D)$ , 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  $A'$ ,  $A''$ , ..., 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 верхушку 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 верхушке 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 верхушке стека «основу» сентенции, т.е. все символы, имеющие равное предшествование или один символ на верхушке 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  $=\cdot$ , 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  $)$ , а также строки  $($  и столбца  $R$ .

Таблица 7.2 – Матрица предшествования символов грамматики

Символы	$S$	$R$	$a$	$($	$)$	$\perp_k$
$S$			$=\cdot$			
$R$			$\cdot>$			$\cdot>$
$a$			$\cdot>$		$=\cdot$	$\cdot>$
$($	$<\cdot$	$=\cdot$	$<\cdot$	$<\cdot$		
$)$			$\cdot>$			$\cdot>$
$\perp_n$			$<\cdot$	$<\cdot$		

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

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



- 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.

### **Содержание отчета**

По выполненной работе составляется отчет. Отчет выполняется в электронном виде. По выполненному отчету проводится защита лабораторной работы.

### **Контрольные вопросы**

1. Автоматизированные методы построения компиляторов. Возможности и использование программ LEX и YACC.
2. Мобильность программного обеспечения. Макроязык, его предназначение и структура. Макропроцессоры и их свойства. Макровывод, макроопределение и макрорасширение – составные части макропроцессора. Структура данных макропроцессора.

## **2. УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ**

**Перечень основной и дополнительной литературы, необходимой для освоения дисциплины**

### **Перечень основной литературы:**

1. В. Н. Пильщикова, В. Г. Абрамов, А. А. Вылиток, И. В. Горячая. Машины Тьюринга и алгоритмы Маркова. Решение задач. — М.: МГУ, 2006. [11]. А. Ахо., Р. Сети, Дж. Ульман. Компиляторы: принципы, технологии, инструменты. — М.: «Вильямс», 2001.
2. А. Ахо, М. Лам, Р. Сети, Дж. Ульман. Компиляторы: принципы, технологии и инструментарий. — М.: «Вильямс», 2008.

### **Перечень дополнительной литературы:**

1. Олифер, В. Г. Компьютерные сети. Принципы, технологии, протоколы: учеб. пособие / В. Г. Олифер, Н. А. Олифер. - 4-е изд. - СПб.: Питер, 2011.
2. Таненбаум, Э. С. Архитектура компьютера / Э. С. Таненбаум; пер.: Ю. Гороховский, Д. Шинтяков. - 5-е изд. - СПб.: Питер, 2009.

**Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимых для освоения дисциплины:**

1. <http://www.intuit.ru> – сайт дистанционного образования в области информационных технологий
2. <http://www.iqlib.ru> - интернет библиотека образовательных изданий, в которой собраны электронные учебники, справочные и учебные пособия;
3. <http://www.biblioclub.ru> - электронная библиотечная система «Университетская библиотека – online»: специализируется на учебных материалах для ВУЗов по научно-гуманитарной тематике.

4. <http://window.edu.ru> – образовательные ресурсы ведущих вузов

## Пример оформления отчета лабораторной работы

### 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  $q, t) = p$ , 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>fable</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

## 6 Алгоритм решения задачи

Укрупненная схема алгоритма программного средства представлена на рисунке 6.1.

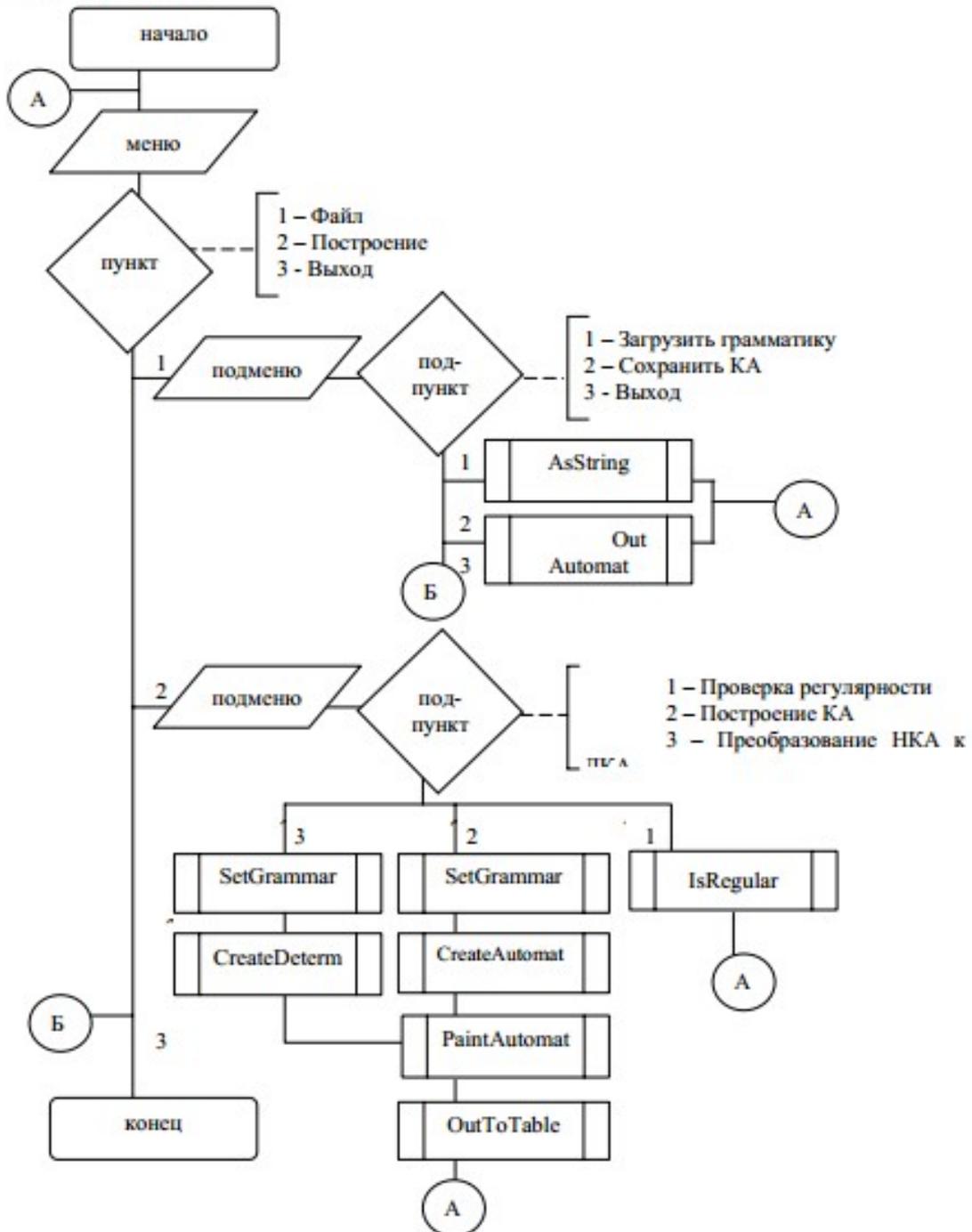


Рисунок 6.1 – Укрупненная схема алгоритма программного средства

## 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.

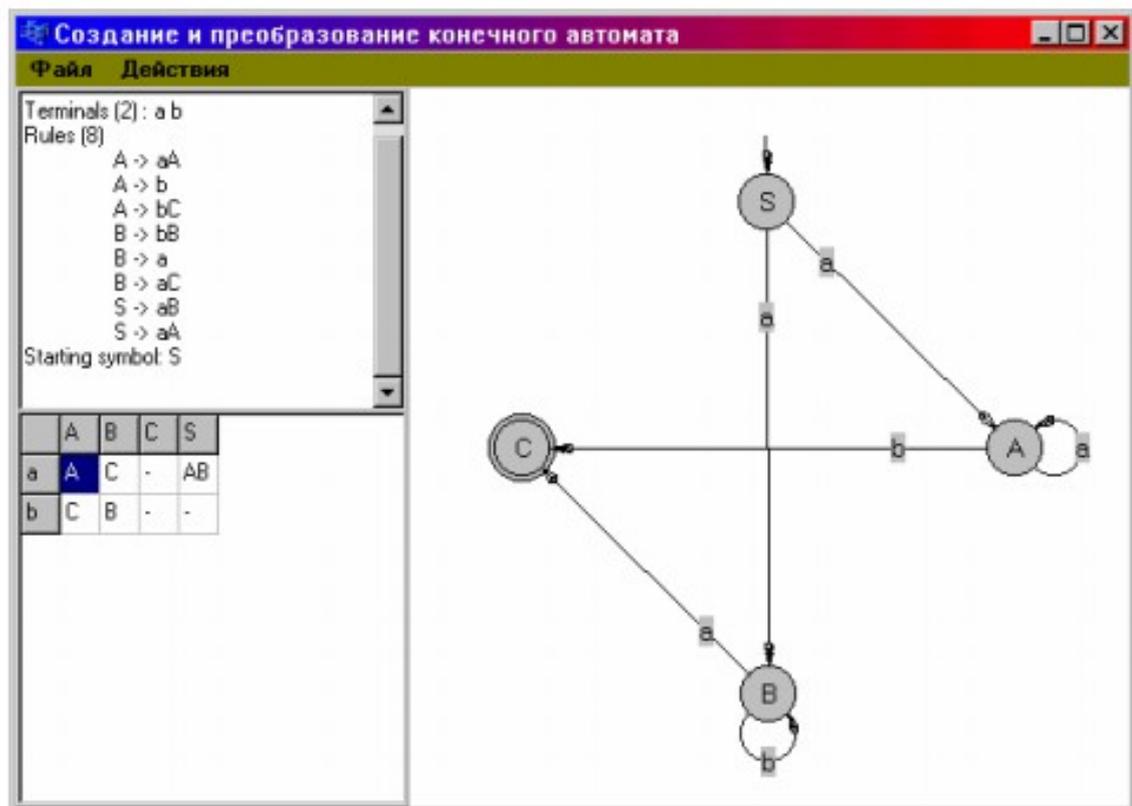


Рисунок А.1 – Исходная КС-грамматика и НКА

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

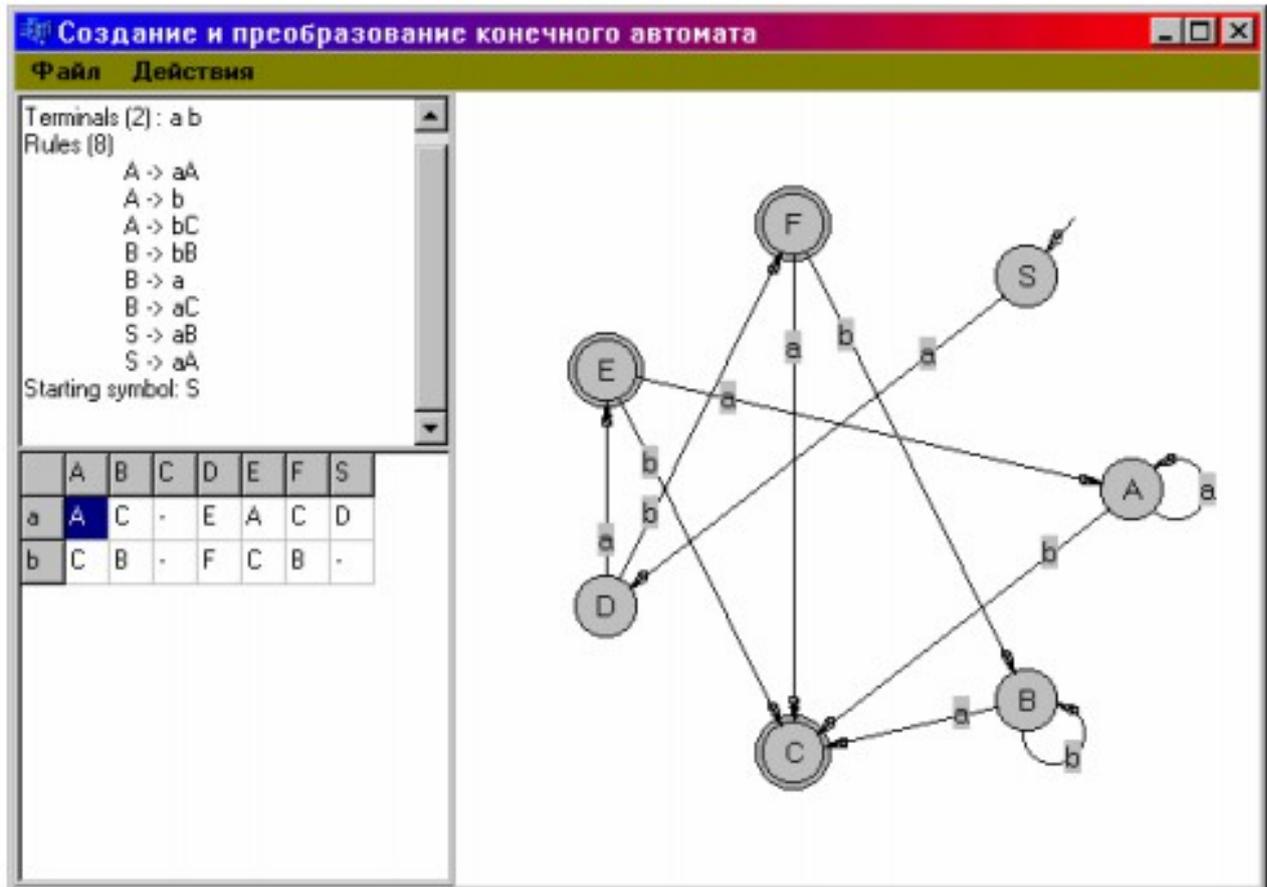


Рисунок А.2 – Результирующий ДКА

## Приложение Б

(обязательное)

### Текст программы

#### Unit2.h

```
#include <fstream.h>
#include <math.h>
#include <sysutils.hpp>
#include <graphics.hpp>
#include <grid.h>
#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 += "\n";
        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;

```

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное автономное образовательное учреждение  
высшего образования  
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»  
Пятигорский институт (филиал) СКФУ

**МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ СТУДЕНТОВ ПО ОРГАНИЗАЦИИ  
САМОСТОЯТЕЛЬНОЙ РАБОТЫ ПО ДИСЦИПЛИНЕ  
ТЕОРИЯ ФОРМАЛЬНЫХ ЯЗЫКОВ, ГРАММАТИК И ПОСТРОЕНИЕ  
ТРАНСЛЯТОРОВ**

Направление подготовки	<b>09.04.02</b>
Направленность (профиль)	<b>Информационные системы и технологии «Технологии работы с данными и знаниями, анализ информации»</b>
Квалификация выпускника	<b>Магистр</b>

Пятигорск, 2024

Введение.....	51
1.....	Цель, задачи

51

2.....	Технологическая карта самостоятельной работы обучающегося	52
3.....	Рекомендации для самоподготовки	52
3.1.	Подготовка к лекциям. Самостоятельное изучение литературы.....	52
3.2.	Подготовка к лабораторным работам.....	55
3.3.	Подготовка к выполнению контрольной работы.....	57
4.	Учебно-методическое и информационное обеспечение дисциплины.....	73

## ВВЕДЕНИЕ

Методические указания содержат перечень тем с вопросами для самостоятельной проработки, требования к оформлению работ и пример выполнения задания. Теоретической основой подготовки специалиста являются знания в области информатики, вычислительной систем.

### 1. ЦЕЛЬ, ЗАДАЧИ

Целью дисциплины «Теория формальных языков, грамматик и построение трансляторов» является ознакомление с основными понятиями и методами использования теории формальных грамматик, овладение теоретическими знаниями о фазы грамматического разбора при компиляции и интерпретации формальных текстов и практическими навыками по алгоритмам лексического, грамматического и семантического анализа.

Задачами дисциплины «Теория формальных языков, грамматик и построение трансляторов» являются:

- освоение принципов построения формальных языков программирования, работы компиляторов;
- разработка алгоритма с использованием грамматических структур формальных грамматик;
- знакомство с работой лексического анализатора языка программирования;
- моделирование лексического анализатора математического выражения.

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 рекомендациях содержатся материалы, необходимые для самостоятельного изучения.

## 2. ТЕХНОЛОГИЧЕСКАЯ КАРТА САМОСТОЯТЕЛЬНОЙ РАБОТЫ ОБУЧАЮЩЕГОСЯ

Для студентов очной формы обучения:

Коды реализуемых компетенций, индикатора(ов)	Вид деятельности студентов	Средства и технологии оценки	Объем часов, в том числе		
			СРС	Контактная работа с преподавателями	Всего
<b>3 семестр</b>					
ПК-8 (ИД-1, ИД-2, ИД-3), ПК-11 (ИД-1, ИД-2, ИД-3), ПК-12 (ИД-1, ИД-2, ИД-3)	Подготовка к лекциям	Собеседование	1,62	0,18	1,8
ПК-8 (ИД-1, ИД-2, ИД-3), ПК-11 (ИД-1, ИД-2, ИД-3), ПК-12 (ИД-1, ИД-2, ИД-3)	Самостоятельное изучение литературы	Собеседование	49,32	5,48	54,8
ПК-8 (ИД-1, ИД-2, ИД-3), ПК-11 (ИД-1, ИД-2, ИД-3), ПК-12 (ИД-1, ИД-2, ИД-3)	Подготовка и выполнение лабораторных работ	Отчет письменный	4,86	0,54	5,4
ПК-8 (ИД-1, ИД-2, ИД-3), ПК-11 (ИД-1, ИД-2, ИД-3), ПК-12 (ИД-1, ИД-2, ИД-3)	Выполнение контрольной работы	Контрольная работа	9	1	10
<b>Итого за 3 семестр</b>			<b>64,8</b>	<b>7,2</b>	<b>72</b>

Для студентов заочной формы обучения:

Коды реализуемых компетенций, индикатора(ов)	Вид деятельности студентов	Средства и технологии оценки	Объем часов, в том числе		
			СРС	Контактная работа с преподавателями	Всего
<b>3 семестр</b>					

ПК-8 (ИД-1, ИД-2, ИД-3), ПК-11 (ИД-1, ИД-2, ИД-3), ПК-12 (ИД-1, ИД-2, ИД-3)	Подготовка к лекциям	Собеседование	0,54	0,06	0,6
ПК-8 (ИД-1, ИД-2, ИД-3), ПК-11 (ИД-1, ИД-2, ИД-3), ПК-12 (ИД-1, ИД-2, ИД-3)	Самостоятельное изучение литературы	Собеседование	99,54	11,06	110,6
ПК-8 (ИД-1, ИД-2, ИД-3), ПК-11 (ИД-1, ИД-2, ИД-3), ПК-12 (ИД-1, ИД-2, ИД-3)	Подготовка и выполнение лабораторных работ	Отчет письменный	1,62	0,18	1,8
ПК-8 (ИД-1, ИД-2, ИД-3), ПК-11 (ИД-1, ИД-2, ИД-3), ПК-12 (ИД-1, ИД-2, ИД-3)	Выполнение контрольной работы	Контрольная работа	9	1	10
<b>Итого за 3 семестр</b>			<b>110,7</b>	<b>12,3</b>	<b>123</b>

### 3. РЕКОМЕНДАЦИИ ДЛЯ САМОПОДГОТОВКИ

#### 3.1. Подготовка к лекциям. Самостоятельное изучение литературы

Подготовить доклад и презентацию по вопросам выносимым на самостоятельную работу.

##### Тема 1. Формальные языки и грамматики.

1. Способы задания схем грамматик: форма Бэкуса-Наура.
2. Итерационная форма.
3. Синтаксические диаграммы.
4. Классификация грамматик и языков по Хомскому.
5. Соотношения между типами грамматик и языков.
6. Примеры грамматик и языков.
1. основные конструкции языков программирования (описание списков, целых чисел без знака и идентификаторов, арифметических выражений, последовательности операторов присваивания, условных операторов и операторов цикла).
2. Рекомендации по построению грамматик.

##### Тема 2. Регулярные множества и регулярные выражения (РВ), конечные автоматы и автоматы с магазинной памятью. КС-грамматики и алгоритмы разбора.

###### Базовый уровень

1. Определение регулярного множества.
2. Регулярные выражения. Свойства РВ.
3. Взаимосвязь регулярных множеств, регулярных грамматик и конечных автоматов.

4. Три способа задания регулярных языков.
5. Построение КА по грамматике.
6. Связь регулярных выражений и регулярных грамматик.
7. Связь регулярных выражений и конечных автоматов.
8. Связь регулярных грамматик и конечных автоматов.
9. Построение конечного автомата на основе левосторонней грамматики.
10. Построение левосторонней грамматики на основе конечного автомата.
11. Пример построения конечного автомата на основе задания грамматики.
12. Свойства регулярных языков (РЯ).
13. Лемма о разрастании для регулярных языков.
14. Автоматные грамматики. Конечные автоматы.
15. Детерминированные и недетерминированные КА.
16. Алгоритм построения детерминированного КА по НККА.
17. Минимизация КА. Конечные автоматы с магазинной памятью.
18. Работа магазинного автомата. Язык, допускаемый магазинным автоматом.
19. Построение магазинного автомата, пример.
20. Распознавание цепочек с помощью МП-автоматов.
21. Свойства КС-языков. Лемма о разрастании для КС-языков.
22. Приведенные грамматики. Преобразования грамматик.
23. Удаление бесплодных и недостижимых символов. Удаление  $\lambda$ -правил и цепных правил.
24. Устранение левой рекурсии. Нормальная форма Хомского.
25. Алгоритм преобразования грамматики в нормальную форму Хомского.
26. Грамматики в нормальной форме Грейбах.
27. Принципы работы распознавателей с возвратом.
28. Алгоритмы разбора с возвратами. Нисходящий распознаватель с возвратом.
29. Распознаватель на основе алгоритма «сдвиг-свертка».
30. Табличные распознаватели.
31. Алгоритмы Кока-Янгера-Касами и Эрли.
32. Метод рекурсивного спуска. О применимости метода рекурсивного спуска.
33. Принципы построения распознавателей КС-языков без возвратов.
34. LL(k)-грамматики. Алгоритм разбора для LL(k)-грамматик.
35. Построение множеств FIRST(k) и FOLLOW(k). LR(k)-грамматики.
36. Восходящие Распознаватели КС-языков без возвратов.
37. Принципы построения Распознавателей для LR(k)-грамматик.
38. Грамматики простого предшествования.
39. Грамматики операторного предшествования. Иерархия классов КС-языков.

### **Тема 3. Элементы теории трансляции. Операционные системы и их роль в процессе трансляции.**

1. Трансляторы, компиляторы, интерпретаторы.
2. Определения и назначение. Способы задания языков.
3. Общая схема работы транслятора. Понятие прохода.
4. Схема работы компилятора.
5. Многопроходные и однопходные компиляторы.
6. Интерпретаторы, особенности их построения и работы.
7. Ассемблеры. Трансляторы с языка ассемблера.
8. Способы задания языков. Вопросы, решаемые при задании языка программирования.
9. **Лексические анализаторы (сканеры).** Задачи лексического анализа. Лексемы. Сканеры.
10. Методы построения сканеров.
11. Таблицы идентификаторов, символов и их организация.

12. Методы организации таблиц символов (бинарные деревья, хэш-функции, цепочки и др.). Методы поиска в таблицах.
13. **Синтаксический анализ.** Назначение синтаксического анализа.
14. Задачи, решаемые синтаксическим анализатором.
15. Взаимосвязь с лексическим анализатором. Понятие прохода.
16. **Семантический анализ.** Контекстные условия языков программирования.
17. Задачи, решаемые семантическим анализатором. Обработка описаний.
18. Контроль контекстных условий в операторах. Распределение памяти.
19. Идентификация переменных. Память для типов данных.
20. Генерация внутреннего представления программ. Назначение этапа. Внутреннее представление программы в виде дерева операций.
21. Польская инверсная запись (ПОЛИЗ). Преобразование дерева операций в ассемблерный код и триады.
22. Синтаксически-управляемый перевод. Обратная польская запись. Использование СУ-перевода для перевода выражений в польскую запись.
23. Генератор внутреннего представления программы на модельном языке. Вычисление выражений в обратной польской записи.
24. Интерпретатор ПОЛИЗа для модельного языка. Классификация ОС.
25. Архитектура ОС. Внутреннее представление файлов и системные операции работы с файлами. Структура процессов.
26. Пользовательский интерфейс операционной среды. Управление процессами ОС. Управление задачами. Управление памятью. Управление вводом-выводом. Управление файлами.
27. Загрузчики. Функции загрузчика. Формат объектного модуля. Распределение памяти. Виды переменных. Статическое и динамическое связывание.
28. Настраиваемый и динамический загрузчики и особенности их работы. Подключение библиотек.

#### **Тема 4. Автоматизированные методы построения компиляторов.**

1. Возможности и использование программ LEX и YACC.
2. **Мобильность программного обеспечения.**
3. Макроязык, его предназначение и структура.
4. Макропроцессоры и их свойства.
5. Макровыводы, макроопределение и макрорасширение – составные части макропроцессора.
6. Структура данных макропроцессора.
7. Сравнение макросредств и подпрограмм.
8. Переносимый машинный язык.
  1. **Кросс-системы.** Назначение кросс-систем.
  2. Задачи, решаемые кросс-системами.
  3. Модели регистров, оперативной памяти, процессора. Время.
  4. Системы прерываний. Ввод-вывод.
  5. Взаимодействие с человеком-оператором.

#### **3.2. Подготовка к лабораторным работам**

При подготовке к лабораторным работам необходимо проработать следующие вопросы:

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

5. Основные понятия и определения языков и грамматик.
6. Способы задания схем грамматик: форма Бэкуса-Наура; итерационная форма; синтаксические диаграммы.
7. Классификация грамматик и языков по Хомскому.
8. Соотношения между типами грамматик и языков.

## **Лабораторная работа № 2. Построение грамматики модельного языка (М-языка).**

### **Построение конечного автомата по регулярной грамматике**

17. Построение грамматик и грамматики, описывающие основные конструкции языков программирования (описание списков, целых чисел без знака и идентификаторов, арифметических выражений, последовательности операторов присваивания, условных операторов и операторов цикла).
18. Автоматные грамматики. Конечные автоматы.
19. Детерминированные и недетерминированные КА. Алгоритм построения детерминированного КА по НКА. Минимизация КА.
20. Определение регулярного множества. Регулярные выражения. Свойства РВ.
21. Взаимосвязь регулярных множеств, регулярных грамматик и конечных автоматов.
22. Три способа задания регулярных языков. Построение КА по грамматике.
23. Связь регулярных выражений и регулярных грамматик. Связь регулярных выражений и конечных автоматов. Связь регулярных грамматик и конечных автоматов.
24. Построение конечного автомата на основе левосторонней грамматики. Построение левосторонней грамматики на основе конечного автомата.
25. Свойства регулярных языков (РЯ). Лемма о разрастании для регулярных языков.
26. Автоматы с магазинной памятью (МП-автоматы). Конечные автоматы с магазинной памятью. Работа магазинного автомата.
27. Язык, допускаемый магазинным автоматом. Построение магазинного автомата, пример. Распознавание цепочек с помощью МП-автоматов.
28. Свойства КС-языков. Лемма о разрастании для КС-языков.
29. Нормальные формы грамматик. Приведенные грамматики.
30. Преобразования грамматик. Удаление бесплодных и недостижимых символов. Удаление  $\lambda$ -правил и цепных правил. Устранение левой рекурсии.
31. Нормальная форма Хомского. Алгоритм преобразования грамматики в нормальную форму Хомского. Грамматики в нормальной форме Грейбах.
32. Универсальные алгоритмы разбора. Принципы работы распознавателей с возвратом. Алгоритмы разбора с возвратами.

## **Лабораторная работа № 3. Построение лексического анализатора (сканера) для М-языка. Минимизация конечных автоматов**

17. Нисходящий распознаватель с возвратом. Распознаватель на основе алгоритма «сдвиг-свертка». Табличные распознаватели.
18. Алгоритмы Кока-Янгера-Касами и Эрли.
19. Метод рекурсивного спуска. О применимости метода рекурсивного спуска.
20. Алгоритмы разбора частного вида. Принципы построения распознавателей КС-языков без возвратов.
21. LL(k)-грамматики. Алгоритм разбора для LL(k)-грамматик.
22. Построение множеств FIRST(k) и FOLLOW(k). LR(k)-грамматики. Алгоритм разбора для LR(k)-грамматик.
23. Восходящие распознаватели КС-языков без возвратов. Принципы построения
24. Распознавателей для LR(k)-грамматик. Грамматики простого предшествования. Грамматики операторного предшествования. Иерархия классов КС-языков.
25. Элементы теории трансляции. Трансляторы, компиляторы, интерпретаторы; определения и назначение. Способы задания языков.
26. Общая схема работы транслятора. Понятие прохода.
27. Схема работы компилятора. Многопроходные и однопроходные компиляторы.
28. Интерпретаторы, особенности их построения и работы.
29. Ассемблеры. Трансляторы с языка ассемблера.

30. Способы задания языков. Вопросы, решаемые при задании языка программирования.
31. Лексические анализаторы. Задачи лексического анализа. Лексемы. Сканеры. Методы построения сканеров.
32. Таблицы идентификаторов, символов и их организация. Методы организации таблиц символов (бинарные деревья, хэш-функции, цепочки и др.). Методы поиска в таблицах.

**Лабораторная работа № 4. Построение синтаксического и семантического анализатора для М-языка. Эквивалентные преобразования контекстно-свободных грамматик**

4. Синтаксический анализ. Назначение синтаксического анализа. Задачи, решаемые синтаксическим анализатором. Взаимосвязь с лексическим анализатором.
5. Семантический анализ. Контекстные условия языков программирования. Задачи, решаемые семантическим анализатором. Обработка описаний. Контроль контекстных условий в операторах.
6. Распределение памяти. Идентификация переменных. Память для типов данных.

**Лабораторная работа № 5. Генерация внутреннего представления программ. ПОЛИЗ. Построение автомата с магазинной памятью по контекстно-свободной грамматике.**

4. Генерация внутреннего представления программ. Назначение этапа. Внутреннее представление программы в виде дерева операций. Польская инверсная запись (ПОЛИЗ).
5. Преобразование дерева операций в ассемблерный код и триады.
6. Использование СУ-перевода для перевода выражений в польскую запись.

**Лабораторная работа № 6. Построение объектного кода модуля М-языка. Ассемблеры. Моделирование функционирования распознавателя для LL(1)-грамматик**

4. Генератор внутреннего представления программы на модельном языке. Вычисление выражений в обратной польской записи. Интерпретатор ПОЛИЗа для модельного языка.
5. Оптимизация кода. Назначение, методы и формы оптимизации программ. Оптимизация внутреннего представления программы.
6. Алгоритмы свертки операций. Исключение лишних операций.

**Лабораторная работа № 7. Автоматизированные методы построения компиляторов. Программы LEX и YACC. Моделирование функционирования распознавателя для грамматик простого предшествования.**

3. Автоматизированные методы построения компиляторов. Возможности и использование программ LEX и YACC.
4. Мобильность программного обеспечения. Макроязык, его предназначение и структура. Макропроцессоры и их свойства. Макровывод, макроопределение и макрорасширение – составные части макропроцессора. Структура данных макропроцессора.

**а. Подготовка к выполнению контрольной работы.**

Для выполнения контрольной работы следует следующий теоретический материал

**ОСНОВНЫЕ ПОНЯТИЯ ТЕОРИИ АЛГОРИТМОВ**

**Предварительные сведения**

Понятие алгоритма, являющееся одним из основных понятий математики, возникло задолго до появления вычислительных машин. На протяжении многих веков люди пользовались интуитивным понятием алгоритма. Арабский математик IX века Мухаммед ибн Муса Аль-Хорезми впервые выдвинул идею о том, что решение любой поставленной

математической и философской задачи может быть оформлено в виде последовательности механически выполняемых правил, т.е. может быть алгоритмизировано. Этому же мнению придерживались Декарт, Лейбниц, Гильберт.

Алгоритм - это строгая и четкая конечная система правил решения некоторого класса задач, определяющая процесс преобразования исходных данных в искомый результат. В рамках данного определения понятие алгоритма отождествлялось с понятием метода вычисления, традиции организации вычислений складывались веками и стали составной частью общей научной культуры, формулировались и успешно применялись на практике различные вычислительные алгоритмы. Поэтому понятие метода вычислений считалось

#### Основные требования к алгоритмам

Каждый алгоритм имеет дело с данными - входными, промежуточными и выходными. Данные как объекты, с которыми могут работать алгоритмы, должны быть четко определены и отличимы как друг от друга, так и от другой информации. Данные для своего размещения требуют памяти. Память обычно считается однородной и дискретной. Единицы измерения объема данных и памяти согласованы, при этом память может быть бесконечной.

Если выполнение алгоритма заканчивается получением результатов, то говорят, что он применим к рассматриваемой совокупности исходных данных. Любой применимый алгоритм имеет следующие основные свойства, раскрывающие его определение.

1. Дискретность. Это свойство состоит в том, что алгоритм должен представлять процесс решения задачи как последовательное выполнение простых (или ранее определенных) шагов. При этом для выполнения каждого шага алгоритма требуется некоторый конечный отрезок времени, то есть преобразование исходных данных в результат осуществляется во времени дискретно.
2. Определенность (или детерминированность). Это свойство состоит в том, что каждое правило алгоритма должно быть четким и однозначным. Благодаря этому свойству выполнение алгоритма носит механический характер и не требует никаких дополнительных указаний или сведений о решаемой задаче.
3. Результативность (или конечность). Это свойство состоит в том, что алгоритм должен приводить к решению задачи за конечное число шагов.
4. Массовость. Это свойство состоит в том, что алгоритм решения задачи разрабатывается в общем виде и должен быть применим для некоторого класса задач, различающихся лишь исходными данными. Для задания алгоритма необходимо выделить и описать следующие его элементы:
  - набор объектов, составляющих совокупность данных: исходных, промежуточных и конечных;
  - правило начала;
  - правила непосредственной переработки информации;
  - правило окончания;
  - правило извлечения результатов.

Алгоритм всегда рассчитан на конкретного исполнителя. Если исполнителем является компьютер, то алгоритм должен быть записан на языке программирования.

#### Математическое определение алгоритма

Интуитивное определение алгоритма не позволяет рассматривать свойства алгоритмов как свойства формальных объектов. Поэтому математическое определение алгоритма необходимо по следующим причинам:

- 1) только при наличии формального определения алгоритма можно сделать вывод о разрешимости или неразрешимости каких-либо проблем;
- 2) это дает возможность для сравнения алгоритмов, предназначенных для решения одинаковых задач;

3) это дает возможность для сравнения различных проблем по сложности алгоритмов их решения. Одна из причин расплывчатости интуитивного определения алгоритма состоит в том, что объектом алгоритма может оказаться все, что угодно. Поэтому естественно было начать с формализации понятия объекта. В вычислительных алгоритмах объектами работы являются числа, в алгоритме шахматной игры - фигуры и позиции, при алгоритмизации производственных процессов объектами служат, например, показания приборов. Однако алгоритмы оперируют не с объектами реального мира, а с изображениями этих объектов.

Поэтому алгоритмами в современной математике принято называть конструктивно задаваемые соответствия между изображениями объектов в абстрактных алфавитах. Абстрактным алфавитом называется любая конечная совокупность объектов, называемых буквами или символами данного алфавита. При этом природа этих объектов нас совершенно не интересует. Символом абстрактных алфавитов можно считать буквы алфавита какого-либо языка, цифры, любые значки и даже слова некоторого конкретного языка. Основным требованием к алфавиту является его конечность. Таким образом, можно утверждать, что алфавит - это конечное множество различных символов. Алфавит, как любое множество, задается перечислением его элементов. Итак, объекты реального мира можно изображать словами в различных алфавитах. Это позволяет считать, что объектами работы алгоритмов могут быть только слова. Тогда можно сформулировать следующее определение.

Алгоритм есть четкая конечная система правил для преобразования слов из некоторого алфавита в слова из этого же алфавита. Слово, к которому применяется алгоритм, называется входным. Слово, вырабатываемое в результате применения алгоритма, называется выходным. Совокупность слов, к которым применим данный алгоритм, называется областью применимости этого алгоритма.

Формальные определения алгоритма появились в 30-х - 40-х годах XX века. Можно выделить три основных типа универсальных алгоритмических моделей, различающихся исходными эвристическими соображениями относительно того, что такое алгоритм. Первый тип связывает понятие алгоритма с наиболее традиционными понятиями математики - вычислениями и числовыми функциями. Наиболее развитая и изученная модель этого типа - рекурсивные функции - является исторически первой формализацией понятия алгоритма. Эта модель основана на функциональном подходе и рассматривает понятие алгоритма с точки зрения того, что можно вычислить с его помощью.

Второй тип основан на представлении алгоритма как некоторого детерминированного устройства, способного выполнять в каждый отдельный момент некоторые примитивные операции, или инструкции. Такое представление не оставляет сомнений в однозначности алгоритма и элементарности его шагов. Основной теоретической моделью этого типа, созданной в 30-х годах, является машина Тьюринга, которая представляет собой автоматную модель, в основе которой лежит анализ процесса выполнения алгоритма как совокупности набора инструкций. Третий тип алгоритмических моделей - это преобразования слов в произвольных алфавитах, в которых элементарными операциями являются подстановки, т.е. замены части слова (подслова) другим словом. Преимущество этого типа состоит в его максимальной абстрактности и возможности применить понятие алгоритма к объектам произвольной природы. Модели второго и третьего типа довольно близки и отличаются в основном эвристическими подходами. Примерами моделей этого типа являются нормальные алгоритмы Маркова и канонические системы Поста.

Понятие алфавитного оператора является наиболее общим. К нему фактически сводятся любые процессы преобразования информации. К изучению алфавитных операторов фактически сводится теория любых преобразователей информации. Основой теории алфавитных операторов являются способы их задания. Если область определения алфавитного оператора конечна, то оператор может быть задан простой таблицей соответствия. В случае бесконечной области определения алфавитного оператора он задается системой правил, позволяющей за конечное число шагов найти выходное слово, соответствующее заданному входному слову. Алфавитные операторы, задаваемые с помощью конечной системы правил, называются алгоритмами.

Алгоритмы, в соответствии с которыми решение поставленных задач сводится к арифметическим действиям, называются численными алгоритмами.

Алгоритмы, в соответствии с которыми решение поставленных задач сводится к логическим действиям, называются логическими алгоритмами.

Различают однозначные и многозначные алфавитные операторы.

Под однозначным алфавитным оператором понимается такой алфавитный оператор, который каждому входному слову ставит в соответствие не более одного выходного слова. Под многозначным алфавитным оператором понимается такой алфавитный оператор, который каждому входному слову ставит в соответствие более одного выходного слова.

Алфавитный оператор, не сопоставляющий данному входному слову  $a_i$  никакого выходного слова  $b_j$  (в том числе и пустого), не определен на этом слове.

Совокупность всех слов, на которых алфавитный оператор определен, называется областью его определения. Два алфавитных оператора считаются равными, если равны соответствующие им алфавитные операторы и совпадает система правил, задающих действие этих алгоритмов на выходные слова. Два алгоритма считаются эквивалентными, если у них совпадают алфавитные операторы, но не совпадают способы их задания (система правил). Обычно в теории алгоритмов рассматриваются лишь такие алгоритмы, которым соответствуют однозначные

### **Задания для самостоятельной работы**

Во всех заданиях необходимо разработать схемы алгоритмов и проанализировать процесс реализации алгоритма, т.е. последовательность шагов, которая будет порождена при применении алгоритма к конкретным исходным данным.

1. Разработать словесные алгоритмы вычитания из некоторого числа  $A$  последовательности  $n$  чисел  $b_1, b_2, \dots, b_n$ :

а) алгоритм вычисления по формуле:

$$C = ((A - b_1) - b_2) - \dots - b_n$$

б) алгоритм вычисления по формуле:

$$C = A - \sum_{i=1}^n b_i .$$

2. Разработать алгоритм вычисления по формуле:

$$C_k = \sum_{i=1}^n a_i - b_k (1 \leq i \leq n, 1 \leq k \leq m) .$$

3. Разработать алгоритм вычисления по формуле:

$$C_i = \prod_{k=1}^m a_k \times b_i (1 \leq i \leq m, 1 \leq k \leq m).$$

4. Разработать алгоритм поиска максимального (минимального) элемента из последовательности, заданной в виде одномерного массива  $A = \{a_1, a_2, \dots, a_k\}$ .
5. Разработать алгоритм определения количества одинаковых чисел в последовательности, заданной в виде одномерного массива  $A = \{a_1, a_2, \dots, a_k\}$ .
6. Разработать алгоритм подсчета количества одинаковых элементов в матрице  $B$  размерностью  $n \times m$ . Размерность матрицы вводится с клавиатуры.
7. Разработать алгоритм умножения матрицы на вектор.
8. Разработать алгоритм умножения матрицы на матрицу.
9. Разработать алгоритм транспонирования матрицы.
10. Разработать алгоритм, реализующий операцию объединения следующих двух множеств:  $A = \{a_1, a_2, \dots, a_k\}$  и  $B = \{b_1, b_2\}$ .

## РЕКУРСИВНЫЕ ФУНКЦИИ

### Общие сведения

Первой алгоритмической системой была система, основанная на использовании конструктивно определяемых арифметических (целочисленных) функций, получивших специальное название рекурсивных функций. Рекурсией называется способ задания функции, при котором значение определяемой функции для произвольных значений аргументов выражается известным образом через значения определяемой функции для меньших значений аргументов. Применение рекурсивных функций в теории алгоритмов основано на идее нумерации слов в произвольном алфавите последовательными натуральными числами. Наиболее простотакую нумерацию можно осуществить, располагая слова в порядке возрастания их длин, а слова, имеющие одинаковую длину, - в произвольном порядке. После нумерации входных и выходных слов в произвольном алфавитном операторе этот оператор превращается в функцию  $y = f(x)$ , в которой аргумент  $x$  и функция  $y$  принимают неотрицательные целочисленные значения. Функция  $f(x)$  может быть определена не для всех значений  $x$ , а лишь для тех, которые составляют область определения этой функции.

### Понятие простейших функций

Числовые функции, значение которых можно установить посредством некоторого алгоритма, называются вычислимыми функциями. Для того чтобы описать класс функций с помощью рекурсивных определений, рассмотрим набор простейших функций:

1)  $Z(x_1, x_2, \dots, x_n) = 0$  - нуль-функция, которая определена для всех неотрицательных значений аргумента;

2)  $s(x) = x + 1$  - функция непосредственного следования, также определенная для всех целых неотрицательных значений своего аргумента;

3)  $I(x_1, x_2, \dots, x_m, \dots, x_n) = x_m$  - функция выбора (тождества), повторяющая значения своих аргументов. Используя простейшие функции в качестве исходных функций, можно с помощью небольшого числа общих конструктивных приемов строить сложные арифметические функции. В теории рекурсивных функций особо важное значение имеют три операции: суперпозиции, примитивной рекурсии и минимизации.

### Оператор суперпозиции

Оператором суперпозиции  $S$  называется подстановка в функцию от  $m$  переменных  $m$  функций от  $n$  одних и тех же переменных. Она дает новую функцию от  $n$  переменных. Например, из функций  $f(x_1, x_2, \dots, x_m)$ ,  $f_1(x_1, x_2, \dots, x_n)$ ,  $f_2(x_1, x_2, \dots, x_n)$ ,  $\dots$ ,  $f_m(x_1, x_2, \dots, x_n)$  можно получить новую функцию:  $S_{m+1}(f, f_1, f_2, \dots, f_m) = g(x_1, x_2, \dots, x_n) = f(f_1(x_1, x_2, \dots, x_n), f_2(x_1, x_2, \dots, x_n), \dots, f_m(x_1, \dots, x_n))$ . (1)

В операции суперпозиции  $S_{m+1}$  индекс сверху указывает на число функций. Таким

образом, при помощи оператора суперпозиции и функции выбора можно выразить любую подстановку функции в функцию. Например, осуществляя операцию суперпозиции функций  $f(x) = 0$  и  $g(x) = x+1$ , получим функцию:  $h(x) = g(f(x)) = 0 + 1 = 1$ .

При суперпозиции функции  $g(x)$  с этой же функцией получим функцию  $h(x) = g(g(x)) = x + 2$ . Используя подстановку и функции тождества, можно переставлять и отождествлять аргументы в функции:

$$f(x_2, x_1, x_3, \dots, x_n) = f(I_{22}(x_1, x_2), I_{12}(x_1, x_2), x_3, \dots, x_n);$$

$$f(x_1, x_1, x_3, \dots, x_n) = f(I_{12}(x_1, x_2), I_{12}(x_1, x_2), x_3, \dots, x_n).$$

Таким образом, если заданы функции тождества и операторы суперпозиции, то можно считать заданными всевозможные операторы подстановки функций в функции, а также переименования, перестановки и отождествления переменных.

Оператор примитивной рекурсии

Оператор примитивной рекурсии  $R_n$  позволяет определить  $(n+1)$ - местную функцию  $f$  по двум заданным функциям, одна из которых является  $n$ - местной функцией  $g$ , а другая  $(n+2)$ - местной функцией  $h$ .

Функция  $f(x_1, x_2, \dots, x_n, y)$  получается оператором примитивной рекурсии из функций  $g(x_1, x_2, \dots, x_n)$  и функции  $h(x_1, x_2, \dots, x_n, y, z)$ , если:

$$f(x_1, x_2, \dots, x_n, 0) = g(x_1, x_2, \dots, x_n); \quad (2)$$

$$f(x_1, x_2, \dots, x_n, y+1) = h(x_1, x_2, \dots, x_n, y, f(x_1, x_2, \dots, x_n, y)).$$

Приведенная пара равенств (2) называется схемой примитивной рекурсии. Для понимания операции примитивной рекурсии необходимо заметить, что всякую функцию от меньшего числа аргументов можно рассматривать как функцию от большего числа аргументов. Существенным в операторе примитивной рекурсии является то, что независимо от числа переменных в  $f$  рекурсия ведется только по одной переменной  $y$ . Остальные  $n$  переменных  $x_1, x_2, \dots, x_n$  на момент применения схемы (2) зафиксированы и играют роль параметров.

Оператор минимизации

Отношение  $P(x_1, x_2, \dots, x_n)$  называется примитивно-рекурсивным, если примитивно рекурсивна его характеристическая функция:

$$\chi(x_1, x_2, \dots, x_n) = \begin{cases} 1, & \text{если } P(x_1, x_2, \dots, x_n) \text{ - "истина",} \\ 0, & \text{если } P(x_1, x_2, \dots, x_n) \text{ - "ложь".} \end{cases}$$

Предикат называется примитивно-рекурсивным, если его характеристическая функция примитивно-рекурсивна.

Функция  $f(x_1, x_2, \dots, x_n)$  получается посредством оператора минимизации из предиката  $P(x_1, x_2, \dots, x_n, z)$ , если в любой точке значением функции  $f(x_1, x_2, \dots, x_n)$  является минимальное значение  $z$ , обращающее предикат  $P(x_1, x_2, \dots, x_n, z)$  в значение «истина»:

$$f(x_1, x_2, \dots, x_n) = \mu z (P(x_1, x_2, \dots, x_n, z)),$$

где  $\mu z$  – оператор минимизации.

#### 2.2.4. Ограниченный оператор минимизации

Функция  $f(x_1, x_2, \dots, x_n)$  получается ограниченным оператором минимизации из предиката  $P(x_1, x_2, \dots, x_n, z)$  и функции  $U(x_1, x_2, \dots, x_n)$ , если в любой точке значение этой функции определено следующим образом:

1) для любого  $z < U(x_1, x_2, \dots, x_n)$  такого, что  $P(x_1, x_2, \dots, x_n, z) = \text{"истина"}$ , значение функции  $f(x_1, x_2, \dots, x_n) = \mu z (P(x_1, \dots, x_n, z))$ ,

2) не для любого  $z < U(x_1, x_2, \dots, x_n)$  такого, что  $P(x_1, x_2, \dots, x_n, z) = \text{"истина"}$ , значение функции  $f(x_1, x_2, \dots, x_n) = U(x_1, x_2, \dots, x_n)$ .

Данное определение записывается следующим образом:  $f(x_1, x_2, \dots, x_n) = \mu z < U(x) (P(x_1, x_2, \dots, x_n, z))$ . Ограничение  $z$  в ограниченном операторе минимизации дает гарантию окончания вычислений, поскольку оно оценивает сверху число вычислений предиката  $P$ . Возможность оценить сверху количество вычислений является существенной особенностью примитивно-рекурсивных функций.

### Примитивно-рекурсивные и частично-рекурсивные функции

Большинство вычислимых функций относится к классу примитивно-рекурсивных функций. Функция называется примитивно-рекурсивной, если она может быть получена из простейших функций с помощью конечного числа операторов суперпозиции и примитивной рекурсии.

Если некоторые функции являются примитивно-рекурсивными, то в результате применения к ним операторов суперпозиции или примитивной рекурсии можно получить новые примитивно-рекурсивные функции. Существует три возможности доказательства того, что функция является примитивно-рекурсивной:

- а) показать, что заданная функция является простейшей;
- б) показать, что заданная функция построена с помощью оператора суперпозиции;
- в) показать, что заданная функция построена с помощью оператора примитивной рекурсии.

Тем не менее примитивно-рекурсивные функции не охватывают все вычислимые функции, которые могут быть определены конструктивно. При построении этих функций могут использоваться другие операции. Функция называется частично-рекурсивной, если она может быть получена из простейших функций с помощью конечного числа операторов суперпозиции, примитивной рекурсии и минимизации. Указанные операции могут быть выполнены в любой последовательности и любое конечное число раз. Если такие функции оказываются всюду определенными, то они называются общерекурсивными функциями. Частично-рекурсивные функции вычислимы путем определенной процедуры механического характера. Практически понятием частично-рекурсивных функций пользуются для доказательства алгоритмической разрешимости или неразрешимости проблем. Приведем тезис Черча, который связывает понятие алгоритма и строгое математическое понятие частично-рекурсивной функции. Тезис Черча: всякий алгоритм может быть реализован частично-рекурсивной функцией. Утверждение о несуществовании частично-рекурсивной функции эквивалентно несуществованию алгоритма решения соответствующей задачи.

### Типы рекурсивных алгоритмов

Эффективность разработки рекурсивного алгоритма определяется наличием некоторых условий:

- 1) если исходные данные имеют рекурсивную структуру, то процедуры анализа таких структур будут более эффективны, если они сами рекурсивны;
- 2) если алгоритм обработки некоторого набора данных построить, разбивая данные на части и обрабатывая в отдельности каждую часть, то из частичных решений можно получить общее;
- 3) если по условию задачи необходимо выбрать оптимальный вариант из некоторого множества решений, то искомое решение может быть найдено через конечное число шагов. При этом на каждом шаге удаляется часть информации, и далее задача решается на меньшем объеме данных. Поиск решения завершается после окончания данных либо при нахождении искомого решения на текущем наборе данных.

### Методика решения задач

Использование оператора примитивной рекурсии

Пример 1. Доказать примитивную рекурсивность функции  $f(x, y) = x + y$ .

Решение. Рассмотрим способ рекурсивного определения данной функции:  $f(x, 0) = x$ ,  $f(x, y+1) = x + y + 1 = f(x, y) + 1$ . Чтобы показать соответствие данной рекурсивной схемы схеме примитивной рекурсии, воспользуемся функциями выбора и следования:  $f(x, 0) = x = I(x)$ ,  $f(x, y+1) = f(x, y) + 1 = S(f(x, y)) = S(I_3(x, y, f(x, y)))$ .

Пример 2. Доказать примитивную рекурсивность функции  $f(x, y) = x \cdot y$ .

Решение. Рассмотрим способ рекурсивного определения данной функции:

$f(x, 0) = 0$ ,  $f(x, y+1) = x \cdot (y + 1) = x \cdot y + x = f(x, y) + x$ , из которого следует, что  $Z(x) = x$ .  
 • 0. Обозначим  $x \cdot y = p(x, y)$ , тогда:  
 $p(x, 0) = Z(x)$ ;  $p(x, y+1) = p(x, y) + x = S(p(x, y), x) = S(I(x, y, p(x, y)), I^3(x, y, p(x, y)))$ .

Пример 3. Доказать примитивную рекурсивность функции

$$Sg(x) = \begin{cases} 0, & \text{если } x = 0, \\ 1, & \text{если } x \neq 0. \end{cases}$$

Решение. Рассмотрим способ рекурсивного определения данной функции:  $Sg(0) = 0 = Z(x)$ ,  $Sg(x+1) = Z(x) + 1 = 1 = S(Z(x))$ .

Пример 4. Доказать примитивную рекурсивность функции  $f(x) = 2x$ .

Решение. Рассмотрим способ рекурсивного определения данной функции:

$$f(0) = 1 = S(Z(x)),$$

$$f(x+1) = 2 \cdot 2x = 2 \cdot f(x) = S(S(Z(x))).$$

Пример 5. Доказать примитивную рекурсивность функции  $f(x, y) = xy$

Решение. Рассмотрим способ рекурсивного определения данной функции:

$$f(x, 0) = 1 = S(Z(x));$$

$$f(x, y+1) = x^{y+1} = x^y \cdot x = f(x, y) \cdot I_1^2(x, y) = p(I_3^3(x, y, f(x, y)), I_1^3(x, y, f(x, y))).$$

Использование оператора минимизации

Пример 1. Пусть  $f(x, y) = \mu z (2 \cdot x + z = y + 1)$ , откуда предикат  $P(x, y, z) = 2 \cdot x + z = y + 1$ .

Покажем примитивную рекурсивность предиката  $P$ . Его характеристическая функция может быть представлена следующим выражением:

$$Sg(Sg((2 \cdot x + z) \div (y + 1)) + Sg((y + 1) \div (2 \cdot x + z))).$$

Найдем значение функции в точке (1, 5):

$$\text{При } z = 0: P(1, 5, 0) = 2 \cdot 1 + 0 = 5 + 1 - \text{"ложь"},$$

$$z = 1: P(1, 5, 1) = 2 \cdot 1 + 1 = 5 + 1 - \text{"ложь"},$$

$$z = 2: P(1, 5, 2) = 2 \cdot 1 + 2 = 5 + 1 - \text{"ложь"},$$

$$z = 3: P(1, 5, 3) = 2 \cdot 1 + 3 = 5 + 1 - \text{"ложь"},$$

$$z = 4: P(1, 5, 4) = 2 \cdot 1 + 4 = 5 + 1 - \text{"истина"}.$$

Таким образом, минимальное значение переменной  $z$ , обращающее предикат в "истину", дает значение функции в точке (1, 5), и оно равно 4.

### Задания для самостоятельной работы

Доказать примитивную рекурсивность следующих функций.

1. Усеченное вычитание

$$x_1 \div x_2 = \begin{cases} x_1 - x_2, & \text{если } x_1 > x_2, \\ 0, & \text{если } x_1 < x_2. \end{cases}$$

2. Функция знака

$$Sg(x) = \begin{cases} 0, & \text{если } x = 0, \\ 1, & \text{если } x \neq 0. \end{cases}$$

3. Нахождение минимального значения из двух заданных чисел  $f(x, y) = \min(x, y)$ .

4. Нахождение максимального значения из двух заданных чисел  $f(x, y) = \max(x, y)$ .

5. Усеченное вычитание

$$x \div 1 = \begin{cases} x - 1, & \text{если } x > 1, \\ 0, & \text{если } x < 1. \end{cases}$$

6. Функция  $r(x, y)$  - остаток от деления  $y$  на  $x$ .
7. Функция  $q(x, y)$  - частное от деления  $y$  на  $x$ .
8. Доказать, что предикат  $Pd_n(x)$  "x делится на n" примитивно-рекурсивен для любого  $n$ .
9. Доказать, что предикат  $Pd_{n,m}(x)$  "x делится на n и m" примитивно-рекурсивен для любых  $n$  и  $m$ .
10. Доказать, что отношение  $x_1 > x_2$  примитивно-рекурсивно.
11. Доказать, что предикат  $f(x) = g(x)$  примитивно-рекурсивен, если функции  $f(x)$  и  $g(x)$  примитивно-рекурсивны.
12. Дано множество слов одинаковой длины, причем первые два слова выделены. Построить цепь от первого выделенного слова ко второму так, чтобы все слова этой цепи были только из заданного множества. Соседние слова построенной цепи должны отличаться только одной буквой.
13. Дано множество слов. Построить из них кроссворд заданной конфигурации (число слов может быть больше требуемого количества для заполнения кроссворда).
14. Грани кубика разбиты на клетки  $5 \times 5$ . Каждая из клеток выкрашена в белый или синий цвет. Переход из клетки в клетку допускается только через общую сторону при условии совпадения цветов этих клеток. Построить маршрут перехода из одной заданной клетки в другую заданную клетку этого же цвета.
15. Имеется клетчатая ткань размером  $N \times N$  клеток. Разрезать ее на  $M$  квадратных частей, не нарушая целостности клеток.

## МАШИНЫ ТЬЮРИНГА

### Общие сведения

Одним из первых формальных определений алгоритма было определение английского математика А. Тьюринга, который в 1936 г. описал схему некоторой гипотетической (абстрактной) машины и формализовал правила выполнения действий при помощи описания работы этой машины. Машина Тьюринга является абстракцией, которую нельзя реализовать практически. Поэтому алгоритмы для машины Тьюринга должны выполняться другими средствами.

Основным следствием формализации алгоритмов с использованием машины Тьюринга является возможность доказательства существования или несуществования алгоритмов решения различных задач. Описывая различные алгоритмы для машин Тьюринга и доказывая реализуемость всевозможных композиций алгоритмов, Тьюринг убедительно показал разнообразие возможностей предложенной им конструкции, что позволило ему выступить со следующим тезисом: "Всякий алгоритм может быть реализован соответствующей машиной Тьюринга".

Доказать тезис Тьюринга нельзя, так как в его формулировке не определено понятие "всякий алгоритм". Его можно только обосновать, представляя различные алгоритмы в виде машин Тьюринга. Было доказано, что класс функций, вычислимых на этих машинах, совпадает с классом частично рекурсивных функций.

### Неформальное определение машины Тьюринга

Машина Тьюринга представляет собой автомат, имеющий бесконечную в обе стороны ленту, считывающую головку и управляющее устройство. Управляющее устройство может находиться в одном из состояний, образующих конечное множество  $Q = \{q_0, q_1, \dots, q_n\}$ . Множество  $Q$  называют внутренним алфавитом машины Тьюринга.

Принципиальное отличие машины Тьюринга от вычислительных машин состоит в том, что ее запоминающее устройство представляет собой бесконечную ленту, из-за

которой невозможна ее физическая реализация. Лента разделена на ячейки, в каждой из которых может быть записан один из символов конечного алфавита  $A = \{a_0, a_1, \dots, a_m\}$ , который называют входным алфавитом машины Тьюринга. Во время функционирования машины Тьюринга может быть заполнено конечное число ячеек. Считывающая головка в каждый момент времени обозревает ячейку ленты, в зависимости от символа в этой ячейке и состояния управляющего устройства записывает в ячейку новый символ или оставляет его без изменения, сдвигается на ячейку влево или вправо или остается на месте. При этом управляющее устройство переходит в новое состояние или остается в старом. Среди состояний управляющего устройства выделены начальное состояние  $q_0$  и заключительное состояние  $q_z$ .

Таким образом, за один такт работы машина Тьюринга может считать символ, записать вместо него новый или оставить его без изменения и сдвинуть головку на одну ячейку влево или вправо или оставить ее на месте.

### Формальное определение машины Тьюринга

Алфавит  $A$  – это некоторое конечное множество символов.

Цепочкой над алфавитом называется последовательность символов из этого алфавита. Длина цепочки  $x$  представляет собой число символов в цепочке и обозначается  $|x|$ . Длина пустой цепочки равна нулю.

Конкатенацией цепочек  $X$  и  $Y$  называют цепочку, полученную приписыванием символов цепочки  $Y$  справа к цепочке  $X$ .

Машиной Тьюринга называется семерка вида  $T = (Q, A, \delta, p_0, p_z, a_0, a_1)$ , где  $Q$  – конечное множество состояний, в которых может находиться управляющее устройство;

$A$  – входной алфавит;

$\delta$  – функция переходов,  $\delta = Q \cdot A \rightarrow Q \cdot A \cdot S$ , где

$S = \{R, L, E\}$  – направления сдвига;

$p_0$  – начальное состояние;  $p_0 \in Q$ ;

$p_z$  – заключительное состояние;  $p_z \in Q$ ;

$a_0$  – символ для обозначения пустой ячейки,  $a_0 \in A$ ;

$a_1$  – специальный символ – разделитель цепочек на ленте,  $a_1 \in A$ .

Командой машины Тьюринга называется элемент функции переходов  $qa \rightarrow pbr$ , где  $q$  и  $r \in Q$ ;  $a$  и  $b \in A$ ;  $r \in S$ . Каждая команда описывает один такт работы машины Тьюринга. Конфигурация машины Тьюринга представляется следующим образом:  $t = \langle CqaB \rangle$ , где  $C$  – цепочка, находящаяся слева от считывающей головки;

$q$  – состояние машины Тьюринга;

$a$  – символ, находящийся под головкой машины Тьюринга;

$B$  – цепочка, находящаяся справа от головки машины Тьюринга.

Конфигурация  $\langle CqaB \rangle$  непосредственно переходит в конфигурацию  $\langle CnqnanBn \rangle$ , если новая конфигурация получена в результате применения одной команды к исходной конфигурации. Обозначим непосредственный переход из одной конфигурации в другую следующим образом:  $\langle CqaB \rangle \Rightarrow \langle CnqnanBn \rangle$ .

Конфигурация, содержащая начальное состояние, называется начальной, а содержащая заключительное состояние – заключительной. Если цепочка  $C$  в конфигурации пустая, то начальная и заключительная конфигурации называются стандартными. Машина Тьюринга перерабатывает цепочку  $x$  в цепочку  $y$ , если, действуя из начальной конфигурации и имея на ленте цепочку  $x$ , машина Тьюринга переходит в заключительную конфигурацию, имея на ленте цепочку  $y$ . Если начальная и заключительная конфигурации являются стандартными, то процесс переработки  $x$  в  $y$  является правильной переработкой.

### Способы представления машины Тьюринга

Существует три способа представления машины Тьюринга: совокупностью команд, в виде графа, в виде таблицы соответствия.

Представление машины Тьюринга совокупностью команд  
Совокупность всех команд, которые может выполнять машина, называется ее программой.

Машина Тьюринга считается заданной, если заданы ее внешний и внутренний алфавиты, программа, начальная конфигурация и указано, какие из символов обозначают пустую ячейку и заключительное состояние. Чтобы записать совокупность команд, нужно воспользоваться следующими правилами:

- 1) начальному шагу алгоритма ставится в соответствие  $q_0$
- 2) циклы реализуются так, что последнее действие цикла должно соответствовать переходу в то состояние, которое соответствует началу цикла;
- 3) соседним шагам алгоритма соответствует переход в смежные состояния, которые соответствуют этим пунктам;
- 4) последний шаг алгоритма вызывает переход в заключительное состояние.

В качестве примера рассмотрим совокупность команд машины Тьюринга, которая инвертирует входную цепочку, записанную с использованием нулей и единиц.

Пусть алфавит машины Тьюринга задан множеством  $A = \{0, 1, \varepsilon\}$ , где символ  $\varepsilon$  соответствует пустой ячейке, а число состояний устройства управления задано в виде множества  $Q = \{q_0, q_1, q_z\}$

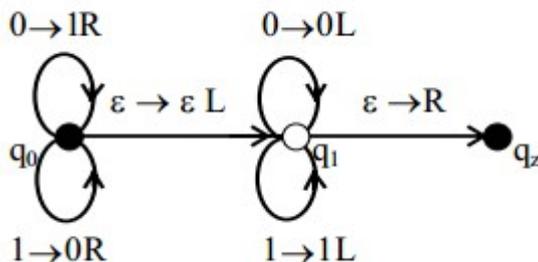
Если, например, начальная конфигурация имеет вид  $q_0110011$ , то конечная конфигурация после завершения операции инвертирования должна иметь вид  $q_z001100$ . Для решения задачи машиной будет порождена следующая последовательность команд:

$q_01 \rightarrow q_00R, q_10 \rightarrow q_10L,$   
 $q_00 \rightarrow q_01R, q_11 \rightarrow q_11L,$   
 $q_0\varepsilon \rightarrow q_1\varepsilon L, q_1\varepsilon \rightarrow q_z\varepsilon R.$

В стандартной начальной конфигурации головка стоит над первым символом слева, и устройство управления находится в начальном состоянии. На следующем такте машина Тьюринга, не меняя своего состояния, заменяет символ 0 на 1 или 1 на 0 и сдвигается вправо на один символ. После просмотра всей цепочки под головкой оказывается символ, указывающий на пустую ячейку. В этом случае машина Тьюринга переходит в новое состояние и сдвигается влево на один символ. На последующих тактах управляющее устройство не меняет своего состояния, оставляет без изменения символ под головкой и перемещается влево до тех пор, пока не встретит пустую ячейку. Встретив пустую ячейку, машина Тьюринга переходит в заключительное состояние и перемещается вправо на один символ, переходя в стандартную заключительную конфигурацию.

Представление машины Тьюринга графом

При представлении машины Тьюринга посредством графа необходимо каждому состоянию поставить в соответствие вершину графа, а каждой команде - помеченную дугу. Машина Тьюринга из рассмотренного примера инвертирования цепочки, состоящей из символов 0 и 1, будет представлена в виде графа следующим образом:



Начальная и конечная вершины графа обычно выделяются; на рисунке они отмечены черным кружком. Если на очередном такте работы машины Тьюринга символ на

ленте не изменяется, то в правой части команды его можно не писать ( $\epsilon$  на ребре в состоянии  $q_z$ ).

### Представление машины Тьюринга таблицей соответствия

При представлении машины Тьюринга данным способом составляется таблица, в которой каждому состоянию соответствует строка, а каждому символу из входного алфавита - столбец. В клетках таблицы на пересечении строки и столбца будет находиться действие(или правая часть команды).

Таблица соответствия для задания машины Тьюринга из рассмотренного примера будет выглядеть следующим образом:

	0	1	$\epsilon$
$q_0$	$q_0 1R$	$Q_0 0R$	$q_1 \epsilon L$
$q_1$	$q_1 0L$	$Q_1 1L$	$q_z \epsilon L$

Инвертирование входной цепочки можно выполнить программой машины Тьюринга, приведенной в таблице соответствия. Эта программа включает шесть команд.

### Вычислимые функции

Говорят, что машина Тьюринга вычисляет функцию  $f(x_1, x_2, \dots, x_n)$ , если выполняются следующие условия:

- 1) для любых  $x_1, x_2, \dots, x_n$ , принадлежащих области определения функции, машина Тьюринга из начальной конфигурации, имея на ленте представление аргументов, переходит в заключительную конфигурацию, имея на ленте результат (представление функции);
- 2) для любых  $x_1, x_2, \dots, x_n$ , не принадлежащих области определения функции, машина Тьюринга из начальной конфигурации работает бесконечно.

Если начальная и заключительная конфигурации машины Тьюринга являются стандартными, то говорят, что машина Тьюринга правильно вычисляет функцию  $f$ . Функция называется вычислимой по Тьюрингу, если существует машина Тьюринга, вычисляющая ее.

Для того чтобы доказать вычислимость функции, а в дальнейшем и существование алгоритма, необходимо построить машину Тьюринга, реализация которой на практике зачастую представляет собой трудоемкую задачу. В связи с этим возникает необходимость разбиения алгоритма на отдельные задачи, каждая из которых будет решаться отдельной машиной Тьюринга. Если объединить программы этих машин, то получится новая программа, позволяющая решить исходную задачу.

Машины Тьюринга могут вычислять искомую функцию с восстановлением и без восстановления. Вычисление функции с восстановлением означает работу машины Тьюринга с сохранением исходных данных:

$$p_0 1X_1^* \dots * 1X_n \Rightarrow^* p_z 1f(X_1, X_2, \dots, X_n) * 1X_1^* \dots * 1X_n.$$

Приведенное определение позволяет получать на ленте сначала результат, а затем исходные данные. В отдельных случаях удобно сделать наоборот:

$$p_0 1X_1^* \dots * 1X_n \Rightarrow^* 1X_1^* \dots * 1X_n * p_z 1f(X_1, X_2, \dots, X_n).$$

Вычисление функции без восстановления означает работу машины Тьюринга без сохранения исходных данных:

$$p_0 1X_1^* \dots * 1X_n \Rightarrow^* p_z 1f(X_1, X_2, \dots, X_n).$$

Справедливо утверждение, что всякая правильно вычислимая функция правильно

вычислима с восстановлением.

Операции над машинами Тьюринга

1. Композиция машин Тьюринга. Пусть машины  $T_1$  и  $T_2$  имеют программы  $P_1$  и  $P_2$ . Предположим, что внутренние алфавиты этих машин не пересекаются; пусть  $qz_1$  - заключительное состояние машины  $T_1$ , а  $q_02$  - начальное состояние машины  $T_2$ . Заменяем всюду в программе  $P_1$  заключительное состояние  $qz_1$  на начальное состояние  $q_02$  машины  $T_2$  и полученную программу объединим с программой  $P_2$ . Новая программа  $P$  определяет машину  $T$ , называемую композицией машин  $T_1$  и  $T_2$  по паре состояний  $(qz_1, q_02)$ .

Композиция машин может быть обозначена  $T_1 \cdot T_2$  или  $T_1 T_2$ . Более подробно композиция машин записывается следующим образом:

$$T = T(T_1, T_2, (qz_1, q_02)),$$

$$\text{где } T_1 = (Q_1, A_1, \delta_1, p_01, pz_1, a_01, a_11),$$

$$T_2 = (Q_2, A_2, \delta_2, p_02, pz_2, a_02, a_12).$$

Пусть  $a_01 = a_02 = a_0$  и  $a_11 = a_12 = a_1$ . Внешний алфавит композиции  $T_1 T_2$  является объединением внешних алфавитов машин  $T_1$  и  $T_2$ :

$$T = (Q_1 \cup Q_2 \setminus \{pz_1\}, A_1 \cup A_2, |(\delta_1, \cup \delta_2), p_01, pz_2, a_0, a_1),$$

$$pz_1.$$

Операция композиции, выполняемая над алгоритмами, позволяет получать новые, более сложные алгоритмы из ранее известных простых алгоритмов.

2. Итерация машины Тьюринга. Эта операция применима только к одной машине. Пусть  $qz$  - заключительное состояние машины  $T$ , а  $qn$  - какое-либо состояние машины  $T$ , не являющееся заключительным. Заменяем всюду в программе  $P$  машины  $T$  состояние  $qz$  на  $qn$ . Полученная программа определяет новую машину  $T'(qz, qn)$ , которая называется итерацией машины  $T$  по паре состояний  $(qz, qn)$ . Если машина Тьюринга имеет одно заключительное состояние, то после выполнения итерации получается машина, не имеющая заключительного состояния.

3. Разветвление машин Тьюринга. Пусть машины Тьюринга  $T_1$ ,  $T_2$  и  $T_3$  задаются программами  $P_1$ ,  $P_2$  и  $P_3$  соответственно. Считаем, что внутренние алфавиты этих машин попарно не пересекаются. Пусть  $qz_{11}$  и  $qz_{12}$  - какие-либо различные заключительные состояния машины  $T_1$ . Заменяем всюду в программе  $P_1$  состояние  $qz_{11}$  начальным состоянием  $q_02$  машины  $T_2$ , а состояние  $qz_{12}$  начальным состоянием  $q_03$  машины  $T_3$ . Затем новую программу объединим с программами  $P_2$  и  $P_3$ . Получим программу  $P$ , задающую машину Тьюринга и обозначаемую:

$$T = T(T_1, (qz_{11}, q_02), T_2, (qz_{12}, q_03), T_3).$$

Машина  $T$  называется разветвлением машин  $T_2$  и  $T_3$ , управляемым машиной  $T_1$ .

Примеры построения машин Тьюринга

Пример 1. Построить машину Тьюринга, которая правильно вычисляет функцию  $f(x) = x+1$  по правилам двоичного сложения.

Решение. Исходя из формулировки задачи, требующей вычислить функцию по правилам сложения в двоичной системе сложения, выберем входной алфавит:

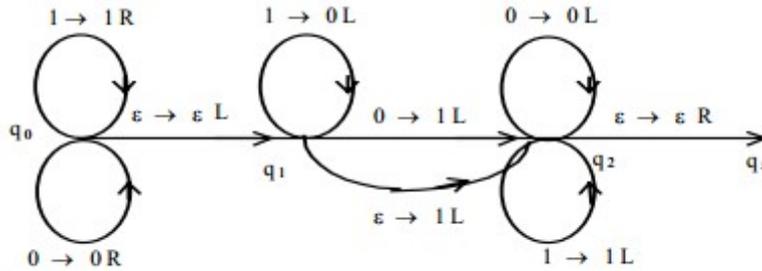
$$A = \{0, 1, \varepsilon\}.$$

Представим машины Тьюринга таблицей соответствия и графом.

Таблица соответствия:

	0	1	$\varepsilon$
$q_0$	$q_0 0R$	$q_0 1R$	$Q_1 \varepsilon L$
$q_1$	$q_2 1L$	$q_1 0L$	$q_2 1L$
$q_2$	$q_2 0L$	$Q_2 1L$	$q_z \varepsilon R$

Представление машины Тьюринга в виде графа:



Запишем программу построенной машины Тьюринга для случая, когда входная цепочка на ленте равна двоичному числу 111. Слева от каждой команды приведем представление входной цепочки на ленте до выполнения данной команды. Символ, который находится под головкой, будем помечать подчеркиванием.

- 1)  $q_0 1 \rightarrow q_0 1R \ \underline{\varepsilon} 11 \varepsilon$  6)  $q_1 1 \rightarrow q_1 0L \ \varepsilon 110 \varepsilon$
- 2)  $q_0 1 \rightarrow q_0 1R \ \varepsilon 111 \varepsilon$  7)  $q_1 1 \rightarrow q_1 0L \ \varepsilon 100 \varepsilon$
- 3)  $q_0 1 \rightarrow q_0 1R \ \varepsilon 111 \varepsilon$  8)  $q_1 \varepsilon \rightarrow q_2 1L \ \varepsilon 000 \varepsilon$
- 4)  $q_0 \varepsilon \rightarrow q_1 \varepsilon L \ \varepsilon 111 \varepsilon$  9)  $q_2 \varepsilon \rightarrow q_z \varepsilon R \ \varepsilon 1000 \varepsilon$
- 5)  $q_1 1 \rightarrow q_1 0L \ \varepsilon 111 \varepsilon$

Из примера видно, что машина Тьюринга из стандартной начальной конфигурации, имея на ленте аргумент 111, выполнив совокупность команд 1-9, перешла в стандартное заключительное состояние, имея на ленте результат 1000. Действительно:  $1112 + 12 = 10002$ .

Пример 2. Построить машину Тьюринга, выполняющую операцию  $(x \div 2)$  и имеющую входной алфавит  $A = \{0, 1, \varepsilon\}$ .

Таблица соответствия данной машины Тьюринга будет иметь следующий вид:

	0	1	$\varepsilon$		0	1	$\varepsilon$
$q_0$	$q_0 0R$	$q_0 1R$	$q_1 \varepsilon L$	$q_2$	$q_3 1L$	$q_2 0L$	$q_3 1L$
$q_1$	$q_3 \varepsilon L$	$q_2 \varepsilon L$		$q_3$	$q_2 0L$	$q_2 1L$	$q_z \varepsilon R$

Операция  $(x \div 2)$  реализована сдвигом цепочки вправо на 1 разряд.

Пример 3. Построить машину Тьюринга, которая выполняет копирование заданного аргумента. Выберем входной алфавит  $A = \{0, 1, \varepsilon\}$ . Представим данную машину таблицей соответствия:

	0	1	*	$\varepsilon$
$q_0$	$q_11L$	$q_10R$	$q_0^*L$	$q_z\varepsilon R$
$q_1$		$q_11R$	$q_2^*R$	$q_2^*R$
$q_2$		$q_21R$		$q_31L$
$q_3$	$q_00R$	$q_31L$	$q_3^*L$	

Запишем программу построенной машины для заданной входной цепочки на ленте, равной двоичному числу 111. Слева от каждой команды приведем представление входной цепочки на ленте до выполнения данной команды. Символ, который находится под головкой, будем помечать подчеркиванием.

- |   |   |
|---|---|
| 1) $q_01 \rightarrow q_10R \ \underline{\varepsilon}111\varepsilon$   | 17) $q_31 \rightarrow q_31L \ \varepsilon001^*11\varepsilon$            |
| 2) $q_11 \rightarrow q_11R \ \varepsilon011\varepsilon$               | 18) $q_30 \rightarrow q_00R \ \varepsilon001^*11\varepsilon$            |
| 3) $q_11 \rightarrow q_11R \ \varepsilon011\varepsilon$               | 19) $q_01 \rightarrow q_10R \ \varepsilon000^*11\varepsilon$            |
| 4) $q_1\varepsilon \rightarrow q_2^*R \ \varepsilon011\varepsilon$    | 20) $q_1^* \rightarrow q_2^*R \ \varepsilon000^*11\varepsilon$          |
| 5) $q_2\varepsilon \rightarrow q_31L \ \varepsilon011^*\varepsilon$   | 21) $q_21 \rightarrow q_21R \ \varepsilon000^*11\varepsilon$            |
| 6) $q_3^* \rightarrow q_3^*L \ \varepsilon011^*1\varepsilon$          | 22) $q_21 \rightarrow q_21R \ \varepsilon000^*11\varepsilon$            |
| 7) $q_31 \rightarrow q_31L \ \varepsilon011^*1\varepsilon$            | 23) $q_2\varepsilon \rightarrow q_31L \ \varepsilon000^*11\varepsilon$  |
| 8) $q_31 \rightarrow q_31L \ \varepsilon011^*1\varepsilon$            | 24) $q_31 \rightarrow q_31L \ \varepsilon000^*111\varepsilon$           |
| 9) $q_30 \rightarrow q_00R \ \varepsilon011^*1\varepsilon$            | 25) $q_31 \rightarrow q_31L \ \varepsilon000^*111\varepsilon$           |
| 10) $q_01 \rightarrow q_10R \ \varepsilon011^*1\varepsilon$           | 26) $q_3^* \rightarrow q_3^*L \ \varepsilon000^*111\varepsilon$         |
| 11) $q_11 \rightarrow q_11R \ \varepsilon001^*1\varepsilon$           | 27) $q_30 \rightarrow q_00R \ \varepsilon000^*111\varepsilon$           |
| 12) $q_1^* \rightarrow q_2^*R \ \varepsilon001^*1\varepsilon$         | 28) $q_0^* \rightarrow q_0^*L \ \varepsilon000^*111\varepsilon$         |
| 13) $q_21 \rightarrow q_21R \ \varepsilon001^*1\varepsilon$           | 29) $q_00 \rightarrow q_01L \ \varepsilon000^*111\varepsilon$           |
| 14) $q_2\varepsilon \rightarrow q_31L \ \varepsilon001^*1\varepsilon$ | 30) $q_00 \rightarrow q_01L \ \varepsilon001^*111\varepsilon$           |
| 15) $q_31 \rightarrow q_31L \ \varepsilon001^*11\varepsilon$          | 31) $q_00 \rightarrow q_01L \ \varepsilon011^*111\varepsilon$           |
| 16) $q_3^* \rightarrow q_3^*L \ \varepsilon001^*11\varepsilon$        | 32) $q_0\varepsilon \rightarrow q_z1R \ \varepsilon111^*111\varepsilon$ |

Из примера видно, что машина Тьюринга из стандартной начальной конфигурации, имея на ленте аргумент 111 и выполнив совокупность команд 1-32, перешла в стандартное заключительное состояние, имея на ленте результат  $\varepsilon111^*111\varepsilon$ .

#### Машина Тьюринга с полулентой

В рассмотренных выше определениях машины Тьюринга использовалась бесконечная в обе стороны лента. Ограничим ленту с одной стороны и покажем, что машина Тьюринга с правой или левой полулентой эквивалентна машине Тьюринга с бесконечной лентой. Говорят, что функция, правильно вычисляемая на машине Тьюринга с обычной лентой, правильно вычислима и на машине Тьюринга с правой полулентой, т.е. для любой машины Тьюринга  $T$  существует эквивалентная ей машина с правой полулентой  $TR$

Идея доказательства этого утверждения основана на следующих предпосылках:

а) рабочая область ленты ограничена двумя маркерами: неподвижным - слева и подвижным - справа;

б) на внутренней части ограниченной области машина Тьюринга с полулентой должна работать как обычная машина Тьюринга, а при выходе на маркеры она должна освобождать рабочее пространство, для этого правый маркер может сдвигаться вправо, а при выходе левый маркер необходимо сдвинуть всю цепочку вправо;

в) полученный результат, находящийся между маркерами, в конце работы машины Тьюринга нужно сдвинуть вплотную к левому маркеру.

Машина Тьюринга может вычислять функцию с восстановлением, то есть с сохранением исходных данных. Это позволяет получить на ленте сначала результат, а затем –

исходные данные или наоборот.

Рассмотрим пример построения машины Тьюринга с правой полулентой, вычисляющей функцию  $f(x) = 2x$ . Алгоритм вычисления данной функции состоит в приписывании нуля к входной цепочке справа.

Таблица соответствия данной машины Тьюринга будет иметь следующий вид:

	<	0	1	>	$\epsilon$
$q_0$		$q_0 0R$	$q_0 1R$	$q_1 0R$	
$q_1$					$q_2 >L$
$q_2$	$q_2 < R$	$q_2 0L$	$q_2 1L$		

Здесь символ “<” обозначает левый маркер, а символ “>” - правый маркер. Машина Тьюринга, начав работу из стандартной начальной конфигурации, после выполнения совокупности команд переходит в стандартную заключительную конфигурацию.

Полученный результат располагается между маркерами и сдвинут вплотную к левому маркеру.

#### Задания для самостоятельной работы

1. Построить машину Тьюринга, выполняющую операцию конкатенации двух цепочек, заданных во входном алфавите  $A = \{0, 1, *, \epsilon\}$ .
2. Построить машину Тьюринга, выполняющую операцию копирования входной цепочки, заданной в алфавите  $A = \{1, *, \epsilon\}$ , где символ “\*” используется в качестве разделителя двух цепочек.
3. Построить машину Тьюринга в алфавите  $A = \{0, 1\}$ , которая, начав работу с последней единицы массива из единиц, сдвигает его на одну ячейку влево, не изменяя остального содержимого ленты. Головка останавливается на первой единице перенесенного массива.
4. По заданной совокупности команд машины Тьюринга  $T$  и начальной конфигурации  $K$  найти заключительную конфигурацию.

4.1.

$$\begin{aligned} q_0 1 &\rightarrow q_0 1R, & q_1 0 &\rightarrow q_2 1E, \\ q_0 0 &\rightarrow q_1 1R, & q_1 1 &\rightarrow q_1 1L, \\ \text{а) } K &= 1101q_0 01; & \text{б) } K &= 101q_0 010. \end{aligned}$$

4.2.

$$\begin{aligned} q_0 0 &\rightarrow q_0 1L, & q_1 1 &\rightarrow q_0 0R, \\ q_0 1 &\rightarrow q_1 1R, & q_1 0 &\rightarrow q_2 0L, \\ \text{а) } K &= 1q_1 0111; & \text{б) } K &= 1q_1 1111. \end{aligned}$$

4.3.

$$\begin{aligned} q_0 0 &\rightarrow q_1 0L, & q_1 0 &\rightarrow q_1 1L, \\ q_0 1 &\rightarrow q_0 0R, & q_1 1 &\rightarrow q_2 0R, \\ \text{а) } K &= 1000q_1 01; & \text{б) } K &= 11q_1 11101. \end{aligned}$$

5. Построить машину Тьюринга, которая во входной цепочке, заданной в алфавите  $A = \{0, 1, \epsilon\}$ , переставляет единицы и нули так, чтобы все единицы были в начале, а нули в конце цепочки.

6. Построить композицию  $T_1 \cdot T_2$  машин Тьюринга  $T_1$  и  $T_2$  по паре состояний  $(q_{1z}, q_{20})$  и найти результат применения композиции  $T_1 \cdot T_2$  к слову  $D$ .

6.1. Машины  $T_1$  и  $T_2$  заданы таблицами соответствия:

$T_1$ :

	$q_{10}$	$q_{11}$	$q_{12}$
0	$q_{1z}0L$	$q_{12}0R$	$q_{10}0R$
1	$q_{11}1R$	$q_{12}1R$	$q_{10}0R$

	$q_{20}$	$q_{21}$

$T_2$ :

0	$q_{21}1L$	$q_{2z}0R$
1	$q_{21}1L$	$q_{20}0L$

а)  $D = 111100111011$ ;

б)  $D = 11010111$ .

6.2. Машины  $T_1$  и  $T_2$  заданы таблицами соответствия:

$T_1$ :

	$q_{10}$	$q_{11}$	$q_{12}$
0	$q_{11}0R$	$q_{12}0R$	$q_{1z}1L$
1	$q_{10}1R$	$q_{10}1R$	

$T_2$ :

	$q_{20}$	$q_{21}$	$q_{22}$
0	$q_{21}0L$	$q_{2z}0L$	$q_{2z}0R$
1	$q_{20}1L$	$q_{21}1L$	$q_{2z}1L$

а)  $D = 11000101001$ ;

б)  $D = 10100111110$ .

7. Найти результат применения итерации машины  $T$  по паре состояний  $(q_{z1}, q_i)$  к слову  $D$ . Заключительными состояниями машины  $T$  являются  $q_{z1}$  и  $q_{z2}$ . На ленте первоначально записаны нули, а в начальной конфигурации головка указывает на левый символ входной цепочки, состоящей из единиц.

7.1. Машина Тьюринга задана таблицей соответствия:

$T$ :

	$q_0$	$q_1$	$q_2$	$q_3$	$q_4$
0	$q_{z1}0E$	$q_30E$	$q_40E$	$q_41R$	$q_{z2}1L$
1	$q_10R$	$q_20R$	$q_00R$		

а)  $i = 1, D = 1^{3k}, k \geq 1$ ;

б)  $i = 1, D = 1^{3k+2}, k \geq 1$ .

7.2. Машина Тьюринга задана таблицей соответствия:

T:

	q <sub>0</sub>	q <sub>1</sub>	q <sub>2</sub>	q <sub>3</sub>	q <sub>4</sub>	q <sub>5</sub>
0	q <sub>22</sub> 0R	q <sub>22</sub> 0R	q <sub>3</sub> 0R	q <sub>4</sub> 1L	q <sub>5</sub> 0L	q <sub>21</sub> 0R
1	q <sub>1</sub> 0R	q <sub>2</sub> 0R	q <sub>2</sub> 1R	q <sub>1</sub> 0E	q <sub>4</sub> 1L	q <sub>5</sub> 1L

а)  $i = 3, D = 1^{2k+1}, k \geq 1;$       б)  $i = 1, D = 1^{3k+2}, k \geq 1.$

#### 4.УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

##### **Перечень основной и дополнительной литературы, необходимой для освоения дисциплины**

###### **Перечень основной литературы:**

3. В. Н. Пильщиков, В. Г. Абрамов, А. А. Вылиток, И. В. Горячая. Машины Тьюринга и алгоритмы Маркова. Решение задач. — М.:МГУ, 2006. [11]. А. Ахо., Р. Сети, Дж. Ульман. Компиляторы: принципы, технологии, инструменты. — М.: «Вильямс», 2001.
4. А. Ахо, М. Лам, Р.Сети, Дж. Ульман. Компиляторы: принципы, технологии и инструментарий. — М.: «Вильямс», 2008.

###### **Перечень дополнительной литературы:**

3. Олифер, В. Г. Компьютерные сети. Принципы, технологии, протоколы: учеб. пособие / В. Г. Олифер, Н. А. Олифер. - 4-е изд. - СПб.: Питер, 2011.
4. Таненбаум, Э. С. Архитектура компьютера / Э. С. Таненбаум; пер.: Ю. Гороховский, Д. Шинтяков. - 5-е изд. - СПб.: Питер, 2009.

##### **Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимых для освоения дисциплины:**

5. <http://www.intuit.ru> – сайт дистанционного образования в области информационных технологий
6. <http://www.iqlib.ru> - интернет библиотека образовательных изданий, в которой собраны электронные учебники, справочные и учебные пособия;
7. <http://www.biblioclub.ru> - электронная библиотечная система «Университетская библиотека – online»: специализируется на учебных материалах для ВУЗов по научно-гуманитарной тематике.
8. <http://window.edu.ru> – образовательные ресурсы ведущих вузов

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное автономное образовательное учреждение  
высшего образования  
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»  
Пятигорский институт (филиал) СКФУ

**МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ КОНТРОЛЬНОЙ РАБОТЫ  
ПО ДИСЦИПЛИНЕ  
ТЕОРИЯ ФОРМАЛЬНЫХ ЯЗЫКОВ, ГРАММАТИК И ПОСТРОЕНИЕ  
ТРАНСЛЯТОРОВ**

Направление подготовки	<b>09.04.02</b>
Направленность (профиль)	<b>Информационные системы и технологии «Технологии работы с данными и знаниями, анализ информации»</b>
Квалификация выпускника	<b>Магистр</b>

Пятигорск, 2024

Введение.....	76
1.....	Цель, задачи

76	
2.....	Формулировка задания и его объем
76	
3.....	Общие требования к написанию и оформлению работы
77	
4.....	ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ НА КОНТРОЛЬНУЮ РАБОТУ
77	
5.....	Рекомендации по выполнению задания
81	
5.1. ОСНОВНЫЕ ПОНЯТИЯ ТЕОРИИ АЛГОРИТМОВ.....	81
5.2. РЕКУРСИВНЫЕ ФУНКЦИИ.....	84
5.3. МАШИНЫ ТЬЮРИНГА.....	87
5.4. ФОРМАЛЬНЫЕ ГРАММАТИКИ И ЯЗЫКИ.....	94
5.5. ТЕОРИЯ АВТОМАТОВ.....	103
6.....	Учебно-методическое и информационное обеспечение дисциплины
115	

## **ВВЕДЕНИЕ**

Методические указания содержат перечень вариантов заданий для контрольных работ, требования к оформлению контрольных работ и пример выполнения задания. Теоретической основой подготовки специалиста являются знания в области информатики, вычислительной систем.

### **4. ЦЕЛЬ, ЗАДАЧИ**

Целью дисциплины «Теория формальных языков, грамматик и построение трансляторов» является ознакомление с основными понятиями и методами использования теории формальных грамматик, овладение теоретическими знаниями о фазы грамматического разбора при компиляции и интерпретации формальных текстов и практическими навыками по алгоритмам лексического, грамматического и семантического анализа.

Задачами дисциплины «Теория формальных языков, грамматик и построение трансляторов» являются:

- освоение принципов построения формальных языков программирования, работы компиляторов;
- разработка алгоритма с использованием грамматических структур формальных грамматик;
- знакомство с работой лексического анализатора языка программирования;
- моделирование лексического анализатора математического выражения.

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 контрольной работы.

### **5. ФОРМУЛИРОВКА ЗАДАНИЯ И ЕГО ОБЪЕМ**

Методические указания содержат теоретический материал по основным разделам

дисциплины, сопровождается методикой решения задач, примерами, а также приводятся задания для контрольной работы, после каждого раздела. Выбор варианта контрольной работе по последней цифре зачетной книжки.

## 6. ОБЩИЕ ТРЕБОВАНИЯ К НАПИСАНИЮ И ОФОРМЛЕНИЮ РАБОТЫ

Контрольная работа выполняется и сдается в электронном виде на CD/CDRW носителе. На конверте необходимо указать название дисциплины, ФИО студента, факультет, номер группы, шифр зачетной книжки, № варианта задания, и список всех созданных в ходе выполнения задания файлов.

Приведенный в конце методических указаний список литературы может использоваться студентами при выполнении контрольной работы.

## 7. ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ НА КОНТРОЛЬНУЮ РАБОТУ

### Тема 1. Формальные языки и грамматики.

Во всех заданиях необходимо разработать схемы алгоритмов и проанализировать процесс реализации алгоритма, т.е. последовательность шагов, которая будет порождена при применении алгоритма к конкретным исходным данным.

*Базовый  
уровень*

1. Разработать словесные алгоритмы вычитания из некоторого числа  $A$  последовательности  $n$  чисел  $b_1, b_2, \dots, b_n$ :

а) алгоритм вычисления по формуле:

$$C = (\dots((A - b_1) - b_2) - \dots - b_n)$$

б) алгоритм вычисления по формуле:

$$C = A - \sum_{i=1}^n b_i .$$

2. Разработать алгоритм вычисления по формуле:

$$C_k = \sum_{i=1}^n a_i - b_k (1 \leq i \leq n, 1 \leq k \leq m) .$$

3. Разработать алгоритм вычисления по формуле:

$$C_i = \prod_{k=1}^m a_k \times b_i (1 \leq i \leq m, 1 \leq k \leq m) .$$

4. Разработать алгоритм поиска максимального (минимального) элемента из последовательности, заданной в виде одномерного массива  $A = \{a_1, a_2, \dots, a_k\}$ .

5. Разработать алгоритм определения количества одинаковых чисел в последовательности, заданной в виде одномерного массива  $A = \{a_1, a_2, \dots, a_k\}$ .

*Повышен  
ный*

1. Разработать алгоритм подсчета количества одинаковых элементов в матрице  $B$

- уровень* размерностью  $n \times m$ . Размерность матрицы вводится с клавиатуры.
2. Разработать алгоритм умножения матрицы на вектор.
  3. Разработать алгоритм умножения матрицы на матрицу.
  4. Разработать алгоритм транспонирования матрицы.
  5. Разработать алгоритм, реализующий операцию объединения следующих двух множеств:  $A = \{a_1, a_2, \dots, a_k\}$  и  $B = \{b_1, b_2\}$ .

**Тема 2. Регулярные множества и регулярные выражения (РВ), конечные автоматы и автоматы с магазинной памятью. КС-грамматики и алгоритмы разбора.**

- Базовый уровень* Доказать примитивную рекурсивность следующих функций.
2. Усеченное вычитание

$$x_1 \div x_2 = \begin{cases} x_1 - x_2, & \text{если } x_1 > x_2, \\ 0, & \text{если } x_1 < x_2. \end{cases}$$

2. Функция знака

$$\text{Sg}(x) = \begin{cases} 0, & \text{если } x = 0, \\ 1, & \text{если } x \neq 0. \end{cases}$$

3. Нахождение минимального значения из двух заданных чисел  $f(x, y) = \min(x, y)$ .
4. Нахождение максимального значения из двух заданных чисел  $f(x, y) = \max(x, y)$ .
5. Усеченное вычитание

$$x \div 1 = \begin{cases} x - 1, & \text{если } x > 1, \\ 0, & \text{если } x < 1. \end{cases}$$

6. Функция  $r(x, y)$  - остаток от деления  $y$  на  $x$ .
7. Функция  $q(x, y)$  - частное от деления  $y$  на  $x$ .
8. Доказать, что предикат  $\text{Pd}_n(x)$  "x делится на n" примитивно-рекурсивен для любого  $n$ .
9. Доказать, что предикат  $\text{Pd}_{n,m}(x)$  "x делится на n и m" примитивно-рекурсивен для любых  $n$  и  $m$ .
10. Доказать, что отношение  $x_1 > x_2$  примитивно-рекурсивно.

- Повышенный уровень*
1. Доказать, что предикат  $f(x) = g(x)$  примитивно-рекурсивен, если функции  $f(x)$  и  $g(x)$  примитивно-рекурсивны.
  2. Дано множество слов одинаковой длины, причем первые два слова выделены. Построить цепь от первого выделенного слова ко второму так, чтобы все слова этой цепи были только из заданного множества. Соседние слова построенной цепи должны отличаться только одной буквой.
  3. Дано множество слов. Построить из них кроссворд заданной конфигурации (число слов может быть больше требуемого количества для заполнения кроссворда).
  4. Грани кубика разбиты на клетки  $5 \times 5$ . Каждая из клеток выкрашена в

белый или синий цвет. Переход из клетки в клетку допускается только через общую сторону при условии совпадения цветов этих клеток. Построить маршрут перехода из одной заданной клетки в другую заданную клетку этого же цвета.

5. Имеется клетчатая ткань размером  $N \times N$  клеток. Разрезать ее на  $M$  квадратных частей, не нарушая целостности клеток.

### Тема 3. Элементы теории трансляции. Операционные системы и их роль в процессе трансляции.

*Базовый  
уровень*

7. Построить машину Тьюринга, выполняющую операцию конкатенации двух цепочек, заданных во входном алфавите  $A = \{0, 1, *, \varepsilon\}$ .
8. Построить машину Тьюринга, выполняющую операцию копирования входной цепочки, заданной в алфавите  $A = \{1, *, \varepsilon\}$ , где символ "\*" используется в качестве разделителя двух цепочек.
9. Построить машину Тьюринга в алфавите  $A = \{0, 1\}$ , которая, начав работу с последней единицы массива из единиц, сдвигает его на одну ячейку влево, не изменяя остального содержимого ленты. Головка останавливается на первой единице перенесенного массива.
10. По заданной совокупности команд машины Тьюринга  $T$  и начальной конфигурации  $K$  найти заключительную конфигурацию.

4.1.

$$\begin{array}{ll} q_0 1 \rightarrow q_0 1 R, & q_1 0 \rightarrow q_2 1 E, \\ q_0 0 \rightarrow q_1 1 R, & q_1 1 \rightarrow q_1 1 L, \\ \text{а) } K = 1101q_0 01; & \text{б) } K = 101q_0 010. \end{array}$$

4.2.

$$\begin{array}{ll} q_0 0 \rightarrow q_0 1 L, & q_1 1 \rightarrow q_0 0 R, \\ q_0 1 \rightarrow q_1 1 R, & q_1 0 \rightarrow q_2 0 L, \\ \text{а) } K = 1q_1 0111; & \text{б) } K = 1q_1 1111. \end{array}$$

4.3.

$$\begin{array}{ll} q_0 0 \rightarrow q_1 0 L, & q_1 0 \rightarrow q_1 1 L, \\ q_0 1 \rightarrow q_0 0 R, & q_1 1 \rightarrow q_2 0 R, \\ \text{а) } K = 1000q_1 01; & \text{б) } K = 11q_1 11101. \end{array}$$

11. Построить машину Тьюринга, которая во входной цепочке, заданной в алфавите  $A = \{0, 1, \varepsilon\}$ , переставляет единицы и нули так, чтобы все единицы были в начале, а нули в конце цепочки.

*Повышен  
ный  
уровень*

1. Построить композицию  $T_1 \cdot T_2$  машин Тьюринга  $T_1$  и  $T_2$  по паре состояний  $(q_{1z}, q_{20})$  и найти результат применения композиции  $T_1 \cdot T_2$  к слову  $D$ .

1.1. Машины  $T_1$  и  $T_2$  заданы таблицами соответствия:

	$q_{10}$	$q_{11}$	$q_{12}$
0	$q_{1z}0L$	$q_{1z}0R$	$q_{10}0R$
1	$q_{11}1R$	$q_{1z}1R$	$q_{10}0R$

	$q_{20}$	$q_{21}$
--	----------	----------

0	$q_{21}1L$	$q_{2z}0R$
1	$q_{21}1L$	$q_{20}0L$

а)  $D = 111100111011$ ;б)  $D = 11010111$ .1.2. Машины  $T_1$  и  $T_2$  заданы таблицами соответствия:

	$q_{10}$	$q_{11}$	$q_{12}$
0	$q_{11}0R$	$q_{1z}0R$	$q_{1z}1L$
1	$q_{10}1R$	$q_{10}1R$	

	$q_{20}$	$q_{21}$	$q_{22}$
0	$q_{21}0L$	$q_{2z}0L$	$q_{2z}0R$
1	$q_{20}1L$	$q_{21}1L$	$q_{2z}1L$

а)  $D = 11000101001$ ;б)  $D = 10100111110$ .

2. Найти результат применения итерации машины  $T$  по паре состояний  $(q_{z1}, q_i)$  к слову  $D$ . Заключительными состояниями машины  $T$  являются  $q_{z1}$  и  $q_{z2}$ . На ленте первоначально записаны нули, а в начальной конфигурации головка указывает на левый символ входной цепочки, состоящей из единиц.

2.1. Машина Тьюринга задана таблицей соответствия:

	$q_0$	$q_1$	$q_2$	$q_3$	$q_4$
0	$q_{z1}0E$	$q_30E$	$q_40E$	$q_41R$	$q_{z2}1L$
1	$q_10R$	$q_20R$	$q_00R$		

а)  $i = 1, D = 1^{3k}, k \geq 1$ ;б)  $i = 1, D = 1^{3k+2}, k \geq 1$ .

2.2. 7.2. Машина Тьюринга задана таблицей соответствия:

	$q_0$	$q_1$	$q_2$	$q_3$	$q_4$	$q_5$
0	$q_{z2}0R$	$q_{z2}0R$	$q_30R$	$q_41L$	$q_50L$	$q_{z1}0R$
1	$q_10R$	$q_20R$	$q_21R$	$q_10E$	$q_41L$	$q_51L$

а)  $i = 3, D = 1^{2k+1}, k \geq 1$ ;б)  $i = 1, D = 1^{3k+2}, k \geq 1$ .

#### Тема 4. Автоматизированные методы построения компиляторов.

Базовый  
уровень

1. Построить КС-грамматику, порождающую язык  $a^n$  для  $n > 0$ .
2. Построить КС-грамматику, порождающую язык  $a^n b^n$  для  $n > 0$ .
3. Построить КС-грамматику, порождающую язык  $(ab)^n c^* a^{n+m}$  для  $n > 0$  и  $m > 0$ .
4. Построить КС-грамматику, порождающую язык  $(ab)^* c a^* b^{n+m}$  для  $n > 0$  и  $m > 0$ .
5. Построить КС-грамматику, порождающую язык  $a^n b^{3m} c^m a^{2n} = a^n (bbb)^m c^m (aa)^n$  для  $n > 1$  и  $m > 1$ .

Повышен  
ный  
уровень

1. Построить КС-грамматику, порождающую язык  $(ab)^n (ca)^* b^{n+2} (abc)^*$  для  $n > 0$ .
2. Для леворекурсивной грамматики  $G: S \rightarrow SabA \mid ba; A \rightarrow AbA \mid bAa \mid c$  построить эквивалентную праворекурсивную грамматику.

3. Построить алгоритм устранения правой рекурсии и доказать эквивалентность соответствующего преобразования.
4. Построить КС-грамматику, порождающую идентификаторы, при этом обозначить символом  $t$  любую букву, а символом  $f$  - любую цифру.
5. Дана КС-грамматика  $G$ :  
 $S \rightarrow Aab \mid bSb \mid abB$ ;  
 $A \rightarrow abS \mid ba \mid bbA$ ;  
 $B \rightarrow bB \mid Da \mid Aa$ ;  
 $D \rightarrow Dbb \mid aDa$   
 Построить эквивалентную грамматику, все нетерминалы которой продуктивны.

## 8. РЕКОМЕНДАЦИИ ПО ВЫПОЛНЕНИЮ ЗАДАНИЯ

### 5.1. ОСНОВНЫЕ ПОНЯТИЯ ТЕОРИИ АЛГОРИТМОВ

Предварительные сведения

Понятие алгоритма, являющееся одним из основных понятий математики, возникло задолго до появления вычислительных машин. На протяжении многих веков люди пользовались интуитивным понятием алгоритма. Арабский математик IX века Мухаммед ибн Муса Аль-Хорезми впервые выдвинул идею о том, что решение любой поставленной математической и философской задачи может быть оформлено в виде последовательности механически выполняемых правил, т.е. может быть алгоритмизировано. Этому же мнению придерживались Декарт, Лейбниц, Гильберт.

Алгоритм - это строгая и четкая конечная система правил решения некоторого класса задач, определяющая процесс преобразования исходных данных в искомый результат. В рамках данного определения понятие алгоритма отождествлялось с понятием метода вычисления, традиции организации вычислений складывались веками и стали составной частью общей научной культуры, формулировались и успешно применялись на практике различные вычислительные алгоритмы. Поэтому понятие метода вычислений считалось

### 5.2. Основные требования к алгоритмам

Каждый алгоритм имеет дело с данными - входными, промежуточными и выходными. Данные как объекты, с которыми могут работать алгоритмы, должны быть четко определены и отличимы как друг от друга, так и от другой информации. Данные для своего размещения требуют памяти. Память обычно считается однородной и дискретной. Единицы измерения объема данных и памяти согласованы, при этом память может быть бесконечной.

Если выполнение алгоритма заканчивается получением результатов, то говорят, что он применим к рассматриваемой совокупности исходных данных. Любой применимый алгоритм имеет следующие основные свойства, раскрывающие его определение.

5. Дискретность. Это свойство состоит в том, что алгоритм должен представлять процесс решения задачи как последовательное выполнение простых (или ранее определенных) шагов. При этом для выполнения каждого шага алгоритма требуется некоторый конечный отрезок времени, то есть преобразование исходных данных в результат осуществляется во времени дискретно.
6. Определенность (или детерминированность). Это свойство состоит в том, что каждое правило алгоритма должно быть четким и однозначным. Благодаря этому свойству выполнение алгоритма носит механический характер и не требует никаких дополнительных указаний или сведений о решаемой задаче.

7. Результативность (или конечность). Это свойство состоит в том, что алгоритм должен приводить к решению задачи за конечное число шагов.
8. Массовость. Это свойство состоит в том, что алгоритм решения задачи разрабатывается в общем виде и должен быть применим для некоторого класса задач, различающихся лишь исходными данными. Для задания алгоритма необходимо выделить и описать следующие его элементы:
  - набор объектов, составляющих совокупность данных: исходных, промежуточных и конечных;
  - правило начала;
  - правила непосредственной переработки информации;
  - правило окончания;
  - правило извлечения результатов.

Алгоритм всегда рассчитан на конкретного исполнителя. Если исполнителем является компьютер, то алгоритм должен быть записан на языке программирования.

### 5.3. Математическое определение алгоритма

Интуитивное определение алгоритма не позволяет рассматривать свойства алгоритмов как свойства формальных объектов. Поэтому математическое определение алгоритма необходимо по следующим причинам:

- 1) только при наличии формального определения алгоритма можно сделать вывод о разрешимости или неразрешимости каких-либо проблем;
- 2) это дает возможность для сравнения алгоритмов, предназначенных для решения одинаковых задач;
- 3) это дает возможность для сравнения различных проблем по сложности алгоритмов их решения. Одна из причин расплывчатости интуитивного определения алгоритма состоит в том, что объектом алгоритма может оказаться все, что угодно. Поэтому естественно было начать с формализации понятия объекта. В вычислительных алгоритмах объектами работы являются числа, в алгоритме шахматной игры - фигуры и позиции, при алгоритмизации производственных процессов объектами служат, например, показания приборов. Однако алгоритмы оперируют не с объектами реального мира, а с изображениями этих объектов.

Поэтому алгоритмами в современной математике принято называть конструктивно задаваемые соответствия между изображениями объектов в абстрактных алфавитах. Абстрактным алфавитом называется любая конечная совокупность объектов, называемых буквами или символами данного алфавита. При этом природа этих объектов нас совершенно не интересует. Символом абстрактных алфавитов можно считать буквы алфавита какого-либо языка, цифры, любые значки и даже слова некоторого конкретного языка. Основным требованием к алфавиту является его конечность. Таким образом, можно утверждать, что алфавит - это конечное множество различных символов. Алфавит, как любое множество, задается перечислением его элементов. Итак, объекты реального мира можно изображать словами в различных алфавитах. Это позволяет считать, что объектами работы алгоритмов могут быть только слова. Тогда можно сформулировать следующее определение.

Алгоритм есть четкая конечная система правил для преобразования слов из некоторого алфавита в слова из этого же алфавита. Слово, к которому применяется алгоритм, называется входным. Слово, вырабатываемое в результате применения алгоритма, называется выходным. Совокупность слов, к которым применим данный алгоритм, называется областью применимости этого алгоритма.

Формальные определения алгоритма появились в 30-х - 40-х годах XX века. Можно выделить три основных типа универсальных алгоритмических моделей, различающихся исходными эвристическими соображениями относительно того, что такое алгоритм. Первый тип связывает понятие алгоритма с наиболее традиционными понятиями математики - вычислениями и числовыми функциями. Наиболее развитая и изученная модель этого типа - рекурсивные функции - является исторически первой формализацией понятия алгоритма. Эта модель основана на функциональном подходе и рассматривает понятие алгоритма с точки зрения того, что можно вычислить с его помощью.

Второй тип основан на представлении алгоритма как некоторого детерминированного устройства, способного выполнять в каждый отдельный момент некоторые примитивные операции, или инструкции. Такое представление не оставляет сомнений в однозначности алгоритма и элементарности его шагов. Основной теоретической моделью этого типа, созданной в 30-х годах, является машина Тьюринга, которая представляет собой автоматную модель, в основе которой лежит анализ процесса выполнения алгоритма как совокупности набора инструкций. Третий тип алгоритмических моделей - это преобразования слов в произвольных алфавитах, в которых элементарными операциями являются подстановки, т.е. замены части слова (подслова) другим словом. Преимущество этого типа состоит в его максимальной абстрактности и возможности применить понятие алгоритма к объектам произвольной природы. Модели второго и третьего типа довольно близки и отличаются в основном эвристическими подходами. Примерами моделей этого типа являются нормальные алгоритмы Маркова и канонические системы Поста.

#### 5.4. Понятие алфавитного оператора

Понятие алфавитного оператора является наиболее общим. К нему фактически сводятся любые процессы преобразования информации. К изучению алфавитных операторов фактически сводится теория любых преобразователей информации. Основой теории алфавитных операторов являются способы их задания. Если область определения алфавитного оператора конечна, то оператор может быть задан простой таблицей соответствия. В случае бесконечной области определения алфавитного оператора он задается системой правил, позволяющей за конечное число шагов найти выходное слово, соответствующее заданному входному слову. Алфавитные операторы, задаваемые с помощью конечной системы правил, называются алгоритмами.

Алгоритмы, в соответствии с которыми решение поставленных задач сводится к арифметическим действиям, называются численными алгоритмами.

Алгоритмы, в соответствии с которыми решение поставленных задач сводится к логическим действиям, называются логическими алгоритмами.

Различают однозначные и многозначные алфавитные операторы.

Под однозначным алфавитным оператором понимается такой алфавитный оператор, который каждому входному слову ставит в соответствие не более одного выходного слова. Под многозначным алфавитным оператором понимается такой алфавитный оператор, который каждому входному слову ставит в соответствие более одного выходного слова.

Алфавитный оператор, не сопоставляющий данному входному слову  $a_i$  никакого выходного слова  $b_j$  (в том числе и пустого), не определен на этом слове.

Совокупность всех слов, на которых алфавитный оператор определен, называется областью его определения. Два алфавитных оператора считаются равными, если равны соответствующие им алфавитные операторы и совпадает система правил, задающих

действие этих алгоритмов на выходные слова. Два алгоритма считаются эквивалентными, если у них совпадают алфавитные операторы, но не совпадают способы их задания (система правил). Обычно в теории алгоритмов рассматриваются лишь такие алгоритмы, которым соответствуют однозначные

## 5.2. РЕКУРСИВНЫЕ ФУНКЦИИ

Общие сведения

Первой алгоритмической системой была система, основанная на использовании конструктивно определяемых арифметических (целочисленных) функций, получивших специальное название рекурсивных функций. Рекурсией называется способ задания функции, при котором значение определяемой функции для произвольных значений аргументов выражается известным образом через значения определяемой функции для меньших значений аргументов. Применение рекурсивных функций в теории алгоритмов основано на идее нумерации слов в произвольном алфавите последовательными натуральными числами. Наиболее простотакую нумерацию можно осуществить, располагая слова в порядке возрастания их длин, а слова, имеющие одинаковую длину, - в произвольном порядке. После нумерации входных и выходных слов в произвольном алфавитном операторе этот оператор превращается в функцию  $y = f(x)$ , в которой аргумент  $x$  и функция  $y$  принимают неотрицательные целочисленные значения. Функция  $f(x)$  может быть определена не для всех значений  $x$ , а лишь для тех, которые составляют область определения этой функции.

Понятие простейших функций

Числовые функции, значение которых можно установить посредством некоторого алгоритма, называются вычислимыми функциями. Для того чтобы описать класс функций с помощью рекурсивных определений, рассмотрим набор простейших функций:

1)  $Z(x_1, x_2, \dots, x_n) = 0$  - нуль-функция, которая определена для всех неотрицательных значений аргумента;

2)  $s(x) = x + 1$  - функция непосредственного следования, также определенная для всех целых неотрицательных значений своего аргумента;

3)  $I(x_1, x_2, \dots, x_m, \dots, x_n) = x_m$  - функция выбора (тождества), повторяющая значения своих аргументов. Используя простейшие функции в качестве исходных функций, можно с помощью небольшого числа общих конструктивных приемов строить сложные арифметические функции. В теории рекурсивных функций особо важное значение имеют три операции: суперпозиции, примитивной рекурсии и минимизации.

Оператор суперпозиции

Оператором суперпозиции  $S$  называется подстановка в функцию от  $m$  переменных  $m$  функций от  $n$  одних и тех же переменных. Она дает новую функцию от  $n$  переменных. Например, из функций  $f(x_1, x_2, \dots, x_m)$ ,  $f_1(x_1, x_2, \dots, x_n)$ ,  $f_2(x_1, x_2, \dots, x_n)$ ,  $\dots$ ,  $f_m(x_1, x_2, \dots, x_n)$  можно получить новую функцию:  $S_{m+1}(f, f_1, f_2, \dots, f_m) = g(x_1, x_2, \dots, x_n) = f(f_1(x_1, x_2, \dots, x_n), f_2(x_1, x_2, \dots, x_n), \dots, f_m(x_1, \dots, x_n))$ . (1)

В операции суперпозиции  $S_{m+1}$  индекс сверху указывает на число функций. Таким образом, при помощи оператора суперпозиции и функции выбора можно выразить любую подстановку функции в функцию. Например, осуществляя операцию суперпозиции функций  $f(x) = 0$  и  $g(x) = x + 1$ , получим функцию:  $h(x) = g(f(x)) = 0 + 1 = 1$ .

При суперпозиции функции  $g(x)$  с этой же функцией получим функцию  $h(x) = g(g(x)) = x + 2$ . Используя подстановку и функции тождества, можно переставлять и отождествлять аргументы в функции:

$$f(x_2, x_1, x_3, \dots, x_n) = f(I_{22}(x_1, x_2), I_{12}(x_1, x_2), x_3, \dots, x_n);$$

$$f(x_1, x_1, x_3, \dots, x_n) = f(I_{12}(x_1, x_2), I_{12}(x_1, x_2), x_3, \dots, x_n).$$

Таким образом, если заданы функции тождества и операторы суперпозиции, то можно

считать заданными всевозможные операторы подстановки функций в функции, а также переименования, перестановки и отождествления переменных.

Оператор примитивной рекурсии

Оператор примитивной рекурсии  $R_n$  позволяет определить  $(n+1)$ -местную функцию  $f$  по двум заданным функциям, одна из которых является  $n$ -местной функцией  $g$ , а другая  $(n+2)$ -местной функцией  $h$ .

Функция  $f(x_1, x_2, \dots, x_n, y)$  получается оператором примитивной рекурсии из функций  $g(x_1, x_2, \dots, x_n)$  и функции  $h(x_1, x_2, \dots, x_n, y, z)$ , если:

$$f(x_1, x_2, \dots, x_n, 0) = g(x_1, x_2, \dots, x_n); \quad (2)$$

$$f(x_1, x_2, \dots, x_n, y+1) = h(x_1, x_2, \dots, x_n, y, f(x_1, x_2, \dots, x_n, y)).$$

Приведенная пара равенств (2) называется схемой примитивной рекурсии. Для понимания операции примитивной рекурсии необходимо заметить, что всякую функцию от меньшего числа аргументов можно рассматривать как функцию от большего числа аргументов. Существенным в операторе примитивной рекурсии является то, что независимо от числа переменных в  $f$  рекурсия ведется только по одной переменной  $y$ . Остальные  $n$  переменных  $x_1, x_2, \dots, x_n$  на момент применения схемы (2) зафиксированы и играют роль параметров.

Оператор минимизации

Отношение  $P(x_1, x_2, \dots, x_n)$  называется примитивно-рекурсивным, если примитивно рекурсивна его характеристическая функция:

$$\chi(x_1, x_2, \dots, x_n) = \begin{cases} 1, & \text{если } P(x_1, x_2, \dots, x_n) \text{ - "истина",} \\ 0, & \text{если } P(x_1, x_2, \dots, x_n) \text{ - "ложь".} \end{cases}$$

Предикат называется примитивно-рекурсивным, если его характеристическая функция примитивно-рекурсивна.

Функция  $f(x_1, x_2, \dots, x_n)$  получается посредством оператора минимизации из предиката  $P(x_1, x_2, \dots, x_n, z)$ , если в любой точке значения функции  $f(x_1, x_2, \dots, x_n)$  является минимальное значение  $z$ , обращающее предикат  $P(x_1, x_2, \dots, x_n, z)$  в значение «истина»:

$$f(x_1, x_2, \dots, x_n) = \mu z (P(x_1, x_2, \dots, x_n, z)),$$

где  $\mu z$  – оператор минимизации.

#### 2.2.4. Ограниченный оператор минимизации

Функция  $f(x_1, x_2, \dots, x_n)$  получается ограниченным оператором минимизации из предиката  $P(x_1, x_2, \dots, x_n, z)$  и функции  $U(x_1, x_2, \dots, x_n)$ , если в любой точке значения этой функции определено следующим образом:

1) для любого  $z < U(x_1, x_2, \dots, x_n)$  такого, что  $P(x_1, x_2, \dots, x_n, z) = \text{"истина"}$ , значение функции  $f(x_1, x_2, \dots, x_n) = \mu z (P(x_1, \dots, x_n, z))$ ,

2) не для любого  $z < U(x_1, x_2, \dots, x_n)$  такого, что  $P(x_1, x_2, \dots, x_n, z) = \text{"истина"}$ , значение функции  $f(x_1, x_2, \dots, x_n) = U(x_1, x_2, \dots, x_n)$ .

Данное определение записывается следующим образом:  $f(x_1, x_2, \dots, x_n) = \mu z < U(x) (P(x_1, x_2, \dots, x_n, z))$ . Ограничение  $z$  в ограниченном операторе минимизации дает гарантию окончания вычислений, поскольку оно оценивает сверху число вычислений предиката  $P$ . Возможность оценить сверху количество вычислений является существенной особенностью примитивно-рекурсивных функций.

Примитивно-рекурсивные и частично-рекурсивные функции

Большинство вычислимых функций относится к классу примитивно-рекурсивных функций. Функция называется примитивно-рекурсивной, если она может быть получена из простейших функций с помощью конечного числа операторов суперпозиции и примитивной рекурсии.

Если некоторые функции являются примитивно-рекурсивными, то в результате применения к ним операторов суперпозиции или примитивной рекурсии можно получить новые примитивно-рекурсивные функции. Существует три возможности доказательства

того, что функция является примитивно-рекурсивной:

- а) показать, что заданная функция является простейшей;
- б) показать, что заданная функция построена с помощью оператора суперпозиции;
- в) показать, что заданная функция построена с помощью оператора примитивной рекурсии.

Тем не менее примитивно-рекурсивные функции не охватывают все вычислимые функции, которые могут быть определены конструктивно. При построении этих функций могут использоваться другие операции. Функция называется частично-рекурсивной, если она может быть получена из простейших функций с помощью конечного числа операторов суперпозиции, примитивной рекурсии и минимизации. Указанные операции могут быть выполнены в любой последовательности и любое конечное число раз. Если такие функции оказываются всюду определенными, то они называются общерекурсивными функциями. Частично-рекурсивные функции вычислимы путем определенной процедуры механического характера. Практически понятием частично-рекурсивных функций пользуются для доказательства алгоритмической разрешимости или неразрешимости проблем. Приведем тезис Черча, который связывает понятие алгоритма и строгое математическое понятие частично-рекурсивной функции. Тезис Черча: всякий алгоритм может быть реализован частично-рекурсивной функцией. Утверждение о несуществовании частично-рекурсивной функции эквивалентно несуществованию алгоритма решения соответствующей задачи.

#### Типы рекурсивных алгоритмов

Эффективность разработки рекурсивного алгоритма определяется наличием некоторых условий:

- 1) если исходные данные имеют рекурсивную структуру, то процедуры анализа таких структур будут более эффективны, если они сами рекурсивны;
- 2) если алгоритм обработки некоторого набора данных построить, разбивая данные на части и обрабатывая в отдельности каждую часть, то из частичных решений можно получить общее;
- 3) если по условию задачи необходимо выбрать оптимальный вариант из некоторого множества решений, то искомое решение может быть найдено через конечное число шагов. При этом на каждом шаге удаляется часть информации, и далее задача решается на меньшем объеме данных. Поиск решения завершается после окончания данных либо при нахождении искомого решения на текущем наборе данных.

#### Методика решения задач

##### Использование оператора примитивной рекурсии

Пример 1. Доказать примитивную рекурсивность функции  $f(x, y) = x + y$ .

Решение. Рассмотрим способ рекурсивного определения данной функции:  $f(x, 0) = x$ ,  $f(x, y+1) = x + y + 1 = f(x, y) + 1$ . Чтобы показать соответствие данной рекурсивной схемы схеме примитивной рекурсии, воспользуемся функциями выбора и следования:  $f(x, 0) = x = I(x)$ ,  $f(x, y+1) = f(x, y) + 1 = S(f(x, y)) = S(I_3(x, y, f(x, y)))$ .

Пример 2. Доказать примитивную рекурсивность функции  $f(x, y) = x \cdot y$ .

Решение. Рассмотрим способ рекурсивного определения данной функции:

$f(x, 0) = 0$ ,  $f(x, y+1) = x \cdot (y + 1) = x \cdot y + x = f(x, y) + x$ , из которого следует, что  $Z(x) = x \cdot 0$ . Обозначим  $x \cdot y = p(x, y)$ , тогда:  
 $p(x, 0) = Z(x)$ ;  $p(x, y+1) = p(x, y) + x = S(p(x, y), x) = S(I_3(x, y, p(x, y)), I_3(x, y, p(x, y)))$ .

Пример 3. Доказать примитивную рекурсивность функции

$$Sg(x) = \begin{cases} 0, & \text{если } x = 0, \\ 1, & \text{если } x \neq 0. \end{cases}$$

Решение. Рассмотрим способ рекурсивного определения данной функции:  $Sg(0) = 0 =$

$Z(x), Sg(x+1) = Z(x) + 1 = 1 = S(Z(x))$ .

Пример 4. Доказать примитивную рекурсивность функции  $f(x) = 2x$ .

Решение. Рассмотрим способ рекурсивного определения данной функции:

$$f(0) = 1 = S(Z(x)),$$

$$f(x+1) = 2 \cdot 2x = 2 \cdot f(x) = S(S(Z(x))).$$

Пример 5. Доказать примитивную рекурсивность функции  $f(x, y) = xy$

Решение. Рассмотрим способ рекурсивного определения данной функции:

$$f(x, 0) = 1 = S(Z(x));$$

$$f(x, y+1) = x^{y+1} = x^y \cdot x = f(x, y) \cdot I_1^2(x, y) = p(I_3^3(x, y, f(x, y)), I_1^3(x, y, f(x, y))).$$

Использование оператора минимизации

Пример 1. Пусть  $f(x, y) = \mu z (2 \cdot x + z = y + 1)$ , откуда предикат  $P(x, y, z) = 2 \cdot x + z = y + 1$ .

Покажем примитивную рекурсивность предиката  $P$ . Его характеристическая функция может быть представлена следующим выражением:

$$Sg(Sg((2 \cdot x + z) \div (y + 1)) + Sg((y + 1) \div (2 \cdot x + z))).$$

Найдем значение функции в точке  $(1, 5)$ :

$$\text{При } z = 0: P(1, 5, 0) = 2 \cdot 1 + 0 = 5 + 1 - \text{"ложь"},$$

$$z = 1: P(1, 5, 1) = 2 \cdot 1 + 1 = 5 + 1 - \text{"ложь"},$$

$$z = 2: P(1, 5, 2) = 2 \cdot 1 + 2 = 5 + 1 - \text{"ложь"},$$

$$z = 3: P(1, 5, 3) = 2 \cdot 1 + 3 = 5 + 1 - \text{"ложь"},$$

$$z = 4: P(1, 5, 4) = 2 \cdot 1 + 4 = 5 + 1 - \text{"истина"}.$$

Таким образом, минимальное значение переменной  $z$ , обращающее предикат в "истину", дает значение функции в точке  $(1, 5)$ , и оно равно 4.

### 5.3. МАШИНЫ ТЬЮРИНГА

#### Общие сведения

Одним из первых формальных определений алгоритма было определение английского математика А. Тьюринга, который в 1936 г. описал схему некоторой гипотетической (абстрактной) машины и формализовал правила выполнения действий при помощи описания работы этой машины. Машина Тьюринга является абстракцией, которую нельзя реализовать практически. Поэтому алгоритмы для машины Тьюринга должны выполняться другими средствами.

Основным следствием формализации алгоритмов с использованием машины Тьюринга является возможность доказательства существования или несуществования алгоритмов решения различных задач. Описывая различные алгоритмы для машин Тьюринга и доказывая реализуемость всевозможных композиций алгоритмов, Тьюринг убедительно показал разнообразие возможностей предложенной им конструкции, что позволило ему выступить со следующим тезисом: "Всякий алгоритм может быть реализован соответствующей машиной Тьюринга".

Доказать тезис Тьюринга нельзя, так как в его формулировке не определено понятие "всякий алгоритм". Его можно только обосновать, представляя различные алгоритмы в виде машин Тьюринга. Было доказано, что класс функций, вычисляемых на этих машинах, совпадает с классом частично рекурсивных функций.

#### Неформальное определение машины Тьюринга

Машина Тьюринга представляет собой автомат, имеющий бесконечную в обе стороны ленту, считывающую головку и управляющее устройство. Управляющее устройство может находиться в одном из состояний, образующих конечное множество  $Q = \{q_0, q_1, \dots, q_n\}$ . Множество  $Q$  называют внутренним алфавитом машины Тьюринга

Принципиальное отличие машины Тьюринга от вычислительных машин состоит в том, что ее запоминающее устройство представляет собой бесконечную ленту, из-за которой невозможна ее физическая реализация. Лента разделена на ячейки, в каждой из которых может быть записан один из символов конечного алфавита  $A = \{a_0, a_1, \dots, a_m\}$ , который называют входным алфавитом машины Тьюринга. Во время функционирования машины Тьюринга может быть заполнено конечное число ячеек. Считывающая головка в каждый момент времени обозревает ячейку ленты, в зависимости от символа в этой ячейке и состояния управляющего устройства записывает в ячейку новый символ или оставляет его без изменения, сдвигается на ячейку влево или вправо или остается на месте. При этом управляющее устройство переходит в новое состояние или остается в старом. Среди состояний управляющего устройства выделены начальное состояние  $q_0$  и заключительное состояние  $q_z$ .

Таким образом, за один такт работы машина Тьюринга может считать символ, записать вместо него новый или оставить его без изменения и сдвинуть головку на одну ячейку влево или вправо или оставить ее на месте.

#### Формальное определение машины Тьюринга

Алфавит  $A$  – это некоторое конечное множество символов.

Цепочкой над алфавитом называется последовательность символов из этого алфавита. Длина цепочки  $x$  представляет собой число символов в цепочке и обозначается  $|x|$ . Длина пустой цепочки равна нулю.

Конкатенацией цепочек  $X$  и  $Y$  называют цепочку, полученную приписыванием символов цепочки  $Y$  справа к цепочке  $X$ .

Машиной Тьюринга называется семерка вида  $T = (Q, A, \delta, p_0, p_z, a_0, a_1)$ , где  $Q$  – конечное множество состояний, в которых может находиться управляющее устройство;

$A$  – входной алфавит;

$\delta$  – функция переходов,  $\delta = Q \cdot A \rightarrow Q \cdot A \cdot S$ , где

$S = \{R, L, E\}$  – направления сдвига;

$p_0$  – начальное состояние;  $p_0 \in Q$ ;

$p_z$  – заключительное состояние;  $p_z \in Q$ ;

$a_0$  – символ для обозначения пустой ячейки,  $a_0 \in A$ ;

$a_1$  – специальный символ - разделитель цепочек на ленте,  $a_1 \in A$ .

Командой машины Тьюринга называется элемент функции переходов  $qa \rightarrow pbr$ , где  $q$  и  $r \in Q$ ;  $a$  и  $b \in A$ ;  $r \in S$ . Каждая команда описывает один такт работы машины Тьюринга.

Конфигурация машины Тьюринга представляется следующим образом:  $t = \langle CqaB \rangle$ , где  $C$  – цепочка, находящаяся слева от считывающей головки;

$q$  – состояние машины Тьюринга;

$a$  – символ, находящийся под головкой машины Тьюринга;

$B$  – цепочка, находящаяся справа от головки машины Тьюринга.

Конфигурация  $\langle CqaB \rangle$  непосредственно переходит в конфигурацию  $\langle CnqnBn \rangle$ , если новая конфигурация получена в результате применения одной команды к исходной конфигурации. Обозначим непосредственный переход из одной конфигурации в другую следующим образом:  $\langle CqaB \rangle \Rightarrow \langle CnqnBn \rangle$ .

Конфигурация, содержащая начальное состояние, называется начальной, а содержащая заключительное состояние – заключительной. Если цепочка  $C$  в конфигурации пустая, то начальная и заключительная конфигурации называются стандартными. Машина Тьюринга перерабатывает цепочку  $x$  в цепочку  $y$ , если, действуя из начальной конфигурации и имея на ленте цепочку  $x$ , машина Тьюринга переходит в заключительную конфигурацию, имея на ленте цепочку  $y$ . Если начальная и заключительная конфигурации являются стандартными, то процесс переработки  $x$  в  $y$  является правильной переработкой.

Способы представления машины Тьюринга

Существует три способа представления машины Тьюринга: совокупностью команд, в виде графа, в виде таблицы соответствия.

Представление машины Тьюринга совокупностью команд

Совокупность всех команд, которые может выполнять машина, называется ее программой.

Машина Тьюринга считается заданной, если заданы ее внешний и внутренний алфавиты, программа, начальная конфигурация и указано, какие из символов обозначают пустую ячейку и заключительное состояние. Чтобы записать совокупность команд, нужно воспользоваться следующими правилами:

- 5) начальному шагу алгоритма ставится в соответствие  $q_0$
- 6) циклы реализуются так, что последнее действие цикла должно соответствовать переходу в то состояние, которое соответствует началу цикла;
- 7) соседним шагам алгоритма соответствует переход в смежные состояния, которые соответствуют этим пунктам;
- 8) последний шаг алгоритма вызывает переход в заключительное состояние.

В качестве примера рассмотрим совокупность команд машины Тьюринга, которая инвертирует входную цепочку, записанную с использованием нулей и единиц.

Пусть алфавит машины Тьюринга задан множеством  $A = \{0, 1, \varepsilon\}$ , где символ  $\varepsilon$  соответствует пустой ячейке, а число состояний устройства управления задано в виде множества  $Q = \{q_0, q_1, q_z\}$

Если, например, начальная конфигурация имеет вид  $q_0 1 1 0 0 1 1$ , то конечная конфигурация после завершения операции инвертирования должна иметь вид  $q_z 0 0 1 1 0 0$ .

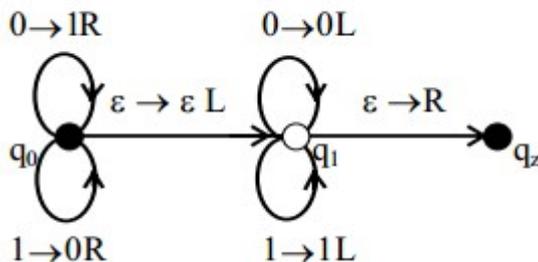
Для решения задачи машиной будет порождена следующая последовательность команд:

$$\begin{aligned} q_0 1 &\rightarrow q_0 0 R, & q_1 0 &\rightarrow q_1 1 L, \\ q_0 0 &\rightarrow q_0 1 R, & q_1 1 &\rightarrow q_1 0 L, \\ q_0 \varepsilon &\rightarrow q_1 \varepsilon L, & q_1 \varepsilon &\rightarrow q_z \varepsilon R. \end{aligned}$$

В стандартной начальной конфигурации головка стоит над первым символом слева, и устройство управления находится в начальном состоянии. На следующем такте машина Тьюринга, не меняя своего состояния, заменяет символ 0 на 1 или 1 на 0 и сдвигается вправо на один символ. После просмотра всей цепочки под головкой оказывается символ, указывающий на пустую ячейку. В этом случае машина Тьюринга переходит в новое состояние и сдвигается влево на один символ. На последующих тактах управляющее устройство не меняет своего состояния, оставляет без изменения символ под головкой и перемещается влево до тех пор, пока не встретит пустую ячейку. Встретив пустую ячейку, машина Тьюринга переходит в заключительное состояние и перемещается вправо на один символ, переходя в стандартную заключительную конфигурацию.

Представление машины Тьюринга графом

При представлении машины Тьюринга посредством графа необходимо каждому состоянию поставить в соответствие вершину графа, а каждой команде - помеченную дугу. Машина Тьюринга из рассмотренного примера инвертирования цепочки, состоящей из символов 0 и 1, будет представлена в виде графа следующим образом:



Начальная и конечная вершины графа обычно выделяются; на рисунке они отмечены черным кружком. Если на очередном такте работы машины Тьюринга символ на ленте не изменяется, то в правой части команды его можно не писать ( $\epsilon$  на ребре в состоянии  $q_z$ ).

#### Представление машины Тьюринга таблицей соответствия

При представлении машины Тьюринга данным способом составляется таблица, в которой каждому состоянию соответствует строка, а каждому символу из входного алфавита - столбец. В клетках таблицы на пересечении строки и столбца будет находиться действие (или правая часть команды).

Таблица соответствия для задания машины Тьюринга из рассмотренного примера будет выглядеть следующим образом:

	0	1	$\epsilon$
$q_0$	$q_0 1R$	$Q_0 0R$	$q_1 \epsilon L$
$q_1$	$q_1 0L$	$Q_1 1L$	$q_z \epsilon L$

Инвертирование входной цепочки можно выполнить программой машины Тьюринга, приведенной в таблице соответствия. Эта программа включает шесть команд.

#### 7.5. Вычислимые функции

Говорят, что машина Тьюринга вычисляет функцию  $f(x_1, x_2, \dots, x_n)$ , если выполняются следующие условия:

- 3) для любых  $x_1, x_2, \dots, x_n$ , принадлежащих области определения функции, машина Тьюринга из начальной конфигурации, имея на ленте представление аргументов, переходит в заключительную конфигурацию, имея на ленте результат (представление функции);
- 4) для любых  $x_1, x_2, \dots, x_n$ , не принадлежащих области определения функции, машина Тьюринга из начальной конфигурации работает бесконечно.

Если начальная и заключительная конфигурации машины Тьюринга являются стандартными, то говорят, что машина Тьюринга правильно вычисляет функцию  $f$ . Функция называется вычислимой по Тьюрингу, если существует машина Тьюринга, вычисляющая ее.

Для того чтобы доказать вычислимость функции, а в дальнейшем и существование алгоритма, необходимо построить машину Тьюринга, реализация которой на практике зачастую представляет собой трудоемкую задачу. В связи с этим возникает необходимость разбиения алгоритма на отдельные задачи, каждая из которых будет решаться отдельной машиной Тьюринга. Если объединить программы этих машин, то получится новая программа, позволяющая решить исходную задачу.

Машины Тьюринга могут вычислять искомую функцию с восстановлением и без восстановления. Вычисление функции с восстановлением означает работу машины Тьюринга с сохранением исходных данных:

$$p_0 1X_1^* \dots 1X_n^* \Rightarrow^* p_z 1f(X_1, X_2, \dots, X_n) 1X_1^* \dots 1X_n^*.$$

Приведенное определение позволяет получать на ленте сначала результат, а затем исходные данные. В отдельных случаях удобно сделать наоборот:

$$p_0 1X_1^* \dots 1X_n^* \Rightarrow^* 1X_1^* \dots 1X_n^* p_z 1f(X_1, X_2, \dots, X_n).$$

Вычисление функции без восстановления означает работу машины Тьюринга без сохранения исходных данных:

$$p_0 \ 1X_1^* \dots * 1X_n \Rightarrow^* p \ z \ 1f(X_1, X_2, \dots, X_n).$$

Справедливо утверждение, что всякая правильно вычислимая функция правильно вычислима с восстановлением.

Операции над машинами Тьюринга

1. Композиция машин Тьюринга. Пусть машины  $T_1$  и  $T_2$  имеют программы  $P_1$  и  $P_2$ . Предположим, что внутренние алфавиты этих машин не пересекаются; пусть  $qz_1$  - заключительное состояние машины  $T_1$ , а  $q_2$  - начальное состояние машины  $T_2$ . Заменяем всюду в программе  $P_1$  заключительное состояние  $qz_1$  на начальное состояние  $q_2$  машины  $T_2$  и полученную программу объединим с программой  $P_2$ . Новая программа  $P$  определяет машину  $T$ , называемую композицией машин  $T_1$  и  $T_2$  по паре состояний  $(qz_1, q_2)$ .

Композиция машин может быть обозначена  $T_1 \cdot T_2$  или  $T_1 T_2$ . Более подробно композиция машин записывается следующим образом:

$$T = T(T_1, T_2, (qz_1, q_2)),$$

$$\text{где } T_1 = (Q_1, A_1, \delta_1, p_1, pz_1, a_1, a_{11}),$$

$$T_2 = (Q_2, A_2, \delta_2, p_2, pz_2, a_2, a_{12}).$$

Пусть  $a_1 = a_2 = a_0$  и  $a_{11} = a_{12} = a_1$ . Внешний алфавит композиции  $T_1 T_2$  является объединением внешних алфавитов машин  $T_1$  и  $T_2$ :

$$p_2$$

$$T = (Q_1 \cup Q_2 \setminus \{pz_1\}, A_1 \cup A_2, |(\delta_1, \cup \delta_2), p_1, pz_2, a_0, a_1).$$

$$pz_1.$$

Операция композиции, выполняемая над алгоритмами, позволяет получать новые, более сложные алгоритмы из ранее известных простых алгоритмов.

2. Итерация машины Тьюринга. Эта операция применима только к одной машине. Пусть  $qz$  - заключительное состояние машины  $T$ , а  $qn$  - какое-либо состояние машины  $T$ , не являющееся заключительным. Заменяем всюду в программе  $P$  машины  $T$  состояние  $qz$  на  $qn$ . Полученная программа определяет новую машину  $T'(qz, qn)$ , которая называется итерацией машины  $T$  по паре состояний  $(qz, qn)$ . Если машина Тьюринга имеет одно заключительное состояние, то после выполнения итерации получается машина, не имеющая заключительного состояния.

3. Разветвление машин Тьюринга. Пусть машины Тьюринга  $T_1$ ,  $T_2$  и  $T_3$  задаются программами  $P_1$ ,  $P_2$  и  $P_3$  соответственно. Считаем, что внутренние алфавиты этих машин попарно не пересекаются. Пусть  $qz_{11}$  и  $qz_{12}$  - какие-либо различные заключительные состояния машины  $T_1$ . Заменяем всюду в программе  $P_1$  состояние  $qz_{11}$  начальным состоянием  $q_2$  машины  $T_2$ , а состояние  $qz_{12}$  начальным состоянием  $q_3$  машины  $T_3$ . Затем новую программу объединим с программами  $P_2$  и  $P_3$ . Получим программу  $P$ , задающую машину Тьюринга и обозначаемую:

$$T = T(T_1, (qz_{11}, q_2), T_2, (qz_{12}, q_3), T_3).$$

Машина  $T$  называется разветвлением машин  $T_2$  и  $T_3$ , управляемым машиной  $T_1$ .

Примеры построения машин Тьюринга

Пример 1. Построить машину Тьюринга, которая правильно вычисляет функцию  $f(x) = x+1$  по правилам двоичного сложения.

Решение. Исходя из формулировки задачи, требующей вычислить функцию по правилам сложения в двоичной системе сложения, выберем входной алфавит:

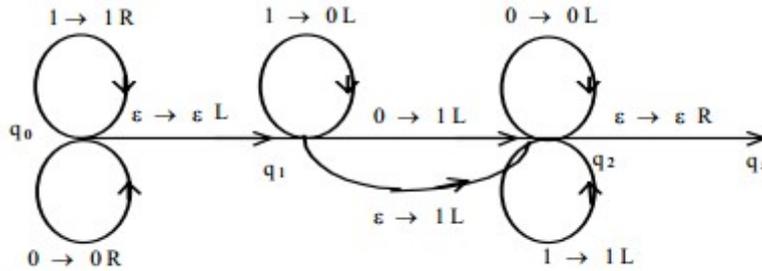
$$A = \{0, 1, \epsilon\}.$$

Представим машины Тьюринга таблицей соответствия и графом.

Таблица соответствия:

	0	1	$\epsilon$
$q_0$	$q_0 0R$	$q_0 1R$	$Q_1 \epsilon L$
$q_1$	$q_2 1L$	$q_1 0L$	$q_2 1L$
$q_2$	$q_2 0L$	$Q_2 1L$	$q_z \epsilon R$

Представление машины Тьюринга в виде графа:



Запишем программу построенной машины Тьюринга для случая, когда входная цепочка на ленте равна двоичному числу 111. Слева от каждой команды приведем представление входной цепочки на ленте до выполнения данной команды. Символ, который находится под головкой, будем помечать подчеркиванием.

- 6)  $q_0 1 \rightarrow q_0 1R \ \epsilon 111 \epsilon$  6)  $q_1 1 \rightarrow q_1 0L \ \epsilon 110 \epsilon$
- 7)  $q_0 1 \rightarrow q_0 1R \ \epsilon 111 \epsilon$  7)  $q_1 1 \rightarrow q_1 0L \ \epsilon 100 \epsilon$
- 8)  $q_0 1 \rightarrow q_0 1R \ \epsilon 111 \epsilon$  8)  $q_1 \epsilon \rightarrow q_2 1L \ \epsilon 000 \epsilon$
- 9)  $q_0 \epsilon \rightarrow q_1 \epsilon L \ \epsilon 111 \epsilon$  9)  $q_2 \epsilon \rightarrow q_z \epsilon R \ \epsilon 1000 \epsilon$
- 10)  $q_1 1 \rightarrow q_1 0L \ \epsilon 111 \epsilon$

Из примера видно, что машина Тьюринга из стандартной начальной конфигурации, имея на ленте аргумент 111, выполнив совокупность команд 1-9, перешла в стандартное заключительное состояние, имея на ленте результат 1000. Действительно:  $1112 + 12 = 10002$ .

Пример 2. Построить машину Тьюринга, выполняющую операцию  $(x \div 2)$  и имеющую входной алфавит  $A = \{0, 1, \epsilon\}$ .

Таблица соответствия данной машины Тьюринга будет иметь следующий вид:

	0	1	$\epsilon$		0	1	$\epsilon$
$q_0$	$q_0 0R$	$q_0 1R$	$q_1 \epsilon L$	$q_2$	$q_3 1L$	$q_2 0L$	$q_3 1L$
$q_1$	$q_3 \epsilon L$	$q_2 \epsilon L$		$q_3$	$q_2 0L$	$q_2 1L$	$q_z \epsilon R$

Операция  $(x \div 2)$  реализована сдвигом цепочки вправо на 1 разряд.

Пример 3. Построить машину Тьюринга, которая выполняет копирование заданного аргумента. Выберем входной алфавит  $A = \{0, 1, \epsilon\}$ . Представим данную машину таблицей соответствия:

	0	1	*	$\varepsilon$
$q_0$	$q_11L$	$q_10R$	$q_0^*L$	$q_z\varepsilon R$
$q_1$		$q_11R$	$q_2^*R$	$q_2^*R$
$q_2$		$q_21R$		$q_31L$
$q_3$	$q_00R$	$q_31L$	$q_3^*L$	

Запишем программу построенной машины для заданной входной цепочки на ленте, равной двоичному числу 111. Слева от каждой команды приведем представление входной цепочки на ленте до выполнения данной команды. Символ, который находится под головкой, будем помечать подчеркиванием.

- |  |  |
|--|--|
| 17) $q_01 \rightarrow q_10R \ \varepsilon 111\varepsilon$              | 17) $q_31 \rightarrow q_31L \ \varepsilon 001^*11\varepsilon$            |
| 18) $q_11 \rightarrow q_11R \ \varepsilon 011\varepsilon$              | 18) $q_30 \rightarrow q_00R \ \varepsilon 001^*11\varepsilon$            |
| 19) $q_11 \rightarrow q_11R \ \varepsilon 011\varepsilon$              | 19) $q_01 \rightarrow q_10R \ \varepsilon 000^*11\varepsilon$            |
| 20) $q_1\varepsilon \rightarrow q_2^*R \ \varepsilon 011\varepsilon$   | 20) $q_1^* \rightarrow q_2^*R \ \varepsilon 000^*11\varepsilon$          |
| 21) $q_2\varepsilon \rightarrow q_31L \ \varepsilon 011^*\varepsilon$  | 21) $q_21 \rightarrow q_21R \ \varepsilon 000^*11\varepsilon$            |
| 22) $q_3^* \rightarrow q_3^*L \ \varepsilon 011^*1\varepsilon$         | 22) $q_21 \rightarrow q_21R \ \varepsilon 000^*11\varepsilon$            |
| 23) $q_31 \rightarrow q_31L \ \varepsilon 011^*1\varepsilon$           | 23) $q_2\varepsilon \rightarrow q_31L \ \varepsilon 000^*11\varepsilon$  |
| 24) $q_31 \rightarrow q_31L \ \varepsilon 011^*1\varepsilon$           | 24) $q_31 \rightarrow q_31L \ \varepsilon 000^*111\varepsilon$           |
| 25) $q_30 \rightarrow q_00R \ \varepsilon 011^*1\varepsilon$           | 25) $q_31 \rightarrow q_31L \ \varepsilon 000^*111\varepsilon$           |
| 26) $q_01 \rightarrow q_10R \ \varepsilon 011^*1\varepsilon$           | 26) $q_3^* \rightarrow q_3^*L \ \varepsilon 000^*111\varepsilon$         |
| 27) $q_11 \rightarrow q_11R \ \varepsilon 001^*1\varepsilon$           | 27) $q_30 \rightarrow q_00R \ \varepsilon 000^*111\varepsilon$           |
| 28) $q_1^* \rightarrow q_2^*R \ \varepsilon 001^*1\varepsilon$         | 28) $q_0^* \rightarrow q_0^*L \ \varepsilon 000^*111\varepsilon$         |
| 29) $q_21 \rightarrow q_21R \ \varepsilon 001^*1\varepsilon$           | 29) $q_00 \rightarrow q_01L \ \varepsilon 000^*111\varepsilon$           |
| 30) $q_2\varepsilon \rightarrow q_31L \ \varepsilon 001^*1\varepsilon$ | 30) $q_00 \rightarrow q_01L \ \varepsilon 001^*111\varepsilon$           |
| 31) $q_31 \rightarrow q_31L \ \varepsilon 001^*11\varepsilon$          | 31) $q_00 \rightarrow q_01L \ \varepsilon 011^*111\varepsilon$           |
| 32) $q_3^* \rightarrow q_3^*L \ \varepsilon 001^*11\varepsilon$        | 32) $q_0\varepsilon \rightarrow q_z1R \ \varepsilon 111^*111\varepsilon$ |

Из примера видно, что машина Тьюринга из стандартной начальной конфигурации, имея на ленте аргумент 111 и выполнив совокупность команд 1-32, перешла в стандартное заключительное состояние, имея на ленте результат  $\varepsilon 111^*111\varepsilon$ .

#### Машина Тьюринга с полулентой

В рассмотренных выше определениях машины Тьюринга использовалась бесконечная в обе стороны лента. Ограничим ленту с одной стороны и покажем, что машина Тьюринга с правой или левой полулентой эквивалентна машине Тьюринга с бесконечной лентой. Говорят, что функция, правильно вычисляемая на машине Тьюринга с обычной лентой, правильно вычислима и на машине Тьюринга с правой полулентой, т.е. для любой машины Тьюринга  $T$  существует эквивалентная ей машина с правой полулентой  $TR$

Идея доказательства этого утверждения основана на следующих предпосылках:

- рабочая область ленты ограничена двумя маркерами: неподвижным - слева и подвижным - справа;
- на внутренней части ограниченной области машина Тьюринга с полулентой должна работать как обычная машина Тьюринга, а при выходе на маркеры она должна освобождать рабочее пространство, для этого правый маркер может сдвигаться вправо, а при выходе левый маркер необходимо сдвинуть всю цепочку вправо;
- полученный результат, находящийся между маркерами, в конце работы машины Тьюринга нужно сдвинуть вплотную к левому маркеру.

Машина Тьюринга может вычислять функцию с восстановлением, то есть с сохранением исходных данных. Это позволяет получить на ленте сначала результат, а затем –

исходные данные или наоборот.

Рассмотрим пример построения машины Тьюринга с правой полулентой, вычисляющей функцию  $f(x) = 2x$ . Алгоритм вычисления данной функции состоит в приписывании нуля к входной цепочке справа.

Таблица соответствия данной машины Тьюринга будет иметь следующий вид:

	<	0	1	>	$\epsilon$
$q_0$		$q_0 0R$	$q_0 1R$	$q_1 0R$	
$q_1$					$q_2 >L$
$q_2$	$q_2 < R$	$q_2 0L$	$q_2 1L$		

Здесь символ "<" обозначает левый маркер, а символ ">" - правый маркер. Машина Тьюринга, начав работу из стандартной начальной конфигурации, после выполнения совокупности команд переходит в стандартную заключительную конфигурацию.

Полученный результат располагается между маркерами и сдвинут вплотную к левому маркеру.

#### 5.4. ФОРМАЛЬНЫЕ ГРАММАТИКИ И ЯЗЫКИ

Общие сведения

В последние три десятилетия появилось большое количество работ по общей теории языков и грамматик. Можно выделить четыре научных направления, которые удалось объединить по методам их исследования в одну общую задачу теории языков. Первое из этих направлений связано с построением формальной, или математической лингвистики, которая начала особенно быстро развиваться в тот период, когда были сформулированы вопросы машинного перевода. Эта проблема требовала формализации понятий словарь, грамматика, язык, их классификации и умения относить конкретные словари, грамматики, языки к определенному классу. Интерес к задачам такого рода еще более усилился с появлением искусственных языков программирования. С появлением трансляторов проблема перевода во многом определила.

Независимо от указанных двух направлений развивалось построение формальных моделей динамических систем. Для создания продуктивной теории эти модели должны были быть, с одной стороны, достаточно узкими, а с другой - достаточно широкими, чтобы охватить некоторый общий класс прикладных задач. Типичным примером такого рода является модель конечного автомата. Эта модель позволяет описать многие процессы, заданные на конечных множествах и развивающиеся в счетном времени, что позволило создать для нее продуктивную теорию. Если в эту модель ввести бесконечность по какому-либо параметру, то это приводит к общему классу систем, например, к машинам Тьюринга.

Независимо и параллельно развивалась общая теория алгоритмов как ветвь современной математики. Развитие вычислительной техники поставило перед математической теорией алгоритмов новую задачу: стало необходимым классифицировать алгоритмы по различным признакам. Эквивалентность понятий "алгоритм" и "машина Тьюринга" позволила предположить, что поиски классификации алгоритмов окажутся связанными с поисками промежуточных моделей между моделями конечного автомата и машиной Тьюринга.

Таким образом, перечисленные четыре направления оказались тесно связанными. Теория языков, порожденная чисто лингвистическими задачами, оказалась в центре интересов математиков, занимающихся теорией алгоритмов, и ученых, разрабатывающих

абстрактные модели динамических систем и теоретические основы автоматки.

Теория формальных языков и грамматик является разделом математической лингвистики - специфической математической дисциплины, ориентированной на изучение структуры естественных и искусственных языков. По характеру используемого математического аппарата теория формальных грамматик и языков близка к теории алгоритмов и теории автоматов.

На интуитивном уровне язык можно определить как некоторое множество предложений с заданной структурой, имеющих определенное значение. Правила, определяющие допустимые конструкции языка, составляют синтаксис языка. Значение, или смысл фразы определяется семантикой языка.

Если бы все языки состояли из конечного числа предложений, то не было бы проблемы синтаксиса. Но, так как язык содержит неограниченное (или довольно большое) число правильно построенных фраз, то возникает проблема описания бесконечных языков с помощью каких-либо конечных средств. Различают следующие виды описания языков:

- 1) порождающее, которое предполагает наличие алгоритма, последовательно порождающего все правильные предложения языка;
- 2) распознающее, которое предполагает наличие алгоритма, определяющего принадлежность любой фразы данному языку.

#### а. Основные понятия порождающих грамматик

Алфавит - это непустое конечное множество. Элементы алфавита называются символами. Цепочка над алфавитом  $\Sigma = \{a_1, a_2, \dots, a_n\}$  есть конечная последовательность элементов  $a_i$ . Множество всех цепочек над алфавитом  $\Sigma$  обозначается  $\Sigma^*$ .

Длина цепочки  $x$  равна числу ее элементов и обозначается  $|x|$ . Цепочка нулевой длины называется пустой и обозначается символом  $\varepsilon$ . Соответственно, непустая цепочка определяется как цепочка ненулевой длины. Пусть имеется алфавит  $\Sigma = \{a, b\}$ . Тогда множество всех цепочек определяется следующим образом:  $\Sigma^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, \dots\}$ .

Цепочки  $x$  и  $y$  равны, если они одинаковой длины и совпадают с точностью до порядка символов, из которых они состоят.

Над цепочками  $x$  и  $y$  определена операция сцепления (конкатенации, произведения), которая получается дописыванием символов цепочки  $y$  за символами цепочки  $x$ .

Язык  $L$  над алфавитом  $\Sigma$  представляет собой множество цепочек над  $\Sigma$ . Необходимо различать пустой язык  $L = \emptyset$  и язык, содержащий только пустую цепочку:  $L = \{\varepsilon\} \neq \emptyset$ .

Формальный язык  $L$  над алфавитом  $\Sigma$  - это язык, выделенный с помощью конечного множества некоторых формальных правил.

Пусть  $M$  и  $L$  - языки над алфавитами. Тогда конкатенация  $LM = \{xy | x \in L, y \in M\}$ . В частности,  $\{\varepsilon\}L = L\{\varepsilon\} = L$ . Используя понятие произведения, определим итерацию  $L^*$  и усеченную итерацию  $L^+$  множества  $L$ :

$$L^+ = \bigcup_{i=1}^{\infty} L^i,$$

$$L^* = \bigcup_{i=1}^{\infty} L^i,$$

где  $i$  - степень языка,  $L$  определяется рекурсивно следующим образом:

$$L^0 = \{\varepsilon\};$$

$$L^1 = L;$$

$$L^{n+1} = L^n L = L L^n;$$

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

Например, если задан язык  $L = \{a\}$ , тогда  $L^* = \{\varepsilon, a, aa, aaa, \dots\}$ ,  $L^+ = \{a, aa, aaa, \dots\}$ .

Порождающая грамматика - это упорядоченная четверка  $G = (VT, VN, P, S)$ , где  $VT$  - конечный алфавит, определяющий множество терминальных символов;  $VN$  - конечный алфавит, определяющий множество нетерминальных символов;  $P$  - конечное множество правил вывода, т.е. множество пар следующего вида:

$$u \rightarrow v, \text{ где } u, v \in (VT \cup VN)^*;$$

$S$  - начальный символ (аксиома грамматики),  $S \in VN$ .

Из терминальных символов состоят цепочки языка, порожденного грамматикой. Аксиомой называется символ в левой части первого правила вывода грамматики.

Для того чтобы различать терминальные и нетерминальные символы, принято обозначать терминальные символы строчными, а нетерминальные символы заглавными буквами латинского алфавита.

В грамматике  $G$  цепочка  $x$  непосредственно порождает цепочку  $y$ , если  $x = \alpha\beta$ ,  $y = \alpha\gamma\beta$  и  $\alpha\beta \rightarrow \alpha\gamma\beta \in P$ , т.е. цепочка  $y$  непосредственно выводится из  $x$ , что обозначается  $x \Rightarrow y$ .

Языком, порождаемым грамматикой  $G = (VT, VN, P, S)$ , называется множество терминальных цепочек, выводимых в грамматике  $G$  из аксиомы:

$$L(G) = \{x \mid x \in VT^*; S \Rightarrow^* x\}, \text{ где } \Rightarrow^* - \text{ выводимость.}$$

Пример. Дана грамматика  $G = (VT, VN, P, S)$ , в которой  $VT = \{a, b\}$ ,  $VN = \{S\}$ ,  $P = \{S \rightarrow aSb, S \rightarrow ab\}$ . Определить язык, порождаемый этой грамматикой.

Решение. Используя рекурсию, выведем несколько цепочек языка, порождаемого данной грамматикой:

$$S \rightarrow ab;$$

$$S \rightarrow aSb \Rightarrow aabb;$$

$$S \rightarrow aSb \Rightarrow aaSbb \Rightarrow aaabbb; \dots$$

Определим язык, порожденный данной грамматикой:

$$L(G) = \{anbn \mid n > 0\}.$$

Говоря о представлении грамматик, нужно отметить, что множество правил вывода грамматики может приводиться без специального указания множеств терминалов и нетерминалов. В таком случае обычно предполагается, что грамматика содержит в точности те терминальные и нетерминальные символы, которые встречаются в правилах вывода.

Также предполагается, что правые части правил, для которых совпадают левые части, можно записать в одну строку с использованием разделителя. Так, правила вывода из приведенного примера можно записать следующим образом:  $S \rightarrow aSb \mid ab$ .

### Классификация грамматик

Правила порождающих грамматик позволяют осуществлять преобразования строк. Ограничения же на виды правил позволяют выделить классы грамматик. Рассмотрим классификацию, которую предложил Н. Хомский.

Грамматики типа 0 - это грамматики, на правила вывода которых нет ограничений. Грамматики типа 1, или контекстные грамматики - это грамматики, все правила которых имеют следующий вид:  $xAy \rightarrow x\phi y$ , где  $A \in VN$ ,  $x, y, \phi \in (VN \cup VT)^+$ . Грамматики типа 2 - это бесконтекстные, или контекстно-свободные грамматики (КС-грамматики). Правила вывода для этих грамматик имеют следующий вид:  $A \rightarrow \phi$ , где  $A \in VN$ ,  $\phi \in (VN \cup VT)^*$

Грамматики типа 3 - это автоматные грамматики, которые делятся на два типа:

а) левосторонние (леворекурсивные), правила вывода для которых имеют следующий вид:  $A \rightarrow Aa \mid a$ , где  $A \in VN$ ;

б) правосторонние (праворекурсивные), правила вывода для которых имеют следующий вид:  $A \rightarrow Aa \mid a$ .

Язык  $L$  называется языком типа  $i$ , если существует грамматика типа  $i$ , порождающая язык  $L$ .

Методика решения задач Пример 1. Дана порождающая грамматика  $G = (VT, VN, P, S)$ , в которой  $VT = \{a, d, e\}$ ,  $VN = \{B, C, S\}$ ,  $P = \{S \rightarrow aB, B \rightarrow Cd \mid dC, C \rightarrow e\}$ . Выписать

терминальные цепочки, порожденные данной грамматикой, и определить длину их вывода.

Решение. Получим следующие терминальные цепочки:

$S \rightarrow aB \rightarrow aCd \rightarrow aed,$

$S \rightarrow aB \rightarrow adC \rightarrow ade,$

длина вывода которых равна 3.

Пример 2. Дана грамматика  $G = (VT, VN, P, C)$ , в которой  $VT = \{a, b, c, d, e\}$ ,  $VN = \{A, B, C, D, E\}$ ,  $P = \{A \rightarrow ed, B \rightarrow Ab, C \rightarrow Bc, C \rightarrow dD, D \rightarrow Ae, E \rightarrow bc\}$ .

Определить, принадлежит ли языку  $L(G)$  цепочка  $eadabcb$ .

Решение. Выведем возможные терминальные цепочки из аксиомы с помощью заданных правил вывода:

$C \rightarrow Bc \rightarrow Abc \rightarrow edbc,$

$C \rightarrow dD \rightarrow dAe \rightarrow dede.$

Так как первые же терминальные символы полученных цепочек не совпадают с заданной, можно сделать вывод о том, что цепочка  $eadabcb$  не принадлежит языку  $L(G)$ .

Пример 3. Построить КС-грамматику (грамматику типа 2), порождающую цепочки из 0 и 1 с одинаковым числом тех и других символов.

Решение. Определим множества, задающие грамматику:  $VT = \{0, 1\}$ ;  $VN = \{S\}$ ;  $P = \{S \rightarrow 0S1, S \rightarrow 1S0, S \rightarrow \varepsilon, S \rightarrow S01, S \rightarrow S10\}$ .

Пример 4. Описать язык, порождаемый следующими правилами:  $S \rightarrow bSS \mid a$ .

Решение. Построим несколько терминальных цепочек, применяя заданные правила вывода:

$S \rightarrow a;$

$S \rightarrow bSS \rightarrow baa;$

$S \rightarrow bSS \rightarrow bbSSS \rightarrow bbaaa;$

$S \rightarrow bSS \rightarrow bbSSbSS \rightarrow bbaabaa;$

$S \rightarrow bSS \rightarrow bbSSbSS \rightarrow bbbSSbSSbbSSbSS \rightarrow \dots$

Из полученных цепочек видно, что:

а) цепочки всегда начинаются с терминала  $b$ , кроме аксиомы, и заканчиваются терминалом  $a$ ;

б) количество терминалов  $a$  в любой цепочке всегда на 1 больше, чем  $b$ .

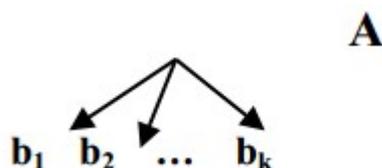
Исходя из этих выводов, определим язык, порождаемый заданными правилами, следующим образом:

$L(G) = \{a \mid \alpha \in \{a, b\}^*\}, |\alpha| = |b| + 1$ , причем цепочки начинаются с терминала  $b$  и заканчиваются терминалом  $a$ .

### Грамматический разбор

В КС-грамматике может быть несколько выводов, эквивалентных в том смысле, что в них применяются одни и те же правила к одинаковым в промежуточном выводе цепочкам, но в различном порядке. Например, для грамматики  $G$  с правилами вывода  $S \rightarrow ScS \mid b \mid a$  возможны следующие эквивалентные выводы терминальной цепочки  $acb$ :  $S \rightarrow ScS \rightarrow Scb \rightarrow acb$  и  $S \rightarrow ScS \rightarrow acS \rightarrow acb$ .

Деревом вывода цепочки  $x$  в КС-грамматике  $G = (VT, VN, P, S)$  называется упорядоченное дерево, каждая вершина которого помечена символом из множества  $V \cup VN \cup \{\varepsilon\}$  так, что каждому правилу  $A \rightarrow b_1b_2b_3 \dots b_k$ , используемому при выводе цепочки  $x$ , в дереве вывода соответствует поддереву с корнем  $A$  и прямыми потомками  $b_1, b_2, b_3, \dots, b_k$ , используемому при выводе цепочки  $x$ , в дереве вывода соответствует поддереву с корнем  $A$  и прямыми потомками  $b_1, b_2, b_3, \dots, b_k$ , которое приведено на рисунке:



В силу того, что цепочка  $x \in L(G)$  выводится из аксиомы  $S$ , корнем вывода всегда является аксиома. Внутренние узлы вывода соответствуют нетерминальным символам. Концевые вершины дерева вывода - листья - это вершины, не имеющие потомков. Такие вершины соответствуют либо терминалам, либо пустым символам ( $\epsilon$ ) при условии, что среди правил грамматики имеются правила с пустой правой частью. При чтении слева направо концевые вершины дерева вывода образуют цепочку  $x$ .

Дерево вывода часто называют деревом грамматического разбора, или синтаксическим деревом, а процесс построения дерева вывода - грамматическим разбором (синтаксическим анализом). Одной цепочке языка может соответствовать больше одного дерева, так как эта цепочка может иметь разные выводы, порождающие разные деревья.

КС-грамматика  $G = (VT, VN, P, S)$  называется неоднозначной (неопределенной), если существует цепочка  $x \in L(G)$ , имеющая два или более дерева вывода.

Рассмотрим пример.

Пусть даны две КС-грамматики:

$G_1 = (VT, VN, P_1, S)$ ,  $VN = \{S\}$ ,

$P_1 = \{S \rightarrow S+S \mid S \mid S \cdot S \mid (S) \mid a\}$ ;

$G_2 = (VT, VN, P_2, S)$ ,  $VN = \{S, A, B\}$ ,

$P_2 = \{S \rightarrow S+A \mid A \mid A, A \rightarrow A \cdot B \mid A / B \mid B, B \rightarrow a \mid (S)\}$ , содержащие в множестве

терминальных символов знаки операций, круглые скобки и символ  $a$ . Определить, являются ли грамматики однозначными. Если какая-либо из них неоднозначна, привести пример цепочки, для которой существует два различных дерева вывода.

Решение: Грамматика  $G_1$  является неоднозначной, так как она имеет правила с правой частью, содержащей два вхождения нетерминала  $S$  и знак арифметической операции. Построим два дерева вывода для простейшей цепочки  $a + a + a$ :



Грамматика  $G_2$  является однозначной, так как не содержит правил с двойным вхождением нетерминального символа. Так, для цепочки  $a + a + a$  она имеет только одно дерево вывода.

В некоторых случаях неоднозначность в грамматиках может устраняться путем эквивалентных преобразований. Однако в общем случае проблема устранения неоднозначности неразрешима. На практике от неоднозначности избавляются путем задания словесных ограничений, называемых контекстными условиями языка, которые включаются в его семантику.

Различают две стратегии грамматического разбора: восходящую и нисходящую, которые соответствуют способу построения синтаксических деревьев. При нисходящей стратегии разбора дерево строится от корня аксиомы вниз, к терминальным вершинам. Главной задачей при нисходящем разборе является выбор того правила, которое следует применить на рассматриваемом шаге. При восходящем разборе дерево строится от терминальных вершин к корню дерева (аксиоме).

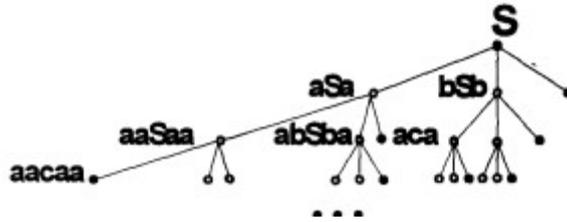
Преобразование цепочки, обратное порождению, называется редукцией.

Представление грамматики в виде графа

Дерево грамматического разбора не следует путать с представлением грамматики в

видеграфа. Граф грамматики в качестве вершин содержит сентенциальные формы (любые цепочки, выводимые из аксиомы).

Рассмотрим представление грамматики  $G$  в виде графа:  $G = (VT, VN, P, S)$ , в которой  $VT = \{a, b, c\}$ ,  $VN = \{S\}$ ,  $P = \{S \rightarrow aSa \mid bSb \mid c\}$ .



### Преобразования КС-грамматик

Часто требуется изменить грамматику таким образом, чтобы она удовлетворяла определенным требованиям, не изменяя при этом порождаемый грамматикой язык. Для этого используются эквивалентные преобразования КС-грамматик, некоторые из которых рассмотрены ниже.

Удаление правил вида  $A \rightarrow B$

Преобразование первого типа состоит в удалении правил  $A \rightarrow B$ , или  $\langle \text{нетерминал} \rangle \rightarrow \langle \text{нетерминал} \rangle$ .

Покажем, что для любой КС-грамматики можно построить эквивалентную грамматику, не содержащую правил вида:  $A \rightarrow B$ , где  $A$  и  $B$  - нетерминальные символы.

Пусть имеется КС-грамматика  $G = (VT, VN, P, S)$ , где множество нетерминалов  $VN = \{A_1, A_2, \dots, A_n\}$ . Разобьем  $P$  на два непересекающихся множества:  $P = P_1 \cup P_2$ . В  $P_1$  включены все правила вида  $A_i \rightarrow A_k$ , в  $P_2$  включены все остальные правила, т.е.  $P_2 = P \setminus P_1$ . Затем для каждого  $A_i$  определим множество правил  $P(A_i)$ , включив в него все такие правила  $A_i \rightarrow \phi$ , что  $A_i \rightarrow^* A_j$  и  $A_j \rightarrow \phi$ , где  $A_j \rightarrow \phi \in P_2$ . Построим эквивалентную КС-грамматику  $G_\alpha = (VT, VN, P_\alpha, S)$ , в которой множества терминальных и нетерминальных символов, а также аксиом совпадают с теми же объектами исходной грамматики  $G$ , а множество правил  $P_\alpha$  получено объединением правил множества  $P_2$  и правил  $P(A_i)$  для всех  $1 \leq i \leq n$ :

Пример. Пусть задана грамматика  $G$  со следующими правилами вывода  $S \rightarrow aFb \mid A$ ;  $A \rightarrow aA \mid B$ ;  $B \rightarrow aSb \mid S$ ;  $F \rightarrow bc \mid bFc$ . Построим множества правил  $P_2, P(S), P(A), P(B), P(F)$ .

Определим правила для  $P_2$ :  $P_2 = \{S \rightarrow aFb; A \rightarrow aA; B \rightarrow aSb; F \rightarrow bc \mid bFc\}$ . Определим правила для  $P(S)$ :  $S \Rightarrow A \Rightarrow B$  или  $S \Rightarrow^* A$ ;  $S \Rightarrow B$ , где  $\Rightarrow^*$  обозначает непосредственную выводимость.  $P(S) = \{S \rightarrow aA; S \rightarrow aSb\}$ . Определим правила для  $P(A)$ :  $A \Rightarrow B \Rightarrow S$  или  $A \Rightarrow^* B$ ;  $A \Rightarrow S$ .  $P(A) = \{A \rightarrow aSb; A \rightarrow aFb\}$ .

Определим правила для  $P(B)$ :  $B \Rightarrow S \Rightarrow A$  или  $B \Rightarrow^* S$ ;  $B \Rightarrow^* A$ .  $P(B) = \{B \rightarrow aFb; B \rightarrow aA\}$ .

Определим правила для  $P(F)$ : так как непосредственно выводимых нетерминалов не существует, то  $P(F) = \emptyset$ .

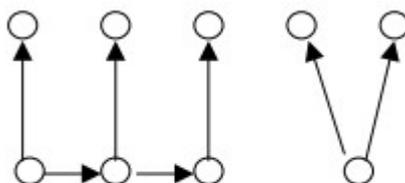
Объединив полученные правила, можно записать грамматику  $G_\alpha$ , эквивалентную исходной:

$S \rightarrow aFb \mid aSb \mid aA$ ;  $A \rightarrow aA \mid aSb \mid aFb$ ;  $B \rightarrow aA \mid aSb \mid aFb$ ;  $F \rightarrow bc \mid bFc$ .

### Графическая модификация метода

Аналитическое преобразование по рассмотренному алгоритму оказывается довольно сложным. При автоматизированном преобразовании грамматик проще применить графическую модификацию этого метода. С этой целью каждому нетерминальному символу и каждой правой части правил из множества  $P_2$  поставлена в соответствие вершина графа.

Из вершины с меткой  $U$  в вершину с меткой  $V$  направлено ребро, если в грамматике существует правило  $U \rightarrow V$ .



В эквивалентную грамматику будут включены правила вида  $A \rightarrow w$ ,  $A \in VN$ ;  $w \in (VT \cup VN)^*$ , если из вершины с меткой  $A$  существует путь в вершину с меткой  $w$ .

$S \rightarrow aFb \mid aSb \mid aA$ ;

$A \rightarrow aA \mid aSb \mid aFb$ ;

$B \rightarrow aA \mid aSb \mid aFb$ ;

$F \rightarrow bc \mid bFc$ .

Получено то же множество правил  $P$ , что и аналитическим методом.

Построение неукорачивающей грамматики

Грамматика, не содержащая правил с пустой правой частью, называется неукорачивающей грамматикой.

В грамматике с правилами вида  $A \rightarrow \varepsilon$  длина выводимой цепочки при переходе от  $k$ -го шага к  $(k+1)$ -му уменьшается. Поэтому грамматики с правилами вида  $A \rightarrow \varepsilon$  называются укорачивающими. Восходящий синтаксический разбор в укорачивающих грамматиках сложнее по сравнению с разбором в неукорачивающих грамматиках, т.к. при редукции необходимо отыскать такой фрагмент входной цепочки, в которую можно вставить пустой символ.

Покажем, что для произвольной КС-грамматики, порождающей язык без пустой цепочки, можно построить эквивалентную неукорачивающую КС-грамматику. Построим множество всех нетерминальных символов грамматики  $G=(VT, VN, P, S)$ , из которых выводится пустая цепочка, выделив следующие множества:

$W_1 = \{A \mid A \rightarrow \varepsilon \in P\}$ ,

$W_{n+1} = W_n \cup \{B \mid B \rightarrow \phi \in P, \phi \in W_n^*\}$ .

Затем найдем множество правил эквивалентной грамматики в два этапа:

а) удалив из множества  $P$  исходной грамматики правила с пустой правой частью  $P_1' = P \setminus \{A \rightarrow \varepsilon \mid A \rightarrow \varepsilon \in P\}$ ;

б) получив новые правила  $A \rightarrow \phi'$  после удаления из каждого правила исходной грамматики  $A \rightarrow \phi \in P$  те нетерминалы, которые вошли в множество  $W_n$  по правилу:  $P_1'' = \{A \rightarrow \phi' \mid A \rightarrow \phi \in P'; \phi = \phi_1 B \phi_2 \mid B \in W_n; \phi' = \phi_1 B \phi_2\}$ .

Повторив п.б) для каждого нетерминала, принадлежащего множеству  $W_n$ , получим эквивалентную грамматику  $G = (VT, VN, P_2, S)$ .

Пример. Пусть задана грамматика  $G$  со следующими правилами вывода:  $S \rightarrow AbA \mid cAb \mid Bb$ ;  $A \rightarrow aAb \mid \varepsilon$ ;  $B \rightarrow AA \mid a$ . Необходимо:

1) построить множество нетерминалов, из которых выводится  $\varepsilon$ ;

2) построить неукорачивающую грамматику, эквивалентную исходной.

Для того чтобы построить множество всех нетерминалов грамматики, из которых выводится пустая цепочка, выделим следующие множества:

$W_1 = \{A \mid A \rightarrow \varepsilon \in P\}$ ;

$W_{m+1} = W_m \cup \{B \mid B \rightarrow \phi \in P, \phi \in W_m^*\}$ .

Так как мы имеем правило  $A \rightarrow \varepsilon \in P$ , то можно построить множество  $W_1 = \{A\}$ , включающее нетерминал  $A$ .

Построим множество  $W_2$ . С нетерминалом  $A$  связан нетерминал  $B$ , т.е. существует правило  $B \rightarrow AA$  и  $A \in W_1$ . Следовательно,  $W_2 = \{A, B\}$ .  $W_3 = W_2$ , т.к. множество  $W_3 = \{B \mid B \rightarrow \phi \in P, \phi \in W_2^*\}$  является пустым.

Исключив правило, содержащее пустую цепочку в правой части, получим неукорачивающую грамматику  $G_1$  следующего вида:

$S \rightarrow AbA \mid cAb \mid Bb \mid bA \mid Ab \mid cb \mid b$ ;

$$A \rightarrow aAb \mid ab;$$

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

### Построение грамматики с продуктивными нетерминалами

Нетерминальный символ  $A$  называется непродуктивным (непроизводящим), если он не порождает ни одной терминальной цепочки, т.е. не существует вывода  $A \rightarrow^* x$ , где  $x \in V^*T$ .

Поэтому представляет интерес удаление из грамматики всех непродуктивных нетерминальных символов. Рассмотрим, как для произвольной КС-грамматики можно построить эквивалентную КС-грамматику, все нетерминальные символы которой продуктивны. С этой целью выделяется множество  $W_1 = \{A \mid A \rightarrow \phi \in P, \phi \in V^*T\}$ . Затем строится множество  $W_1, W_2, \dots, W_{n+1}$  по следующим правилам:  $W_{n+1} = W_n \cup \{B \mid B \rightarrow x \in P, x \in (VT \cup W_n)^*\}$

Пусть задана грамматика  $G$  со следующими правилами вывода:  $S \rightarrow SA \mid BSb \mid bAb; A \rightarrow aSa \mid bb; B \rightarrow bBb \mid BaA$ . Построим множество продуктивных нетерминалов:

$$W_1 = \{A\}, \text{ т.к. в множестве } P \text{ есть правило } A \rightarrow bb;$$

$$W_2 = \{A, S\}, \text{ т.к. имеется правило } S \rightarrow bAb \text{ и } A \in W_1;$$

$$W_3 = W_2.$$

Все символы множества  $V_N \setminus W_n$  являются непродуктивными, не используются в выводе никакой терминальной цепочки и их можно удалить из грамматики вместе со всеми правилами, в которые они входят. Грамматика  $G_1$ , эквивалентная исходной грамматике, будет включать следующее множество правил:

$$S \rightarrow SA \mid bAb; A \rightarrow aSa \mid bb.$$

В грамматике  $G_1$  все нетерминалы продуктивны.

### Построение грамматики, аксиома которой зависит от всех нетерминалов

Существует еще один тип нетерминальных символов, которые можно удалять из грамматики вместе с правилами, в которые они входят. Например, в грамматике  $G$ , заданной множеством правил  $P: S \rightarrow aSb \mid ba; A \rightarrow aAa \mid bba$ , нетерминал  $A$  не участвует ни в каком выводе, т.к. из аксиомы нельзя вывести цепочку, содержащую этот нетерминал. Поэтому из заданной грамматики можно удалить нетерминал  $A$ . Рассмотрим, как для произвольной КС-грамматики можно построить эквивалентную КС-грамматику, аксиома которой зависит от всех нетерминальных символов.

Для этого построим множество нетерминалов, от которых зависит аксиома. С этой целью выделим множества  $W_1, W_2, \dots, W_{n+1}$  по следующим правилам:

$$W_1 = \{S\},$$

$$W_{n+1} = W_n \cup \{B \mid A \rightarrow xBy \in P, A \in W_n\}.$$

Нетерминал  $B \in W_n$  тогда и только тогда, когда аксиома  $S$  зависит от  $B$ . Все нетерминалы, не содержащиеся в  $W_n$ , можно удалить из грамматики вместе с правилами, в которые они входят.

Пример. Пусть дана КС-грамматика  $G$ :

$$S \rightarrow abS \mid ASa \mid ab; A \rightarrow abAa \mid ab; B \rightarrow bAab \mid bB.$$

Найдем нетерминалы, от которых зависит аксиома:

$$W_1 = \{S\},$$

$$W_2 = \{S, A\}, \text{ т.к. имеется правило } S \rightarrow ASa \text{ и } S \in W_1;$$

$$W_3 = W_2.$$

Эквивалентная грамматика  $G_1$ , аксиома которой зависит от всех нетерминальных символов:

$$S \rightarrow abS \mid ASa \mid ab; A \rightarrow abAa \mid ab.$$

Удаление правил с терминальной правой частью

Пусть в грамматике  $G$  имеется с терминальной правой частью  $A \rightarrow \beta$ , где  $\beta \in V^*T$ . Тогда любой вывод с использованием этого правила имеет вид:  $S \xrightarrow{*} xAy \rightarrow x\beta y$ . Здесь нетерминал  $A$  в сентенциальной форме  $xAy$  появился на предыдущем шаге вывода  $B \rightarrow uAv$ .

Если в это правило вместо  $A$  подставить  $\beta$ , получим правило  $B \rightarrow u\beta v$ . Тогда длина вывода некоторой цепочки сократится на один шаг. Следовательно, для того чтобы удалить терминальное правило грамматики  $A \rightarrow \beta$ , необходимо учесть следующие правила:

а) если для нетерминала  $A$  больше нет правил, тогда во всех правых частях  $A$  заменяется на  $\beta$ ;

б) если для нетерминала  $A$  есть другие правила, тогда добавляются новые правила, в которых  $A$  заменяется на  $\beta$ .

Пример. Пусть дана КС-грамматика  $G$ :

$S \rightarrow aSb \mid bAa \mid B$ ;  $A \rightarrow AVa \mid b \mid \varepsilon$ ;  $B \rightarrow ab \mid ba$ .

Удалим правила для нетерминала  $B$ . Тогда эквивалентная грамматика  $G_1$  будет включать следующие правила:

$S \rightarrow aSb \mid bAa \mid ab \mid ba$ ;  $A \rightarrow Aaba \mid Abaa \mid b \mid \varepsilon$ .

Построение эквивалентной праворекурсивной КС-грамматики

Некоторые специальные методы грамматического разбора неприменимы к леворекурсивным или праворекурсивным грамматикам, поэтому рассмотрим устранение левой или правой рекурсии. В общем случае избавиться от рекурсии в правилах грамматики невозможно, т.к. бесконечные языки порождаются грамматиками с конечным числом правил только благодаря рекурсии. Поэтому можно говорить лишь о преобразовании одного вида рекурсии в другой. Рассмотрим, как для любой леворекурсивной КС-грамматики построить эквивалентную праворекурсивную КС-грамматику.

Пусть нетерминал  $A$  - леворекурсивен, т.е. для него имеются правила следующего вида:  $A \rightarrow Ax_1 \mid Ax_2 \mid \dots \mid Ax_r \mid w_1 \mid \dots \mid w_k$ , где  $x_i$  и  $w_j$  - цепочки над множеством  $V_T \cup V_N$ . Введем дополнительные нетерминалы  $B$  и  $D$  и указанные правила заменим на эквивалентные им:

$A \rightarrow AB \mid D$ ;

$B \rightarrow x_1 \mid x_2 \mid \dots \mid x_r$ ;

$D \rightarrow w_1 \mid w_2 \mid \dots \mid w_k$ .

В результате замены получим вывод, который имеет вид:

$A \rightarrow AB \rightarrow AB^2 \rightarrow AB^3 \rightarrow \dots \rightarrow AB^* \rightarrow DB^*$ .

Таким образом, для нетерминала  $A$  можно определить эквивалентные правила:

$A \rightarrow DK$ ;

$K \rightarrow BK \mid \varepsilon$ .

Выполняя подстановку  $D$  и  $B$  в эти правила, получим следующие праворекурсивные правила:

$A \rightarrow w_1K \mid w_2K \mid \dots \mid w_kK$ ;

$K \rightarrow x_1K \mid x_2K \mid \dots \mid x_rK \mid \varepsilon$ .

Пример. Пусть задана грамматика  $G$  со следующими правилами вывода:  $S \rightarrow Sa \mid ba$ .

Требуется построить эквивалентную ей праворекурсивную грамматику. Для решения задачи воспользуемся рассмотренным выше алгоритмом. В исходной грамматике один леворекурсивный нетерминал  $S$ . Построим для него вывод:

$S \rightarrow SB \mid D$ ;

$B \rightarrow a$ ;  $D \rightarrow ba$ . Вывод из  $S$  имеет вид:  $S \rightarrow SB \rightarrow SBB \rightarrow \dots \rightarrow DB^*$ .

Запишем эквивалентные правила для  $S$ :

$S \rightarrow DK$ ;  $K \rightarrow BK \mid \varepsilon$ .

Подставим в эти правила  $B$  и  $D$  и получим эквивалентную праворекурсивную грамматику  $G_1$ :

$S \rightarrow baK; K \rightarrow aK \mid \varepsilon.$

Рассмотрим вывод цепочки баааа в исходной леворекурсивной грамматике G:

$S \rightarrow Sa \rightarrow Saa \rightarrow Saaa \rightarrow баааа.$

Эту же цепочку можно вывести в эквивалентной праворекурсивной грамматике G1:

$S \rightarrow baK \rightarrow бааK \rightarrow баааK \rightarrow бааааK \rightarrow баааа.$

## 5.5. ТЕОРИЯ АВТОМАТОВ

Понятие автомата. Типы автоматов

Автомат - это алгоритм, определяющий некоторое множество и, возможно, преобразующий его в другое множество. Неформальное описание автоматов выглядит следующим образом: автомат имеет входную ленту, управляющее устройство с конечной памятью для хранения номера состояния, а также может иметь вспомогательную (рабочую) и выходную ленты.

Существует два типа автоматов:

- 1) распознаватели - автоматы без выхода, которые распознают, принадлежит ли входная цепочка заданному множеству L;
- 2) преобразователи - автоматы с выходом, которые преобразуют входную цепочку x в цепочку y при условии, что  $x \in L$ .

Входную ленту можно рассматривать как линейную последовательность ячеек, причем каждая ячейка может хранить один символ из некоторого конечного входного алфавита.

Лента автомата бесконечна, но занято на ней в каждый момент времени только конечное число ячеек. Граничные слева и справа от занятой области ячейки могут занимать специальные концевые маркеры. Маркер может стоять только на одном конце ленты или может отсутствовать вообще.

Входная головка в каждый момент времени читает (обозревает) одну ячейку входной ленты. За один такт работы автомата входная головка может сдвинуться на одну ячейку вправо или остаться на месте, при этом она выполняет только чтение, т.е. в ходе работы автомата символы в ячейках входной ленты не меняются.

Рабочая лента - это вспомогательное хранилище информации. Данные с нее могут читаться автоматом, могут и записываться на нее. Управляющее устройство - это программа, управляющая поведением автомата. Оно представляет собой конечное множество состояний вместе с отображением, описывающим, как меняются состояния в соответствии с текущим входным символом, читаемым входной головкой, и текущей информацией, извлеченной с рабочей ленты. Управляющее устройство также определяет направление сдвига рабочей головки и то, какую информацию записать на рабочую ленту.

Автомат работает, выполняя некоторую последовательность рабочих тактов. В начале такта читается входной символ и исследуется информация на рабочей ленте. Затем, в зависимости от прочитанной информации и текущего состояния, определяются действия автомата:

- 1) входная головка сдвигается вправо или стоит на месте;
- 2) на рабочую ленту записывается некоторая информация;
- 3) изменяется состояние управляющего устройства;
- 4) на выходную ленту (если она есть) записывается символ.

Поведение автомата удобно описывать в терминах конфигурации автомата, которая включает в себя:

- а) состояние управляющего устройства;
- б) содержимое входной ленты с положением входной головки;
- в) содержимое рабочей ленты вместе с положением рабочей головки;
- г) содержимое выходной ленты, если она есть.

Управляющее устройство может быть недетерминированным. В таком случае для каждой

конфигурации существует конечное множество возможных следующих тактов, любой из которых автомат может сделать, исходя из этой конфигурации. Управляющее устройство будет детерминированным, если для каждой конфигурации будет возможно только один следующий такт.

Существуют следующие типы автоматов:

- 1) машина Тьюринга (МТ);
- 2) линейно-ограниченный автомат (ЛОА);
- 3) автомат с магазинной памятью (МП-автомат);
- 4) конечный автомат (КА).

Сложность рабочей ленты определяет сложность автомата. Так, например:

- 1) машина Тьюринга имеет неограниченную в обе стороны ленту;
- 2) у линейно-ограниченного автомата длина рабочей ленты представляет собой линейную функцию длины входной цепочки;
- 3) у МП-автомата рабочая лента работает по принципу магазина LIFO;
- 4) у конечного автомата рабочая лента отсутствует.

Формальное определение автомата

Неинициальный автомат - это пятерка вида  $A = (K, X, Y, \delta, \gamma)$ , где

$K$  - множество состояний (алфавит состояний);

$X$  - входной алфавит;

$Y$  - выходной алфавит;

$\delta$  - функция переходов, задающая отображение  $K \cdot X \rightarrow K$ ;

$\gamma$  - функция выходов, задающая отображение  $K \cdot X \rightarrow Y$ .

Функционирование автомата можно задать множеством команд вида  $qx \rightarrow py$ , где  $q$  и  $p \in K$ ,  $x \in X$ ,  $y \in Y$ .

Пусть на некотором такте  $t_i$  управляющее устройство находится в состоянии  $q$ , а из входной ленты читается символ  $x$ . Если в множестве команд есть команда  $qx \rightarrow py$ , то на такте  $t_i$  на выходную ленту записывается символ  $y$ , а к следующему такту  $t_{i+1}$  управляющее устройство перейдет в состояние  $p$ , т.е.:

$$y(t) = \gamma(q(t), x(t)), q(t+1) = \delta(q(t), x(t)).$$

Если же команда  $qx \rightarrow py$  отсутствует, то автомат оказывается заблокированным и не реагирует на символ, принятый в момент  $t_i$ , а также перестает воспринимать символы в последующие моменты времени.

В соответствии с определением неинициального автомата в начальный момент состояние автомата может быть произвольным.

Если зафиксировано некоторое начальное состояние, то такой автомат называется инициальным, т.е.  $q(0) = q_0$ .

Инициальный автомат - это шестерка вида  $A = (K, X, Y, \delta, \gamma, q_0)$ , где

$K$  - множество состояний (алфавит состояний);

$X$  - входной алфавит;

$Y$  - выходной алфавит;

$\delta$  - функция переходов (отображение  $K \cdot X \rightarrow K$ );

$\gamma$  - функция выходов (отображение  $K \cdot X \rightarrow Y$ );

$q_0$  - начальное состояние.

Распознаватели. Языки и автоматы

Задача грамматического разбора заключается в нахождении вывода цепочки в заданной грамматике и определения дерева вывода этой цепочки. Языки могут быть заданы

двумя способами:

- 1) грамматиками (порождающее средство языка);
- 2) автоматами (распознающее средство языка). Различным по сложности автоматам соответствуют разные типы языков. Простейшим типом автоматов являются конечные автоматы. Конечный автомат имеет входную ленту, с которой за один такт может быть считан один входной символ. Возврат по входной ленте не допускается.

Конечным автоматом называется пятерка вида  $A = (K, \Sigma, \delta, p_0, F)$ , где

$K$  - конечное множество состояний;

$\Sigma$  - алфавит;

$\delta$  - функция переходов;

$p_0$  - начальное состояние;

$F$  - множество заключительных состояний.

Автомат можно определить как формальную систему через состояния, через символы, которые пишутся (читаются) с ленты или с нескольких лент, и через набор команд.

Конечный автомат можно представить графом, таблицей переходов, командами, а также матрицей переходов.

### Регулярные множества

Регулярные множества образуют класс языков, имеющих важное значение для теории формальных языков. Рассмотрим несколько методов задания языков, каждый из которых определяет регулярные множества. Среди них - регулярные выражения, праволинейные грамматики, детерминированные и недетерминированные конечные автоматы.

Пусть  $\Sigma$  - некоторый алфавит. Регулярное множество в алфавите  $\Sigma$  определяется рекурсивно следующим образом:

- 1)  $\emptyset$  - пустое множество;
- 2)  $\{\epsilon\}$  - множество из пустой цепочки;
- 3)  $\{a\}$  - регулярное множество для каждого элемента  $a \in \Sigma$ ;
- 4) если  $P$  и  $Q$  - регулярные множества в алфавите  $\Sigma$ , то регулярными являются множества:

а)  $P \cup Q$

б)  $PQ$

в)  $P^*$ .

Других регулярных множеств в алфавите  $\Sigma$  нет. Таким образом, некоторое множество цепочек в заданном алфавите  $\Sigma$  называется регулярным тогда и только тогда, когда либо оно является одним из множеств:  $\emptyset$ ,  $\{\epsilon\}$  или  $\{a\}$  для некоторого  $a \in \Sigma$ , либо его можно получить из этих множеств применением конечного числа операций объединения, конкатенации и итерации. Для каждого регулярного множества существует по крайней мере одно регулярное выражение, обозначающее это множество. Язык, распознаваемый конечным автоматом, - это множество цепочек, читаемых автоматом при переходе из начального состояния в одно из заключительных состояний:

$$L(A) = \{a_1 a_2 \dots a_n \mid p_0 a_1 \rightarrow p_1, p_1 a_2 \rightarrow p_2, \dots, p_{n-1} a_n \rightarrow p_n; p_n \in F\}.$$

Множество называется регулярным, если существует конечный детерминированный автомат, распознающий его.

### Операции над регулярными языками

Так как произвольному конечному автомату однозначно соответствует детерминированный конечный автомат, операции над конечными автоматами эквивалентны операциям над регулярными множествами, или регулярными языками. Известно, что для произвольного конечного автомата можно построить эквивалентный автомат без циклов в начальных и (или) конечных состояниях.

Теорема. Для произвольного конечного автомата существует конечный автомат

безциклов в начальном состоянии.

Доказательство. Пусть  $A = (K, \Sigma, \delta, p_0, F)$  - произвольный конечный автомат. Построим автомат:

$$A_1 = (K \cup \{q_0\}, \Sigma, \delta \cup \{q_0 a \rightarrow p_i \mid p_0 a \rightarrow p_i \in \delta\}, q_0, F \cup \{q_0 \mid p_0 \in F\}).$$

Любая цепочка  $x = a_1 a_2 \dots a_k$  принадлежит языку  $L(A)$  тогда и только тогда, когда существует следующая последовательность команд автомата  $A$ :

$$p_0 a_1 \rightarrow p_1; p_1 a_2 \rightarrow p_2; \dots; p_{h-1} a_h \rightarrow p_h, p_h \in F$$

и соответствующая ей последовательность команд автомата  $A_1$ :

$$q_0 a_1 \rightarrow p_1; p_1 a_2 \rightarrow p_2; \dots; p_{k-1} a_k \rightarrow p_h.$$

Таким образом, имеем:  $A = A_1$ .

Теорема. Для произвольного конечного автомата существует эквивалентный автомат без циклов в заключительном состоянии.

Доказательство. Будем считать, что автомат не имеет циклов в начальном состоянии.

Сопоставим заданному произвольному конечному автомату  $A = (K, \Sigma, \delta, p_0, F)$  новый автомат  $A_1$ :  $A_1 = (K \cup \{f\}, \Sigma, \delta \cup \{q_j a \rightarrow f \mid p_j a \rightarrow p_i \in \delta \& p_i \in F\}, p_0, \{f\} \cup \{p_0 \mid p_0 \in F\})$ .

В результате имеем:  $A = A_1$ .

Теорема. Множество регулярных языков замкнуто относительно операций итерации, усеченной итерации, объединения, произведения, пересечения, дополнения и разности.

Доказательство. Для доказательства необходимо выполнить операции над соответствующими конечными автоматами и показать, что в результате таких преобразований построенный автомат допускает требуемый язык. Предполагается, что в автомате удалены циклы из начальных и заключительных состояний. Для выполнения перечисленных в теореме операций необходимо выполнить соответствующие преобразования над заданными автоматами.

1. Операция итерации реализуется удалением циклов из начальных и заключительных состояний и объединения полученных состояний. Объединение начального и заключительного состояний означает, что построенный автомат допускает пустую цепочку. Однократный переход из начального в заключительное состояние исходного автомата соответствует допуску цепочек языка  $L$ . Поскольку эти состояния объединены, автомат допускает цепочки языков  $LL, LLL$  и т.д., т.е. он распознает язык  $\{\varepsilon\} \cup L \cup L^2 \cup \dots = L^*$ .

2. Операция произведения над  $L(A_1)$  и  $L(A_2)$  выполняется с помощью двух преобразований: а) удаляются циклы из начального состояния  $A_2$  и заключительного состояния  $A_1$ ; б) каждому заключительному состоянию  $A_1$  ставим в соответствие свой экземпляр  $A_2$  и объединяем заключительные состояния  $A_1$  с начальным состоянием соответствующего экземпляра  $A_2$ .

3. Объединение  $L(A_1)$  и  $L(A_2)$  строится с помощью удаления циклов в начальных состояниях  $A_1$  и  $A_2$  и объединения полученных начальных состояний.

4. Усеченная итерация может быть построена двумя способами:

$$\text{а) } L(A_1)^+ = L(A_1)^* L(A_1), \text{ б) } L(A_1)^+ = L(A_1) L(A_1)^*.$$

5. Рассмотрим дополнение  $L(A_1)$  до  $\Sigma^*$ . Пусть автомат  $A_1$  детерминированный, тогда любая цепочка  $x = a_1 a_2 \dots a_n$  распознается по единственному маршруту:

$$p_0 a_1 \rightarrow p_1 \quad p_1 a_2 \rightarrow p_2$$

...

$$p_{n-1} a_n \rightarrow p_n, p_n \in F.$$

Автомат не распознает только те цепочки, которые:

1) либо представляют собой начальную часть цепочки  $a_1 a_2 \dots a_j$ , при чтении которой автомат переходит в состояние, не являющееся заключительным;

2) либо имеют вид  $y = a_1 a_2 \dots a_k b c_1 c_2 \dots c_m$  ( $k < n$ ), где начало цепочки  $a_1 a_2 \dots a_k$  совпадает с началом цепочки  $x \in L(A_1)$ , но за символом  $a_k$  стоит такой символ  $b$ , что автомат  $A_1$  его прочитать не может.

Поэтому для того чтобы построить автомат, распознающий дополнение языка,

необходимо:

а) все заключительные состояния сделать незаключительными, а незаключительные состояния - заключительными;

б) ввести дополнительное состояние, сделать его заключительным и из каждого состояния провести в это состояние такие дуги, каждая из которых соответствует символам алфавита, не читаемым в этом состоянии; в) в построенном дополнительном состоянии построить петли для всех символов алфавита, чтобы обеспечить чтение произвольного окончания цепочки  $s_1c_2 \dots s_n$ . 6. Разность  $L(A_1) \setminus L(A_2)$  строится в соответствии со следующим преобразованием:  $L(A_1) \setminus L(A_2) = L(A_1) \cap L(A_2)$ .

7. Операция пересечения строится в соответствии со следующим преобразованием:  $L(A_1) \cap L(A_2) = L(A_1) \cup L(A_2)$ . На основании этой теоремы можно строить конечные автоматы, последовательно синтезируя их на основе уже построенных автоматов.

### Автоматные грамматики

Линейные грамматики (праворекурсивные и леворекурсивные) называются автоматными грамматиками, так как языки, порождаемые ими, совпадают с языками, распознаваемыми конечными автоматами. Рассмотрим ряд теорем.

**Теорема.** Для каждой праволинейной грамматики существует эквивалентный конечный автомат.

**Доказательство.** Каждому нетерминалу произвольной праволинейной грамматики  $G$  поставим в соответствие одно состояние конечного автомата  $A$ . Добавим еще одно состояние - единственное конечное состояние. Состояние, соответствующее аксиоме, назовем начальным состоянием. Каждому правилу  $A \rightarrow aB$  поставим в соответствие команду  $Aa \rightarrow B$ , а каждому терминальному правилу  $A \rightarrow a$  поставим в соответствие команду  $Aa \rightarrow F$ .

Таким образом, выводу цепочки в грамматике  $S \Rightarrow a_1A_1 \Rightarrow a_1a_2A_2 \Rightarrow \dots \Rightarrow a_1a_2\dots a_{k-1}A_{k-1} \Rightarrow a_1a_2\dots a_k$  взаимно однозначно соответствует последовательность команд построенного автомата  $A$ :

$Aa_1 \rightarrow A_1; A_1a_2 \rightarrow A_2; \dots; A_{k-1}a_k \rightarrow F$ .

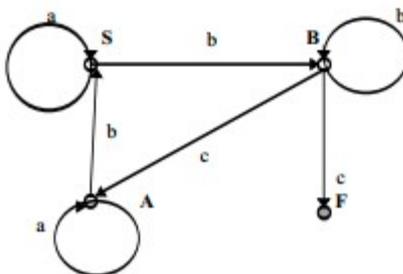
Таким образом, язык  $L(G) = L(A)$ .

**Пример.** Пусть для заданной грамматики  $G$  требуется построить конечный автомат.

$S \rightarrow aS \mid bB;$

$A \rightarrow aA \mid bS;$

$B \rightarrow bB \mid c \mid cA.$



**Решение.** Граф автомата будет иметь четыре вершины, три из них помечены нетерминальными символами грамматики  $S, A, B$ , четвертая вершина, помеченная символом  $F$ , является единственным заключительным состоянием. Начальным состоянием является вершина, соответствующая аксиоме  $S$ .

Каждому правилу грамматики поставим в соответствие команду конечного автомата:

$Sa \rightarrow S$  - в начальном состоянии при поступлении на вход терминального символа  $a$  автомат остается в том же состоянии  $S$ ;

$Sb \rightarrow B$  - из начального состояния при поступлении на вход терминала  $b$  автомат переходит в состояние  $B$ ;

$Bb \rightarrow B$  - в состоянии  $B$  при поступлении на вход терминала  $b$  автомат остается в том же состоянии  $B$ ;

$Bc \rightarrow F$  - из состояния  $B$  при поступлении на вход терминала  $c$  автомат переходит в заключительное состояние  $F$ ;

$Bc \rightarrow A$  - из состояния  $B$  при поступлении на вход терминала  $c$  автомат переходит в состояние  $A$ ;

$Aa \rightarrow A$  - в состоянии  $A$  при поступлении на вход терминала  $a$  автомат остается в этом же состоянии  $A$ ;

$Ab \rightarrow S$  - из состояния  $A$  при поступлении на вход терминала  $b$  автомат переходит в состояние  $S$ .

Полученный недетерминированный конечный автомат распознает цепочки языка, порождаемые праворекурсивной грамматикой  $G$ .

Теорема. Для произвольного конечного автомата существует эквивалентная праволинейная грамматика.

Доказательство. Каждому состоянию произвольного автомата поставлен в соответствие нетерминал грамматики, причем начальному состоянию будет соответствовать аксиома. Тогда для каждой команды  $Ac \rightarrow B$  в множество правил грамматики включим правило  $A \rightarrow cB$ , причем в случае, если  $B$  - заключительное состояние, добавим правило  $A \rightarrow c$ . Эквивалентность исходного конечного автомата и построенной грамматики очевидна.

Теорема. Для каждой леворекурсивной грамматики существует эквивалентный конечный автомат.

Доказательство. Каждому нетерминальному символу произвольной леворекурсивной грамматики поставим в соответствие состояние конечного автомата, причем состояние, соответствующее аксиоме  $S$ , сделаем заключительным. Добавим еще одно состояние  $N$  и сделаем его начальным состоянием. Каждому правилу грамматики  $A \rightarrow Ba$  поставим в соответствие команду автомата  $Ba \rightarrow A$ . Тогда каждому выводу в грамматике:

$S \Rightarrow A_1 a_1 \Rightarrow A_2 a_1 a_2 \Rightarrow \dots \Rightarrow A_{k-1} a_1 a_2 \dots a_{k-1} \Rightarrow a_1 a_2 \dots a_k$  однозначно соответствует следующая последовательность команд построенного автомата  $A$ :

$N a_1 \rightarrow A_{k-1}; \dots; A_2 a_{k-1} \rightarrow A_1; A_1 a_k \rightarrow S$ .

Таким образом, язык  $L(G) = L(A)$ .

Теорема. Для произвольного конечного автомата существует эквивалентная леворекурсивная грамматика.

Доказательство. Каждому состоянию произвольного автомата поставим в соответствие нетерминальный символ грамматики, добавим нетерминал  $S$  и сделаем его аксиомой. Каждой команде  $Aa \rightarrow B$  в множество правил включим соответствующее правило  $B \rightarrow Aa$ , причем в том случае, если  $B$  - заключительное состояние, дополнительно введем правило  $S \rightarrow Aa$ , а если  $A$  - начальное состояние, то дополнительно введем правило  $B \rightarrow a$ .

Тогда последовательности команд:

$A_0 a_1 \rightarrow A_1; A_1 a_2 \rightarrow A_2; \dots;$

$A_{k-1} a_k \rightarrow F$

соответствует следующий вывод:

$S \Rightarrow A_{k-1} a_k \Rightarrow \dots \Rightarrow A_1 a_2 a_3 \dots a_k \Rightarrow a_1 a_2 a_3 \dots a_k$ .

Важной особенностью автоматных грамматик является возможность представления их с помощью конечных графов. По графу грамматики легко отыскивается вывод нужной цепочки.

Любой вывод цепочки в автоматной грамматике соответствует пути в графе этой грамматики, который начинается из вершины  $S$  (вершины, помеченной аксиомой) и заканчивается в конечной вершине.

Пример. Построить конечный автомат, распознающий язык  $L(A) = \{(ab)^*\}$ .

Сначала построим некоторую грамматику  $G$ , которая бы порождала язык  $L(A)$ :

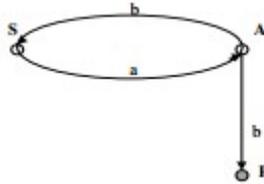
$S \rightarrow aA$ ;

$A \rightarrow bS \mid b$ .

Проверим, действительно ли эта грамматика порождает язык  $L(A)$ . Для этого построим несколько выводов возможных вариантов цепочек:

- 1)  $S \Rightarrow aA \Rightarrow ab$ ;
- 2)  $S \Rightarrow aA \Rightarrow abS \Rightarrow abaA \Rightarrow abab$ ;
- 3)  $S \Rightarrow aA \Rightarrow abS \Rightarrow abaA \Rightarrow ababS \Rightarrow ababaA \Rightarrow ababab$ ; ит.д.

Таким образом, грамматика  $G$  действительно порождает язык  $L(A)$ , следовательно, можно построить соответствующий этой грамматике конечный автомат. Для этого введем заключительное состояние  $F$ , начальное состояние соответствует аксиоме  $S$ .



Запишем преобразование правил вывода в команды:

$Sa \rightarrow A$  - из состояния  $S$  при поступлении на вход терминала  $a$  автомат переходит в состояние  $A$ ;

$Ab \rightarrow S$  - из состояния  $A$  при поступлении на вход терминала  $a$  автомат переходит в состояние  $S$ ;

$Ab \rightarrow F$  - из состояния  $A$  при поступлении на вход терминала  $b$  автомат переходит в заключительное состояние  $F$ .

Таким образом, построен недетерминированный конечный автомат, распознающий заданный язык  $L(G)$ .

#### Автоматы с магазинной памятью

Автомат с магазинной памятью (МП-автомат) имеет рабочую ленту, которая организована в виде магазина. МП-автомат - это семерка вида:

$M = (K, \Sigma, \Gamma, \delta, p_0, F, B_0)$ , где

$K$  - конечное множество состояний;

$\Sigma$  - алфавит;  $\Gamma$  - алфавит магазина;

$\delta$  - функция переходов;

$p_0$  - начальное состояние;

$F$  - множество заключительных состояний;

$B_0$  - символ из множества  $\Gamma$  для обозначения маркера дна магазина.

В общем случае данное определение соответствует недетерминированному автомату. В отличие от конечного автомата для произвольного МП-автомата нельзя построить эквивалентный детерминированный автомат.

Основное использование распознавательных средств задания языков состоит в построении алгоритмов грамматического разбора. Поэтому необходимо для произвольной КС-грамматики уметь строить эквивалентный МП-автомат. МП-автомат представляет интерес как средство разбора в КС-грамматиках произвольного вида. Этот факт сформулирован в следующей теореме.

**Теорема.** Языки, порождаемые КС-грамматиками, совпадают с языками, распознаваемыми МП-автоматами.

**Доказательство.** Существуют две стратегии разбора: восходящий и нисходящий разбор.

Восходящий разбор в МП-автомате

При восходящей стратегии необходимо найти основу и редуцировать ее к какому-нибудь нетерминалу в соответствии с правилами данной грамматики. Это можно сделать в том случае, если реализовать следующий алгоритм функционирования МП-автомата:

- 1) любой входной символ записывается в магазин;
- 2) если в верхушке магазина сформирована основа, совпадающая с правой частью правила, то она заменяется на нетерминал в левой части этого правила;
- 3) разбор заканчивается, если в магазине остается аксиома, а входная цепочка рассмотрена полностью.

В соответствии с этим алгоритмом для КС-грамматики  $G = (VT, VN, P, S)$  построим МПАвтомат:

$M = (K, VT, \Gamma, \delta, p_0, F, B_0)$ , где  $\Gamma = VT \cup VN \cup \{B_0\}$ ,

$K = \{p_0, F\}$ ,  $F = \{f\}$ .

Функция переходов  $\delta$  будет содержать следующие команды:

а)  $p_0, a, \varepsilon \rightarrow p_0, a$  - для любых  $a \in VT$ ;

б)  $p_0, \varepsilon, \phi' \rightarrow p_0, A$  - для всех правил  $A \rightarrow \phi \in P$ , где  $\phi'$  - зеркальное отображение  $\phi$ ;

в)  $p_0, \varepsilon, SB_0 \rightarrow f, B_0$ . В общем случае команда выглядит так:

$p_i, \sigma, \gamma \rightarrow p_j, \lambda$ , где  $p_i \in K$  - состояние автомата до выполнения команды,  $\sigma \in Vt$  - символ на входной ленте,  $\gamma \in \Gamma$  - символ верхушки магазина,  $p_j \in K$  - состояние автомата после выполнения команды,  $\lambda \in \Gamma$  - символ, который записывается в магазин.

Таким образом, любому выводу в грамматике  $G$  взаимно однозначно соответствует последовательность команд построенного МП-автомата. Обратное построение КС-грамматики по произвольному МП-автомату также возможно, но не представляет практического интереса.

Рассмотрим пример восходящей стратегии разбора.

Пусть дана грамматика  $G$ :

$S \rightarrow S+A \mid S/A \mid A A \rightarrow a \mid (S)$ ;

$VN = \{S, A\}$ ,  $VT = \{a, (, ), +, /\}$ .

Для заданной КС-грамматики  $G$  необходимо построить МП-автомат. Эквивалентный МП-автомат должен содержать следующие команды:

1. Команды переноса терминальных символов в магазин:

$p_0, a, \varepsilon \rightarrow p_0, a$  ;

$p_0, +, \varepsilon \rightarrow p_0, +$  ;

$p_0, /, \varepsilon \rightarrow p_0, /$  ;

$p_0, (, \varepsilon \rightarrow p_0, ($  ;

$p_0, ), \varepsilon \rightarrow p_0, )$ .

Эти команды обеспечивают занесение терминального символа из входной ленты в магазин.

2. Команды редукции по правилам грамматики:

$p_0, \varepsilon, A+S \rightarrow p_0, S$  ;

$p_0, \varepsilon, A/S \rightarrow p_0, S$  ;

$p_0, \varepsilon, A \rightarrow p_0, S$  ;

$p_0, \varepsilon, )S( \rightarrow p_0, A$  ;

$p_0, \varepsilon, a \rightarrow p_0, A$ .

Эти команды заменяют зеркальное отображение правила, полученного в верхушке магазина, на нетерминал в левой части данного правила грамматики.

3. Команды проверки на завершение разбора:

$p_0, \varepsilon, SB_0 \rightarrow f, B_0$ .

Разбор завершается, если в магазине остались аксиома и маркер дна магазина, а входная цепочка полностью рассмотрена. Подадим на вход автомата цепочку  $a/(a+a)$  и выполним разбор. Процесс разбора представлен в таблице 1.

Таблица 1

ВХОД	a / ( a + A )															
м											a	A				
а										+	+	+	)			
г							a	A	S	S	S	S	S	S		
а						(	(	(	(	(	(	(	(	(	A	
з				/	/	/	/	/	/	/	/	/	/	/	/	
н	$B_0$	a	A	S	S	S	S	S	S	S	S	S	S	S	S	S
н	$B_0$	$B_0$	$B_0$	$B_0$	$B_0$	$B_0$	$B_0$	$B_0$	$B_0$	$B_0$	$B_0$	$B_0$	$B_0$	$B_0$	$B_0$	$B_0$
р	$p_0$	$p_0$	$p_0$	$p_0$	$p_0$	$p_0$	$p_0$	$p_0$	$p_0$	$p_0$	$p_0$	$p_0$	$p_0$	$p_0$	$p_0$	f

## Нисходящий разбор в МП-автомате

На любом шаге нисходящего разбора должно применяться какое-либо правило. В начальный момент таким нетерминалом является аксиома. МП-автомат, выполняющий нисходящий разбор, работает по следующему алгоритму:

- 1) в начальный момент времени в магазин заносится аксиома:  
 $p_0, \varepsilon, \varepsilon \rightarrow p_1, S$ ;
- 2) для каждого правила  $A \rightarrow \phi \in P$  нетерминал в верхушке магазина заменяется на правую часть правила с помощью команды:  $p_1, \varepsilon, A \rightarrow p_1, \phi$ ;
- 3) для каждого терминала  $a \in VT$  выполняется сравнение символа на входной ленте с символом в верхушке магазина и его поглощение:  $p_1, a, a \rightarrow p_1, \varepsilon$ ;
- 4) разбор заканчивается по команде:  $p_1, \varepsilon, B_0 \rightarrow f, B_0$ .

Для грамматики, рассмотренной в предыдущем примере, разбор той же входной цепочки по нисходящей стратегии будет выполняться посредством следующего множества команд:

- 1) команда занесения аксиомы в магазин:  $p_0, \varepsilon, \varepsilon \rightarrow p_0, S$ ;
- 2) команды замены нетерминала правой частью правила:  
 $p_1, \varepsilon, S \rightarrow p_1, S+A$   
 $p_1, \varepsilon, S \rightarrow p_1, S/A$   
 $p_1, \varepsilon, S \rightarrow p_1, A$   
 $p_1, \varepsilon, A \rightarrow p_1, a$   
 $p_1, \varepsilon, A \rightarrow p_1, (S)$ ;

3) команды сравнения и поглощения символа с входной ленты и символа в верхушке магазина:

- $$p_1, a, a \rightarrow p_1, \varepsilon$$
- $$p_1, +, + \rightarrow p_1, \varepsilon$$
- $$p_1, /, / \rightarrow p_1, \varepsilon$$
- $$p_1, (, ( \rightarrow p_1, \varepsilon \quad p_1, ), ) \rightarrow p_1, \varepsilon;$$

4) команда завершения разбора:  $p_1, \varepsilon, B_0 \rightarrow f, B_0$ . Процесс разбора цепочки представлен в таблице 2.

ВХОД	a / ( a + a )																
м																	
а																	
г																	
а			S	a	a			(		S	A	a					
з			/	/	/	/		S	S	A	A	A	A	A	a		
и		S	A	A	A	A	A	)	)	)	)	)	)	)	)	)	)
н	B <sub>0</sub>																
р	p <sub>0</sub>	p <sub>1</sub>	f														

Рассмотренные выше МП-автоматы работают недетерминированно, то есть если цепочка принадлежит языку, порождаемому заданной грамматикой, то какой-то из вариантов функционирования автомата осуществит правильный разбор. Если же цепочка не принадлежит языку, то никакой из вариантов разбора не приведет к цели.

Отсутствие детерминированного эквивалентного автомата для произвольной КСграмматики означает невозможность построения универсальной простой однопроходной программы синтаксического анализа. Поэтому для эффективного разбора необходимо выделять специальные классы КС-грамматик, удовлетворяющие требованиям конкретных типов анализаторов. Если требуется выполнить разбор для произвольной КС-грамматики, то придется использовать детерминированную программную модель недетерминированного МП-автомата.

#### Понятие преобразователей

Автоматы с выходом называются преобразователями. В зависимости от вида функции, отображающей множество состояний и входных символов в множество выходных символов и новых состояний, а также от типа рабочей ленты различают разные виды преобразователей. Рассмотрим конечные автоматы-преобразователи.

Конечным преобразователем называется шестерка вида

$P = (K, X, Y, f, g, q_0)$ , где  $K$  - конечное множество состояний;

$X$  - входной алфавит;

$Y$  - выходной алфавит;

$f$  - функция переходов;

$g$  - функция выходов;

$q_0$  - начальное состояние.

Типы отображений  $f$  и  $g$  определяют различные виды автоматов. Если  $g$  - отображение  $K \cdot X$  в  $Y$ , то конечный преобразователь называется синхронным. В общем случае это отображение имеет вид  $K \cdot X \rightarrow Y^*$ .

Пусть  $P = (K, X, Y, f, g, q_0)$  - конечный преобразователь. Тогда отображение  $S(x) = g(q_0, x)$ , определенное для любой цепочки  $x \in X^*$ , называется конечным преобразователем.

Заметим, что для того чтобы выходную цепочку  $y$  можно было считать переводом входной цепочки  $x$ , цепочка  $x$  должна перевести преобразователь из начального состояния в заключительное.

#### Автоматы Мили и Мура

Автоматы Мили и Мура являются неинициальными автоматами. В отображении  $S(x) = g(q_0, x)$  зафиксируем начальное состояние  $q_0$ , в котором автомат находится в начальный момент времени. Оно существенно влияет на процесс конечного преобразования, т.к. определяет не только результирующую цепочку, но и множество входных цепочек.

Рассмотрим поведение инициальных автоматов, которые могут начинать работать из любого указанного состояния. Такой автомат получает на вход одну цепочку бесконечной длины и перерабатывает её. Реакция такого преобразователя на определенные воздействия непредсказуема, если неизвестно его начальное состояние. Поэтому необходимо решить две задачи, имеющие важное практическое значение:

- 1) определение того состояния автомата, в котором он находится в момент, начиная с которого исследуется его поведение;
  - 2) распознавание конечного состояния, в которое перешел автомат после завершения испытательной операции. Это состояние будет начальным для следующей серии испытаний.
- Эти задачи анализа получили название экспериментов по распознаванию состояния.

Автомат Мили

Автомат Мили - это пятерка вида  $M = (K, X, Y, f, g)$ , где:

$K$  - множество состояний автомата;

$X$  - входной алфавит;

$Y$  - выходной алфавит;

$f$  - функция переходов (отображение  $K \cdot X \rightarrow K$ );

$g$  - функция выходов (отображение  $K \cdot X \rightarrow Y$ ).

Как и любой другой автомат, автомат Мили можно представить в виде таблицы или графа. В графе переходов автомата Мили на дугах указываются через символ ‘/’ входные и выходные символы. Таблица переходов состоит из двух частей: в левой части записываются значения функции выходов, в правой части - значения функции переходов. Пример. Построим преобразователь, который распознает арифметические выражения, порожаемые грамматикой:

$$S \rightarrow a+S \mid a-S \mid +S \mid -S \mid a$$

и устраняет из этих выражений избыточные унарные операции. Например, выражение  $-a+-a+-a$  он переведет в  $-a-a+a$ . Во входном языке символ  $a$  представляет идентификатор и перед идентификатором допускается произвольная последовательность знаков унарных операции  $+$  и  $-$ . Заметим, что входной язык является регулярным множеством.

Пусть

$M = (K, X, Y, f, g)$ , где

1.  $K = \{q_0, q_1, q_2, q_3, q_4\}$ ,

2.  $X = \{a, +, -\}$ , 3.  $Y = X$ .

Преобразователь  $M$  начинает работу в состоянии  $q_0$  и, чередуя состояния  $q_0$  и  $q_4$  на входном символе « $-$ », определяет, четное или нечетное число знаков  $-$  предшествует первому символу  $a$ . Когда появляется  $a$ , преобразователь  $M$  переходит в состояние  $q_1$ , допуская вход, и выдает  $a$  или  $-a$  в зависимости от того, четно или нечетно число появившихся минусов. Для следующих символов  $a$  он подсчитывает, четно или нечетно число предшествующих минусов, с помощью состояний  $q_2$  и  $q_3$ . Единственное различие между парами  $q_2, q_3$  и  $q_0, q_4$  состоит в том, что если символу  $a$  предшествует четное число минусов, то первая из них выдает  $+a$ , а не только  $a$ .

Таблица переходов выглядит следующим образом:

K \ X	Y			K		
	a	+	-	a	+	-
q <sub>0</sub>	a	ε	ε	q <sub>1</sub>	q <sub>0</sub>	q <sub>4</sub>
q <sub>1</sub>	-	ε	ε	-	q <sub>2</sub>	q <sub>3</sub>
q <sub>2</sub>	+a	ε	ε	q <sub>1</sub>	q <sub>2</sub>	q <sub>3</sub>
q <sub>3</sub>	-a	ε	ε	q <sub>1</sub>	q <sub>3</sub>	q <sub>2</sub>
q <sub>4</sub>	-a	ε	ε	q <sub>1</sub>	q <sub>4</sub>	q <sub>0</sub>

Автомат Мура

Автомат Мура - это «пятерка» вида  $U = (K1, X, Y, f1, h)$ , где:

$K1$  - множество состояний автомата;

$X$  - входной алфавит;

$Y$  - выходной алфавит;

$f1$  - функция переходов (отображение  $K \cdot X \rightarrow K$ );

$h$  - функция выходов (отображение  $K \cdot X \rightarrow Y$ ).

При представлении автомата Мура графом дуги помечаются символами входного алфавита, а каждая вершина графа - состоянием и символом выходного алфавита. При формальном сравнении определений автоматов Мили и Мура может показаться, что автомат Мура может быть задан как входонезависимый автомат Мили, т.е. такой автомат

Мили, выходная функция которого удовлетворяет следующим условиям:  $\forall a \in X, \forall b \in X, \forall z \in Z (g(z, a) = g(z, b))$ . Однако это не соответствует способу функционирования автоматов Мура в соответствии с введенным определением. В автомате Мура реализована иная временная связь между переходами из одного состояния в другое и выходом, по сравнению с автоматом Мили, у которого выход, соответствующий некоторому входу и определенному состоянию, порождается во время перехода автомата в следующее состояние. У автомата Мура сначала порождается выход, а потом - переход в следующее состояние, причем выход определяется только состоянием автомата.

Равносильность автоматов Мили и Мура Равносильность заключается в том, что множество реакций этих автоматов совпадает:

$$L(M) = \{qz \mid qZ \in K\};$$

$$L(U) = \{ht \mid ht \in K1\};$$

$$L(M) = L(U).$$

Теорема. Для каждого автомата Мура можно построить равносильный автомат Мили.

Доказательство. Граф равносильного автомата Мили  $M$  можно получить в том случае, если каждому ребру автомата Мура  $U$  сопоставить ребро автомата  $M$ . Пусть  $w = x_1 x_2 \dots x_n$  - входная цепочка, тогда множества реакций для автоматов  $M$  и  $U$  будут соответственно представлены следующим образом:

$$q_0 / y_1, x_1 \rightarrow q_1 / y_2, x_2 \rightarrow \dots \rightarrow q_{n-1} / y_n, x_n;$$

$$q_0, x_1 / y_1 \rightarrow q_1, x_2 / y_2 \rightarrow \dots \rightarrow q_{n-1}, x_n / y_n.$$

Теорема. Для любого автомата Мили можно построить эквивалентный автомат Мура.

Доказательство. В качестве множества  $K1$  автомата Мура возьмем  $K1 = K \cdot Y$ . Для обеспечения равносильности автоматов  $M = U$  функции переходов и выходов определим следующим образом:

$$f1(p \cdot y, a) = \{qb \mid f(p, a) = q, b \in X\};$$

$$h(p \cdot y) = y.$$

Если реакция автомата  $M$  на входную цепочку вида  $w = x_1 x_2 \dots x_n$  из состояния  $q_0$  имеет вид

$$q_0, x_1 / y_1 \rightarrow q_1, x_2 / y_2 \rightarrow \dots \rightarrow q_{k-1}, x_k / y_k \quad (1),$$

то существует такое состояние  $q_0 \cdot x_1$  недетерминированного автомата  $U$ , что, начиная работу из этого состояния, автомат  $U$  выполняет следующие действия:

$$q_0 \cdot x_1 / y_1 \rightarrow q_1 \cdot x_2 / y_2 \rightarrow \dots \rightarrow q_{k-1} \cdot x_k / y_k, x_k \quad (2).$$

Аналогично можно доказать и обратную теорему о том, что из существования реакции (2) следует существование реакции (1), что подтверждает равносильность автоматов Мили и Мура

5.7.4. Задания для самостоятельной работы.

1. Построить конечный преобразователь, моделирующий работу светофора. Рассмотреть различные алгоритмы переключения светофора.
2. Построить автомат Мили, который читает текст, написанный на русском языке. Автомат должен считать все слова, начинающиеся с символа "б" и оканчивающиеся символом "т", т.е. такие, как "бит", "байт", "батут" и т.д. При построении автомата все буквы кроме "б" и "т" можно обозначить каким-либо символом, например, "|".
3. Построить автомат Мили, который читает программу, написанную на языке

## **9. УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ**

**Перечень основной и дополнительной литературы, необходимой для освоения дисциплины**

**Перечень основной литературы:**

5. В. Н. Пильщиков, В. Г. Абрамов, А. А. Вылиток, И. В. Горячая. Машины Тьюринга и алгоритмы Маркова. Решение задач. — М.: МГУ, 2006. [11]. А. Ахо., Р. Сети, Дж. Ульман. Компиляторы: принципы, технологии, инструменты. — М.: «Вильямс», 2001.
6. А. Ахо, М. Лам, Р. Сети, Дж. Ульман. Компиляторы: принципы, технологии и инструментарий. — М.: «Вильямс», 2008.

**Перечень дополнительной литературы:**

5. Олифер, В. Г. Компьютерные сети. Принципы, технологии, протоколы: учеб. пособие / В. Г. Олифер, Н. А. Олифер. - 4-е изд. - СПб.: Питер, 2011.
6. Таненбаум, Э. С. Архитектура компьютера / Э. С. Таненбаум; пер.: Ю. Гороховский, Д. Шинтяков. - 5-е изд. - СПб.: Питер, 2009.

**Перечень ресурсов информационно-телекоммуникационной сети «Интернет», необходимых для освоения дисциплины:**

9. <http://www.intuit.ru> – сайт дистанционного образования в области информационных технологий
10. <http://www.iqlib.ru> - интернет библиотека образовательных изданий, в которой собраны электронные учебники, справочные и учебные пособия;
11. <http://www.biblioclub.ru> - электронная библиотечная система «Университетская библиотека – online»: специализируется на учебных материалах для ВУЗов по научно-гуманитарной тематике.
12. <http://window.edu.ru> – образовательные ресурсы ведущих вузов