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 Информационные системы и технологии**
Направленность (профиль)	**«Технологии работы с данными и знаниями, анализ информации»**
Квалификация выпускника	Магистр

Пятигорск 2025 г.

СОДЕРЖАНИЕ

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

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

Определение 1.4. Формальное определение цепочки символов в алфавите V:

1) ε - цепочка в алфавите V;

2) если α - цепочка в алфавите V и a – символ этого алфавита, то αa – цепочка в алфавите V;

3) β - цепочка в алфавите V тогда и только тогда, когда она является таковой в силу утверждений 1) и 2).

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

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

Пример 1.1. Пусть V = {1, 0}, тогда V^* = {ε, 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:

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

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

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

P – множество правил вывода грамматики, являющееся конечным подмножеством множества (VT ∪ VN)+ × (VT ∪ VN)*; элемент (α, β) множества P называется правилом вывода и записывается в виде α→β (читается: «из цепочки α выводится цепочка β»);

S - начальный символ грамматики, S ∈VN.

Для записи правил вывода с одинаковыми левыми частями вида α → β1,α → β2,K,α → βn используется сокращенная форма записи α → β1 | β2 |K| βn .

Пример 1.2. Грамматика G1=({0, 1}, {A, S}, P1, S), где множество P1 состоит из правил вида: 1) S→ 0A1; 2) 0A→ 00A1; 3) A→ε.

Определение 1.7. Цепочка β ∈ (VT ∪ VN)* непосредственно выводима из цепочки α ∈(VT ∪VN)+ в грамматике) G = (VT ,VN , P, S (обозначается: α⇒β), если α =ξ1γξ2 и β =ξ1δξ2, где ξ1,ξ2,δ ∈(VT ∪VN)*, γ ∈(VT ∪VN)+ и правило вывода γ →δ содержится во множестве P.

Определение 1.8. Цепочка β ∈ (VT∪ VN)* выводима из цепочки α ∈$(VT \cup VN)^+$ в грамматике G = (VT ,VN , P, S) (обозначается α⇒*β), если существует последовательность цепочек γ 0,γ1 ,K,γ n (n≥0) такая, что α = γ 0 ⇒ γ1 ⇒K⇒ γ n = β .

Пример 1.3. В грамматике G1 S⇒*000111, т.к. существует вывод

S ⇒ 0A1⇒ 00A11⇒ 000A111⇒ 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) = {α ∈VT * | S ⇒ *α .

Пример 1.4. Для грамматики G1 L(G1)={0n1n | n>0}. Определение 1.10. Цепочка α ∈(VT ∪VN)*, для которой существует вывод S⇒ *α, 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 → 0S1 | 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.

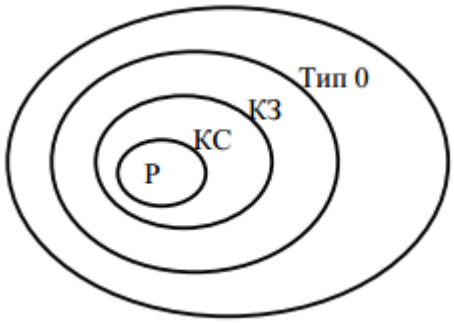
Тип 1. Грамматика G = (VT ,VN , P, S) называется контекстно-зависимой грамматикой (КЗ-грамматикой), если каждое правило вывода из множества P имеет вид α→β, где α ∈ (VT ∪ VN)+, β ∈ (VT ∪ VN)* и |α| ≤ |β|.

Тип 2. Грамматика) G = (VT ,VN , P, S называется контекстно-свободной грамматикой (КС-грамматикой), если ее правила вывода имеют вид: A → β , где A∈VN и . β ∈V *

Тип 3. Грамматика) G = (VT ,VN , P, S называется регулярной грамматикой (P-грамматикой) выровненной вправо, если ее правила вывода имеют вид A → aB | a , где a∈VT ; . A, B∈VN Грамматика) G = (VT ,VN , P, S называется регулярной грамматикой (P грамматикой) выровненной влево, если ее правила вывода имеют вид A → Ba | a , где a∈VT ; . A, B∈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)={a2bn2 −1 | n ≥1} определяется грамматикой с правилами вывода:

1) S → aaCFD; 2) AD → D;

3) F → AFB | AB; 4) Cb → bC;

5) AB → bBA; 6) CB → C;
7) Ab → bA; 8) bCD → ε.

б) Контекстно-зависимый язык L(G)={anbncn | n≥1} определяется грамматикой с правилами вывода:

1) S → aSBC | abc; 2) bC → bc;
3) CB → BC; 4) cC → cc;
5) BB → bb.

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

1) S → aQb | accb;
2) Q → cSc.

г) Регулярный язык L(G)={ω⊥ | ω∈{a, b}+, где нет двух рядом стоящих a}определяется грамматикой с правилами вывода:

1) S → A⊥ | B⊥;
2) A → a | Ba;
3) B → b | Bb | 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	$L(G)=\{c^{2n} d^n \mid n>0\}$
6	$L(G)=\{l+l-l \mid l \in\{a, b\}^+\}$
7	$L(G)=\{(10)^{n-1}(01)^{n+1} \mid n>0\}$
8	$L(G)=\{(ac)^n \mid n>0, a\in\{b, d\}, c\in\{+, -\}\}$
9	$L(G)=\{\perp(010)^n\perp \mid n>0\}$
10	$L(G)=\{a_1 a_2 \ldots a_n a_n \ldots a_2 a_1 \mid a_i\in\{0, 1\}\}$
11	$L(G)=\{a_1 a_2 \ldots a_n a_1 a_2 \ldots a_n \mid a_i\in\{c, d\}\}$
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) , (2.1)

где Q - конечное множество состояний автомата;

T - конечное множество допустимых входных символов;

F - функция переходов, отображающая множество Q × T во множество Q;

H - конечное множество начальных состояний автомата;

Z - множество заключительных состояний автомата, Z ⊆ Q.

Определение 2.2. Недетерминированным конечным автоматом (НКА) называется конечный автомат, в котором в качестве функции переходов используется

отображение Q ×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′∈ P(Q) .

Способы представления функции переходов. Командный способ. Каждую команду КА записывают в форме

F(, q, t) = p

где q, . p ∈Q, t ∈T

Табличный способ. Строки таблицы переходов соответствуют входным символам автомата t ∈ 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 ∈ 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 → aN , где A∈VN ,a∈VT и N - новый нетерминал, для каждого правила вида A → a , если в грамматике нет соответствующего ему правила A → aB , где B∈VN .

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

Шаг 3. Каждое правило A → aB преобразовать в функцию переходов F(A, a) = B где A, . B∈VN ,a∈VT

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

Шаг 5. Если в грамматике имеется правило S → ε , 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 ∅) в соответствующие позиции столбца D таблицы M ′ т.е.:

F′(D, x) ={s | s∈ F(t, x) для некоторого t ∈ D }.

Шаг 3. Для каждого нового множества R (кроме ∅), 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 ∈ 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 → aB | aA; 2) B → bB | a; 3) A → aA | 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 → bN и B → 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*	∅
b	∅	*N*	*B*	∅

1.4 Множество заключительных состояний Z = {N} .

1.5 Для начального символа грамматики ε-правила отсутствуют. Конечный автомат M - недетерминированный, граф НКА представлен на рисунке 2.1 слева.

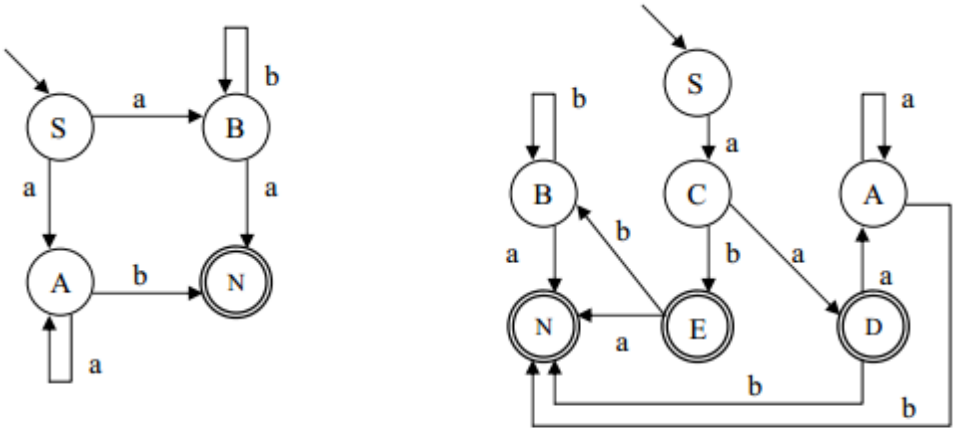


Рисунок 2.1 - Граф НКА (слева) и ДКА (справа) для Р- грамматики

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*	∅	*N*
b	∅	*B, N*	*N*	*B*	*N*	∅	*B*

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

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

2.4 Искомый ДКА определяется следующей пятеркой объектов: Q′ = {S, A, B, C, D, E, N}, } T ={a, b , функция переходов задана таблицей 2.3,

H = {S}, Z′ = {N, D, E}.
Граф полученного ДКА представлен на рисунке 2.1 справа.
Таблица 2.3 – Функция переходов для ДКА M ′

F′	*S*	*A*	*B*	*C*	*D*	*E*	*N*
a	*C*	*A*	*N*	*D*	*A*	*N*	∅
b	∅	*N*	*B*	*E*	*N*	*B*	∅

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 леволинейной грамматики. Построение леволинейной 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 λ-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) $S\to1C \mid 0D$; 2) $C\to0D \mid 0S \mid 1$; 3) $D\to1C \mid 1S \mid 0$.
2	$G=(\{S, A, B, C\}, \{a, b, c\}, P, S)$, где P: 1) $S\to aA \mid bB \mid aC$; 2) $A\to bA \mid bB \mid c$; 3) $B\to aA \mid cC \mid b$; 4) $C\to bB \mid bC \mid a$.
3	$G=(\{K, L, M, N\}, \{a, b, +, -, \perp\}, P, K)$, где P: 1) $K\to aL \mid bM$; 2) $L\to$-$N \mid$ -M; 3) $M\to+N$; 4) $N\to$a$L \mid bM \mid \perp$.
4	$G=(\{X, Y, Z, W, V\}, \{0, 1, \sim, \#, \&\}, P, X)$, где P: 1) $X\to0Y \mid 1Z \mid \varepsilon$; 2) $Y\to0Z \mid \sim W \mid \#$; 3) $Z\to1Y \mid 1W \mid 0V$; 4) $W\to0W \mid 1W \mid \#$; 5) $V\to\&Z$.
5	$G=(\{K, L, M, N, Q, P, R, S\}, \{0, 1, *, \$, /\}, V, K)$, где V: 1) $K\to1L \mid 0N$; 2) $L\to0M \mid 0P \mid /Q$; 3) $N\to1R \mid 1M \mid *S$; 4) $Q\to1P$; 5) $P\to*L \mid \$$; 6) $M\to\$$; 7) $S\to0R$; 8) $R\to/N \mid \$$.
6	$G=(\{E, A, B, C, D\}, \{0, 1, a, b, c\}, P, E)$, где P: 1) $E\to0A \mid \varepsilon$; 2) $A\to aB \mid aD$; 3) $B\to bB \mid 1C \mid c$; 4) $D\to aD \mid 0C \mid c$.
7	$G=(\{X, Y, Z, V, W\}, \{0, 1, x, y, z\}, P, X)$, где P: 1) $X\to yY \mid zZ$; 2) $Y\to1V$; 3) $Z\to0W \mid 0Y$; 4) $V\to xZ \mid xW \mid 1$; 5) $W\to1Y \mid 0$.
8	$G=(\{S, A, B, C, D\}, \{a, b, c, d, \perp\}, P, S)$, где P: 1) $S\to aA \mid bB$; 2) $A\to cC \mid \perp$; 3) $C\to cC \mid cA$; 4) $B\to dD \mid \perp$; 5) $D\to dD \mid dB$.
9	$G=(\{K, L, M, N, P\}, \{0, 1, \&, \%, a, b\}, C, K)$, где C: 1) $K\to1M \mid \varepsilon$; 2) $M\to0L \mid \&N \mid \&P$; 3) $L\to1L \mid 0L \mid \%P$; 4) $N\to aN \mid bN \mid \%P$; 5) $P\to1P \mid aP \mid 0$.
10	$G=(\{I, J, K, M, N\}, \{0, 1, \sim, !\}, P, I)$, где P: 1) $I\to0J \mid 1K \mid 0M$; 2) $J\to\sim K \mid 0M$; 3) $K\to\sim M \mid 0J \mid 0N$; 4) $M\to1K \mid !$; 5) $N\to0I \mid 1I \mid !$.
11	$G=(\{S, A, B, C, D, E\}, \{a, b, c, d, e, \$, \perp\}, P, S)$, где P: 1) $S\to aA \mid bB \mid cC$; 2) $A\to dD$; 3) $B\to\#D \mid \$E$; 4) $D\to dD \mid dB \mid \perp$; 5) $C\to cE$; 6) $E\to eE \mid eB \mid \perp$.
12	$G=(\{X, Y, Z, V\}, \{(,), y, z, v\}, P, X)$, где P: 1) $X\to(Y \mid \varepsilon$; 2) $Y\to yY \mid zY \mid zZ$; 3) $Z\to zZ \mid vZ \mid vV$; 4) $V\to vV \mid)$.

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, $n \in N \cup \{0\}$, если, находясь в одном их этих состояний и получив на вход любую цепочку символов ω: $\omega \in V_{T}^{*}$, $|\omega| \le n$, 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: КАM ′ = (Q′, T, F′, H, Z′).

Шаг 1. Поместить начальное состояние КА в список достижимых состояний $Q_д$, т.е. $Q_{д\,0} = H$.

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

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

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

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

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

Таблица 3.1 – Функция переходов конечного автомата M

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

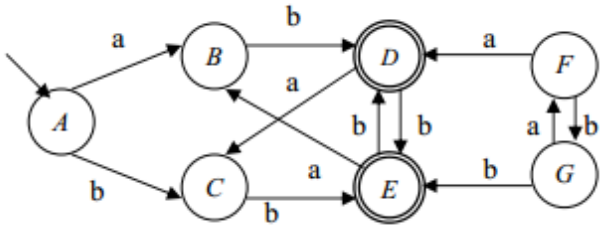


Рисунок 3.1 – Граф исходного конечного автомата M

Последовательность устранения недостижимых состояний КА имеет вид:

$Q_0 = \{A\}$;

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

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

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

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

Функция переходов автомата M ′ представлена в таблице 3.2.

Таблица 3.2 - Функция переходов автомата M ′

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

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

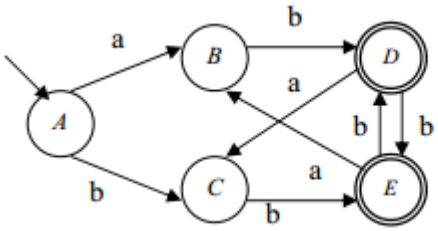


Рисунок 3.2 - Граф КА M ′ после устранения недостижимых состояний

Алгоритм 3.2. Объединение эквивалентных состояний КА

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

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

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

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

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

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

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

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

Пример 3.2. Минимизировать конечный автомат из примера 3.1.

Последовательность построения разбиений будет иметь вид:

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

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

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

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

Переобозначим оставшиеся неразбитые группы состояний:

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

Получим минимальный автомат M ″ , где Q″={A, X, Y}, Z″={Y}.

Функция переходов автомата M ″представлена в таблице 3.3.

F″	*A*	*X*	*Y*
a	*X*		*X*
b	*X*	*Y*	*Y*

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

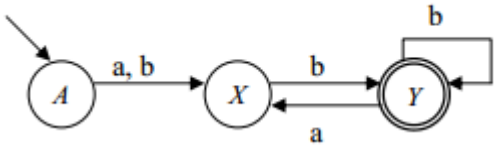


Рисунок 3.3 – Граф минимального КА M ″

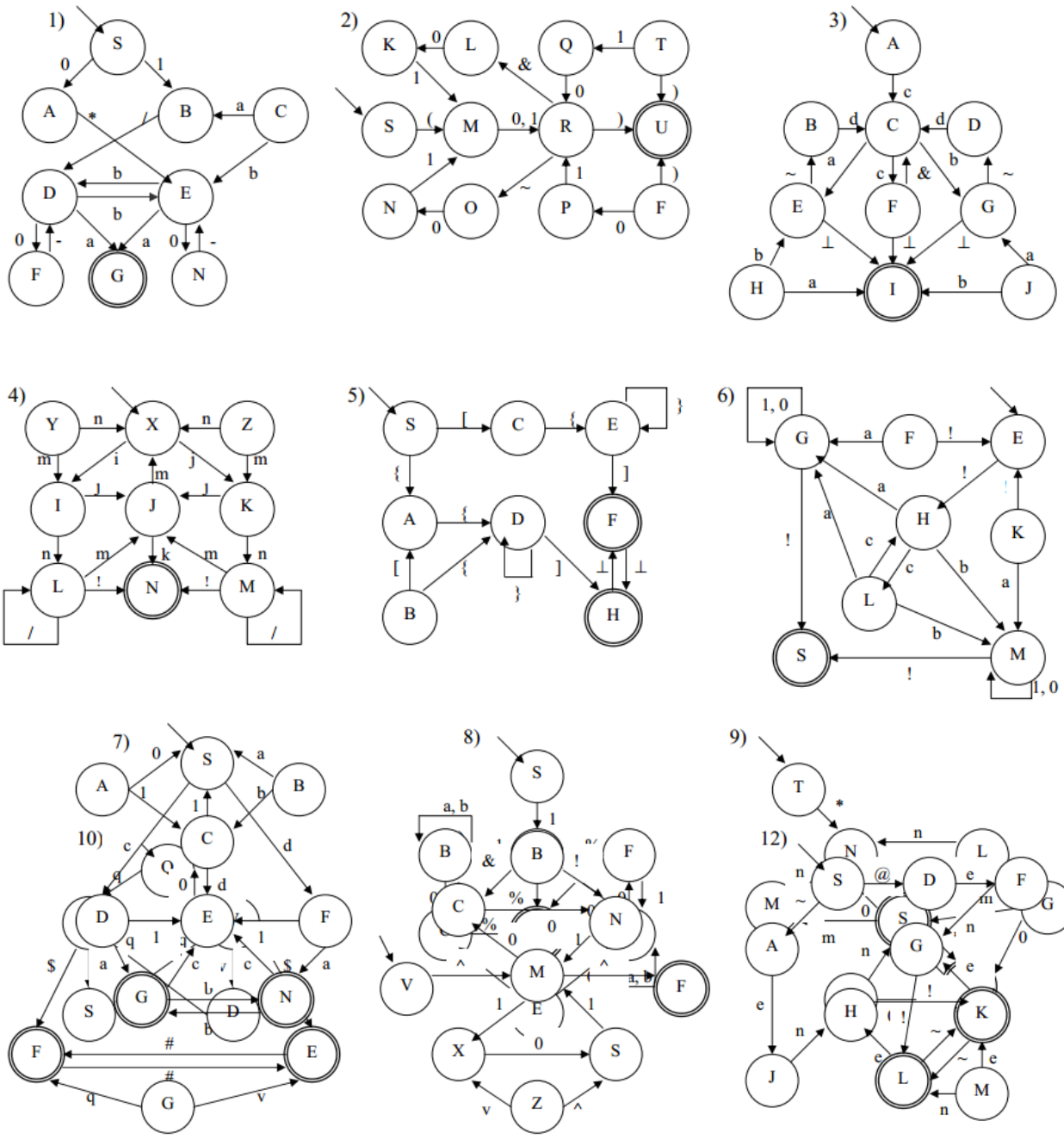
Постановка задачи к лабораторной работе № 3

Разработать программное средство, реализующее следующие функции:

1) ввод исходного конечного автомата и вывод на экран его графа;
2) устранение недостижимых состояний конечного автомата;
3) исключение эквивалентных состояний конечного автомата;
4) вывод на экран графа минимального конечного автомата. Разработать серию контрольных примеров для тестирования реализованных алгоритмов.

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

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, ε-правил и бесполезных символов.

Рассмотрим основные алгоритмы приведения КС-грамматик.

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

Алгоритм 4.1. Проверка существования языка грамматики

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

Выход: заключение о существовании или отсутствии языка грамматики.

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

Шаг 1. Положить $N_0 = \varnothing$.

Шаг 2. Вычислить N N A A P

$i = i-1 \cup \{ \mid (\rightarrow \alpha) \in$ 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 1) $S \to AB$; 2) $A \to 0A$; 3) $A \to 0$; 4) $B \to$. Построим последовательность приближений множества N:

$N_0 = \varnothing$;

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

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

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

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

Определение 4.2. Бесполезными символами грамматики называют:

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

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

б) недостижимые нетерминалы, порождающие терминальные строки, т.е. множество символов $\{X \mid X \in V_N , \neg\exists (S \Rightarrow *\alpha X\beta), \exists (X \Rightarrow *x);\alpha, \beta \in V *; x\in V_T *\}$;

в) недостижимые терминалы, т.е. множество символов $\{X \mid X \in V_T , \neg\exists (S \Rightarrow *\alpha X\beta);\alpha, \beta \in V *\}$.

Алгоритм 4.2. Устранение нетерминалов, не порождающих терминальных строк

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

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

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

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

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

Преобразуем ее в эквивалентную грамматику G′ по алгоритму 4.2:

$N_0 = Ø$;

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

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

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) $S \to ab$; 2) $B \to b$; 5) $C \to cb$.

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:

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

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 $V_N' = V_N \cap W$, $V_T' = V_T \cap W$, $V_Б = V - W$, $P' = P - P_Б$, где $P_Б \subseteq P$ - это множество правил, содержащих недостижимые символы $X \in V_Б$.

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

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 $V_Б = \{B, C, c\}$. Тогда после удаления недостижимых символов, получим грамматику $G' = (\{a, b\}, \{S\}, P, S)$ с правилом P′: . $S \to ab$

Алгоритм 4.4. Устранение ε-правил

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

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

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

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

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

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

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

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

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

Шаг 4. Если $S \in N$, то $P' = P \cup \{S' \to \varepsilon, S' \to S\}, V_N' = V_N \cup S'$, 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) $S \to AB$; 2) $A \to 0A \mid \varepsilon$; 3) $B \to 1B \mid \varepsilon$. 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) $S \to AB \mid A \mid B$; 2) 0 $A \to 0A \mid$; 3) . $B \to 1B \mid 1$

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

Алгоритм 4.5. Устранение цепных правил

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

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

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

1.1 Положить. $N_{0A} = \{A\}$

1.2 Вычислить. $N_{iA} = N_{iA-1} \cup \{C \mid (B \to C) \in P, B \in N_{iA-1}, C \in V_N\}$

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

Шаг 2. Построить множество P′ так: если $(B \to \alpha) \in P$ не является цепным правилом) $(\alpha \notin V_N$, то включить в P′ правило $A \to \alpha$ 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 → M ; 2) M → N; 3) N → 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 → N+ | n; 2) M → N+ | n; 3) N → N+ | n .

Алгоритм 4.6. Устранение левой факторизации правил

Вход: КС-грамматика G = (VT ,VN , P, S).

Выход: Эквивалентная КС-грамматика) G′ = (VT ,VN ′ , P′, S′ без одинаковых префиксов в правых частях правил, определяющих нетерминалы.

Шаг 1. Записать все правила для нетерминала X , имеющие одинаковые префиксы α ∈ V *, в виде одного правила с альтернативами: X →αβ1 |αβ2 |K|αβn; β1, β2,K, βn ∈ V *.

Шаг 2. Вынести за скобки влево префикс α в каждой строкеальтернативе:. X →α(β1 | β2 |K| βn)

Шаг 3. Обозначить новым нетерминалом Y выражение, оставшееся в скобках: X →αY, Y → β1 | β2 |K| βn.

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 → kSl; 2) S → kSm; 3) S → n Преобразуем ее в эквивалентную грамматику G′ по алгоритму 4.6:

Шаг 1. S → kSl | kSm | n .

Шаг 2. S → kS(l | m) | n .

Шаг 3,4. Пополнив множество нетерминалов новым нетерминалом C и

заменив правила, подвергшиеся факторизации, получим грамматику G′ = ({k,l,m,n},{S, C}, P′, S) с правилами : P′ ; 1) S → kSC; 2) S → n; 3) C → l 4) C → m.

Алгоритм 4.7. Устранение прямой левой рекурсии

Вход: КС-грамматика G = (VT ,VN , P, S).

Выход: Эквивалентная КС-грамматика) G′ = (VT ,VN ′ , P′, S′ без прямой левой рекурсии, т.е. без правил вида . A → Aα, A∈ VN ,α ∈ V *

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

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

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

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

$$Y \to \alpha_1 Y \mid \alpha_2 Y \mid \ldots \mid \alpha_m Y$$

$$Y \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_m.$$

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

$$X \to \beta_1 Y \mid \beta_2 Y \mid \ldots \mid \beta_n Y$$

$$X \to \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$$

$$Y \to \alpha_1 Y \mid \alpha_2 Y \mid \ldots \mid \alpha_m Y$$

$$Y \to \alpha_1 \mid \alpha_2 \mid \ldots \mid \alpha_m.$$

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

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

Пример 4.7. Дана грамматика) G = ({a, b, c, d, z}, {S, A, B, C}, P, S с правилами : P 1) S → Aa; 2) A → Bb; 3) B → Cc | d; 4) C → 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 → Aa; 2) A → Bb; 3) B → Cc | d; 4) C → dbzZ | dbz; 5) Z → 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) ε-правил;
г) цепных правил;
д) левой факторизации правил;
е) прямой левой рекурсии.

Варианты индивидуальных заданий представлены в таблице 4.1.

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

Вариант	Контекстно-свободная грамматика
1	$G=(\{S, A, B, D, E\}, \{a, b, c, e\}, P, S)$, где P: 1) $S\to AB \mid \varepsilon$; 2) $A\to Aa \mid S \mid a$; 3) $B\to bD \mid bS \mid b$; 4) $D\to ccD$; 5) $E\to eE \mid e$.
2	$G=(\{E, T, F, G, H\}, \{+, -, *, /, n, m, h\}, P, E)$, где P: 1) $E\to T \mid E+T \mid E-T \mid \varepsilon$; 2) $T\to F \mid F*T \mid F/T \mid \varepsilon$; 3) $F\to G \mid Fn \mid n$; 4) $G\to Gm$; 5) $H\to Hh \mid h$.
3	$G=(\{S, R, T, X, Y\}, \{a, b, p, g, y\}, P, S)$, где P: 1) $S \to R \mid T$; 2) $R \to pX \mid paR \mid paT \mid \varepsilon$ 3) $T \to Tg \mid g$; 4) $X \to aXb$; 5) $Y \to aYa \mid y$.
4	$G=(\{Q, A, B, C, D\}, \{a, b, c, d\}, P, Q)$, где P: 1) $Q \to acA \mid acB \mid \varepsilon$; 2) $B \to A \mid Cb \mid \varepsilon$; 3) $A \to Aa \mid Ab \mid a$; 4) $C \to dCc$ 5)$D \to dc$
5	$G=(\{R, T, F, G, K\}, \{m, i, j, k, \wedge, \sim, \perp\}, P, R)$, где P: 1) $R\to R\sim T\perp \mid R\wedge T\perp \mid \varepsilon$; 2) $T\to F \mid Fi \mid Fj \mid Gk \mid \varepsilon$; 3) $G\to GkG$; 4) $K\to Ki \mid Km \mid m$.
6	$G=(\{S, X, Y, Z, K\}, \{x, y, z, k, \#, \$\}, P, S)$, где P: 1) $S\to X \mid Y \mid Z$; 2) $X\to x\#X \mid x\#Y \mid \varepsilon$; 3) $Y\to Yy\$ \mid Yz\$ \mid \$ \mid \varepsilon$; 4) $Z\to Zz\$$; 5) $K\to Kk\$ \mid k\$$.
7	$G=(\{S, L, M, P, N\}, \{n, m, l, p, @, \perp\}, V, S)$, где V: 1) $S\to @nL \mid @mM \mid P$; 2) $L\to M \mid Ll\perp \mid Lm\perp \mid \varepsilon$; 3) $M\to L \mid Mm \mid mm$; 4) $N\to pN@ \mid @$; 5) $P\to nmP$.
8	$G=(\{X, Y, Z, K, L\}, \{a, b, l, =, <, >, \wedge, \vee, \neg\}, V, X)$, где V: 1) $X\to Y \mid Y=Y \mid Y<Y \mid Y>Y \mid K$; 2) $Y\to Y\wedge Z \mid Y\vee Z \mid \varepsilon$; 3) $Z\to \neg a \mid \neg b \mid \varepsilon$; 4) $K\to \neg K$; 5) $L\to l \mid a \mid b$.
9	$G=(\{Q, A, B, C, D\}, \{0, 1, -\}, P, Q)$, где P: 1) $Q\to 01A \mid 01B \mid A$; 2) $A\to 0B1 \mid B \mid 1 \mid \varepsilon$; 3) $B\to BA0 \mid B1 \mid C \mid \varepsilon$;
	4) $C\to 0C11$; 5) $D\to -D1 \mid -0 \mid -1$.
10	$G=(\{R, T, U, W, V\}, \{0, 1, +, -, *, /\}, P, R)$, где P: 1) $R\to T1T \mid T1U \mid W \mid \varepsilon$; 2) $T\to U \mid T01 \mid T10 \mid \varepsilon$; 3) $U\to +U \mid +0 \mid +1$ 4) $W\to W-W \mid W+W$; 5) $V\to *0 \mid /1$.

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

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:

$$M = (Q, T, N, F, q0, N0, Z), \quad (5.1)$$

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.

$F : (Q\times (T \cup \{\varepsilon\})\times N) \to P(Q \times N^*)$; q0 – начальное состояние автомата, $q0 \in Q$;

N0– начальный символ магазина, $N0 \in T$;

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

Определение 5.2. Конфигурацией МП-автомата называется тройка вида:

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

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

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

α - содержимое магазина, . α ∈ N*.

Общая схема МП-автомата представлена на рисунке 5.1.

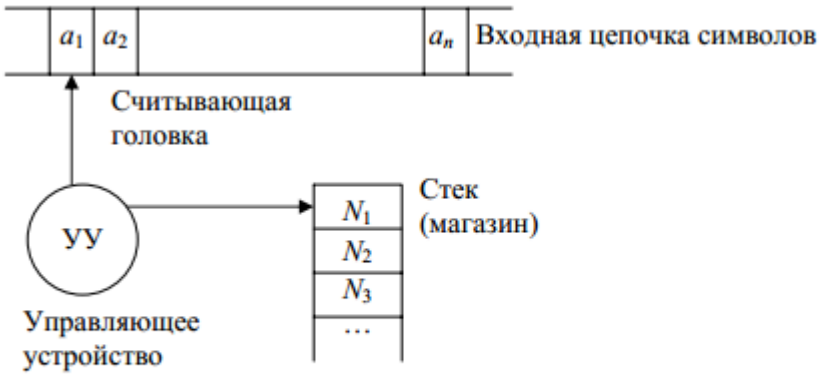


Рисунок 5.1 – Схема МП-автомата

Алгоритм 5.1. Функционирование МП-автомата

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

Шаг работы МП-автомата будем представлять в виде отношения непосредственного следования конфигураций (обозначается «|=») и отношения достижимости конфигураций (обозначается «|=*»). Если одним из значений магазинной функции) F(q ∈ Q, t ∈ (T ∪ {ε}), S ∈ N является) (q′∈ Q, γ ∈ N* , то записывается) (q, tω, Sα) |= (q′,ω, γα . При этом возможны следующие варианты.

1) Случай t ∈ T. Автомат находится в текущем состоянии q, читает входной символ t, имеет в вершине стека символ S. Он переходит в очередное состояние q′, сдвигает входную головку на ячейку вправо и заменяет верхний символ S строкой γ магазинных символов. Вариант γ = ε означает, что S удаляется из стека.

2) Случай t = ε . Отличается от первого случая тем, что входной символ tпросто не принимается во внимание, и входная головка не сдвигается. Такой шаг работы МП-автомата называется ε -шагом, который может выполняться даже после завершения чтения входной строки. Заключительной конфигурацией МП-автомата является конфигурация (q, ε , α), где q ∈ Z.

Определение 5.3. МП-автомат допускает входную стоку ω , если существует путь по конфигурациям (q_0,ω, N_0) |= *(q, ε,α)для некоторых q ∈ Z и * α ∈ N .

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

L(M) = {ω |ω ∈ T^* и L(M) = {ω |ω ∈ T и) (q_0,ω, N_0) |= * (q, ε,α для некоторых q ∈ Z и * α ∈ N }.

Определение 5.5. МП-автомат с магазинной функцией F : (Q× (T ∪{ε})× N*) → P(Q× 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 сентенции, первоначально это начальный символ грамматики. Очередная сентенция получается заменой верхнего нетерминала стека.

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

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

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

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

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

Пример 5.1. Дана КС-грамматика: $G(\{+, (,), a\}, \{S, A\}, \{S \rightarrow S+A \mid A, A \rightarrow (S) \mid a\}, \{S\})$. Последовательность построения МП-автомата будет иметь вид.

1) $Q = \{q\}$, $q_0 = q$, $T = \{+, (,), a\}$, $N = \{+, (,), a, S, A\}$, $N_0 = S$, $Z = \emptyset$.
2) $F(q, \varepsilon, S) = (q, S+A)$, $F(q, \varepsilon, S) = (q, A)$, $F(q, \varepsilon, A) = (q, (S))$; $F(q, \varepsilon, A) = (q, a)$.
3) $F(q, t, t) = (q, \varepsilon)$ для каждого $t \in \{+, (,), a\}$.

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

Таблица 5.1 – Распознавание МП-автоматом строки (a)

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

Алгоритм 5.3. Построение расширенного МП-автомата по КС-грамматике. Построим МП-автомат, выполняющий правосторонний разбор. Данный автомат имеет единственное текущее состояние и одно заключительное состояние, в котором стек пуст. Стек содержит левую часть текущей сентенции. Первоначально в стек помещается специальный магазинный символ, маркер пустого стека #. На каждом шаге автомат по

правилу грамматики замещает нетерминалом строку верхних символов стека или дописывает в вершину входной символ.

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

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

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

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

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

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

Пример 5.2. Для грамматики из примера 5.1 построить расширенный МП-автомат. Последовательность построения МП-автомата будет иметь вид. 1) $Q = \{q, r\}$, $q_0 = q$, $T = \{+, (,), a\}$, $N = \{+, (,), a, S, A\}$, $N_0 = \#$, $Z = r$.

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

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

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

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

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

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

Постановка задачи к лабораторной работе № 5

Разработать программное средство, реализующее следующие функции:

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

б) построение МП-автомата по КС-грамматике;

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

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

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

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

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, α) и FOLLOW(k, α), 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, α) – 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:

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

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

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

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

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

$\forall\ i \ne j,\ 0<i \le n,\ 0<j \le n$.

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

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

Следовательно, достаточно рассмотреть алгоритмы построения множеств FIRST(1, A) и FOLLOW(1, A) для каждого нетерминального символа A.

Алгоритм 6.1. Построение множества FIRST(1, A)

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

Шаг 1. Первоначально внести во множество первых символов для каждого нетерминального символа A все символы, стоящие в начале правых частей правил для этого нетерминала, т.е. $\forall\ A \in V_N\ FIRST(1, A) = \{X \mid A \to X\alpha \in P, X \in (V_T \cup V_N), \alpha \in (V_T \cup V_N)^*\}$.

Шаг 2. Для всех $A \in V_N$ положить: $FIRST_{i+1}(1, A) = FIRST_i(1, A) \cup FIRST_i(1, B)$, $\forall\ B \in (FIRST(1, A) \cap V_N)$.

Шаг 3. Если существует $A \in V_N$, такой что $FIRST_{i+1}(1, A) \ne FIRST_i(1, A)$, 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. $\forall\ A \in V_N\ FIRST(1, A) = FIRST_i(1, A) \setminus N$.

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

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

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

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

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

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

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

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

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

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

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

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

- если на верхушке стека находится терминальный символ a, совпадающий с очередным символом входной строки, то выполняем операцию «выброс», т.е. удаляем из стека и входного буфера данный терминальный символ;

- если содержимое стека и входного буфера пусто, то исходная строка прочитана полностью, и разбор завершен удачно;

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

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

$N_0 = \{R\}$;

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

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

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

Таблица 6.1 – Построение множеств FIRST(1, A)

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

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

Таблица 6.2 – Построение множеств FOLLOW(1, A)

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

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

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

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

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*→*TR*, т.к. (∈ *FIRST*(1, *TR*)
TR	(*a*+(*b*-*a*))	свертка *T*→(*S*), т.к. (∈ *FIRST*(1, (*S*))
(*S*)*R*	(*a*+(*b*-a))	выброс
S)*R*	*a*+(*b*-*a*))	свертка *S*→*TR*, т.к. *a*∈ *FIRST*(1, *TR*)
TR)*R*	*a*+(*b*-*a*))	свертка *T*→*a*, т.к. *a*∈ *FIRST*(1, *a*)
aR)*R*	*a*+(*b*-*a*))	выброс
R)*R*	+(*b*-*a*))	свертка *R*→+*TR*, т.к. +∈ *FIRST*(1, *TR*)
+*TR*)*R*	+(*b*-*a*))	выброс
TR)*R*	(*b*-*a*))	свертка *T*→(*S*), т.к. (∈ *FIRST*(1, (*S*))
(*S*)*R*)*R*	(*b*-*a*))	выброс
S)*R*)*R*	*b*-*a*))	свертка *S*→*TR*, т.к. *b*∈ *FIRST*(1, *TR*)
TR)*R*)*R*	*b*-*a*))	свертка *T*→*b*, т.к. *b*∈ *FIRST*(1, *b*)
bR)*R*)*R*	*b*-*a*))	выброс
R)*R*)*R*	-*a*))	свертка *R*→-*TR*, т.к. -∈ *FIRST*(1, -*TR*)
-*TR*)*R*)*R*	-*a*))	выброс
TR)*R*)*R*	*a*))	свертка *T*→*a*, т.к. *a*∈ *FIRST*(1, *a*)
aR)*R*)*R*	*a*))	выброс
R)*R*)*R*	))	свертка *R*→ε, т.к.) ∈ *FOLLOW*(1, *R*)
)*R*)*R*	))	выброс
R)*R*	)	свертка *R*→ε, т.к.) ∈ *FOLLOW*(1, *R*)
)*R*	)	выброс
R	ε	свертка *R*→ε, т.к. ε∈ *FOLLOW*(1, *R*)
ε	ε	строка принята полностью

Шаг 6. Получили следующую цепочку вывода:

S⇒TR⇒(S)R⇒(TR)R⇒(aR)R⇒(a+TR)R⇒(a+(S)R)R⇒(a+(TR)R)R⇒
⇒(a+(bR)R)R⇒(a+(b-TR)R)R⇒(a+(b-aR)R)R⇒(a+(b-a)R)R⇒(a+(b-a))R⇒(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.

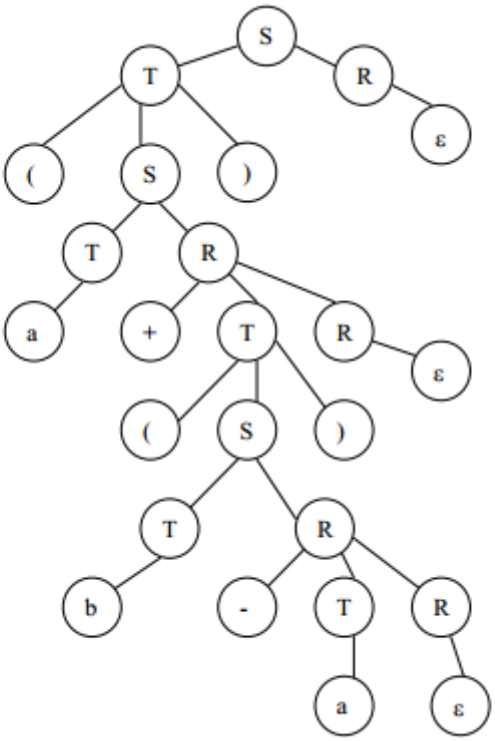


Рисунок 6.1 – Дерево вывода для цепочки (a+(b-a)) в грамматике G

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

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

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

2) Различные правила в грамматике имеют разные правые части.

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

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

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

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

Алгоритм 7.1. Поиск основы сентенции грамматики

Если грамматика является грамматикой простого предшествования, то для поиска основы каждой ее 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 $x_i\ x_{i+1} \ldots x_{j-1}\ x_j$. 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:

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

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

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

Матрицу предшествования дополняют символами $\perp_н$ и $\perp_к$ (начало и конец цепочки). Для них определены следующие отношения предшествования:

- $\perp_н <\cdot\ X$, $\forall\ X \in V$, если $X \in L(S)$;

- $\perp_к \cdot > X$, $\forall\ X \in V$, 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, содержание 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.

$\forall\ A \in V_N$: $L_0(A) = \{X \mid A \rightarrow Xy, X \in V, y \in V^*\}$,

$R_0(A) = \{X \mid A \rightarrow yX, X \in V, y \in V^*\}$.

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 $\perp_н$, 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 $\perp_н$S, а во входном буфере только символ $\perp_к$, 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→(R | a; 2) R→Sa). Построить распознаватель для строки (((aa)a)a)$\perp_к$.

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→(R символ (стоит слева от нетерминального символа R. Во множестве L(R)

входят символы S, (, a. Ставим знак <· в клетках матрицы, соответствующих этим символам, в строке для символа (.

В правиле грамматики R→Sa) символ a стоит справа от нетерминального символа S. Во множество R(S) входят символы R, a,). Ставим знак · > в клетках матрицы, соответствующих этим символам, в столбце для символа a. В строке символа $\perp_н$ ставим знак <· в клетках символов, входящих во множество L(S), т.е. символов (, a. В столбце символа $\perp_к$ ставим знак · > в клетках, входящих во множество R(S), т.е. символов R, a,). В клетках, соответствующих строке символа S и столбцу символа a, ставим знак =· , т.к. существует правило R→Sa), в котором эти символы стоят рядом. По тем же соображениям ставим знак =· 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*	(	)	$\perp_к$
S			=.			
R			.>			.>
a			.>		=.	.>
(	<.	=.	<.	<.		
)			.>			.>
$\perp_н$			<.	<.		

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

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

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

Шаг 4. Получили следующую цепочку вывода:
S⇒(R⇒(Sa)⇒((Ra)⇒((Sa)a)⇒(((Ra)a)⇒(((Sa)a)a)⇒(((aa)a)a).
Восходящее дерево вывода цепочки представлено на рисунке 7.2.

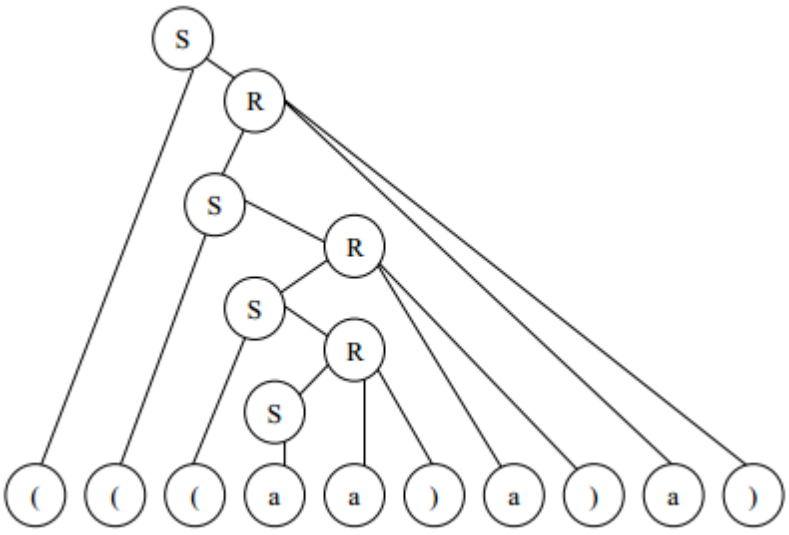


Рисунок 7.2 – Дерево вывода для цепочки (((aa)a)a) в грамматике G

Постановка задачи к лабораторной работе № 7

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

1) ввод произвольной грамматики;

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

Составить набор контрольных примеров для случаев:

а) введенная грамматика не является грамматикой простого предшествования;

б) исходная грамматика является грамматикой простого предшествования , но анализируемая строка не принадлежит языку грамматики;

в) заданная грамматика является грамматикой простого предшествования и входная строка принадлежит языку грамматики.

Разбор цепочек представить в виде таблицы, строки вывода и дерева вывода. Вариантами индивидуального задания к лабораторной работе № 7 являются выходные данные лабораторной работы № 4.

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. Макровызов, 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. 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.
2. А. Ахо, М. Лам, Р.Сети, Дж. Ульман. Компиляторы: принципы, технологии и инструментарий. — М.: «Вильямс», 2008.

Перечень дополнительной литературы:

1. Олифер, В. Г. Компьютерныесети. Принципы, технологии, протоколы: учеб. пособие / В. Г. Олифер, Н. А. Олифер. - 4-е изд. - СПб.: Питер, 2011.
2. Таненбаум, Э. С. Архитектуракомпьютера / 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

ПРИЛОЖЕНИЕ 1

Пример оформления отчета лабораторной работы

Содержание

1 Тема и цель лабораторной работы

Лабораторная работа № 2.

Тема: «Построение конечного автомата по регулярной амматике»

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

- сформировать умения и навыки построения конечного томата по регулярной грамматике и преобразования недетерминиро-нного конечного автомата к детерминированному конечному автома-

2 Постановка задачи

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

1) $S \to aB \mid aA$;
2) $A \to aA \mid bC \mid b$;
3) $B \to bB \mid aC \mid a$.

Разработать программное средство, реализующее следующие нкции:

1) ввод произвольной формальной грамматики с клавиатуры и оверка ее на принадлежность к классу регулярных грамматик;

2) построение по заданной регулярной грамматике конечного томата;

3) преобразование недетерминированного конечного автомата к герминированному конечному автомату;

4) вывод графа результирующего конечного автомата на экран.

3 Формальная модель задачи

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

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

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 *Grammar*		
N	*charset*	множество нетерминалов грамматики
T	*charset*	множество терминалов грамматики
P	*rulemap*	множество правил грамматики
S	*char*	начальный символ грамматики
Класс *FAutomat*		
**G*	*Grammar*	связанная с данным автоматом грамматика
Q	*charset*	множество состояний автомата
T	*charset*	множество входных символов автомата
F	*ftable*	таблица правил автомата
H	*char*	начальное состояние автомата
Z	*charset*	множество конечных состояний автомата
MustPaint	*int*	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.

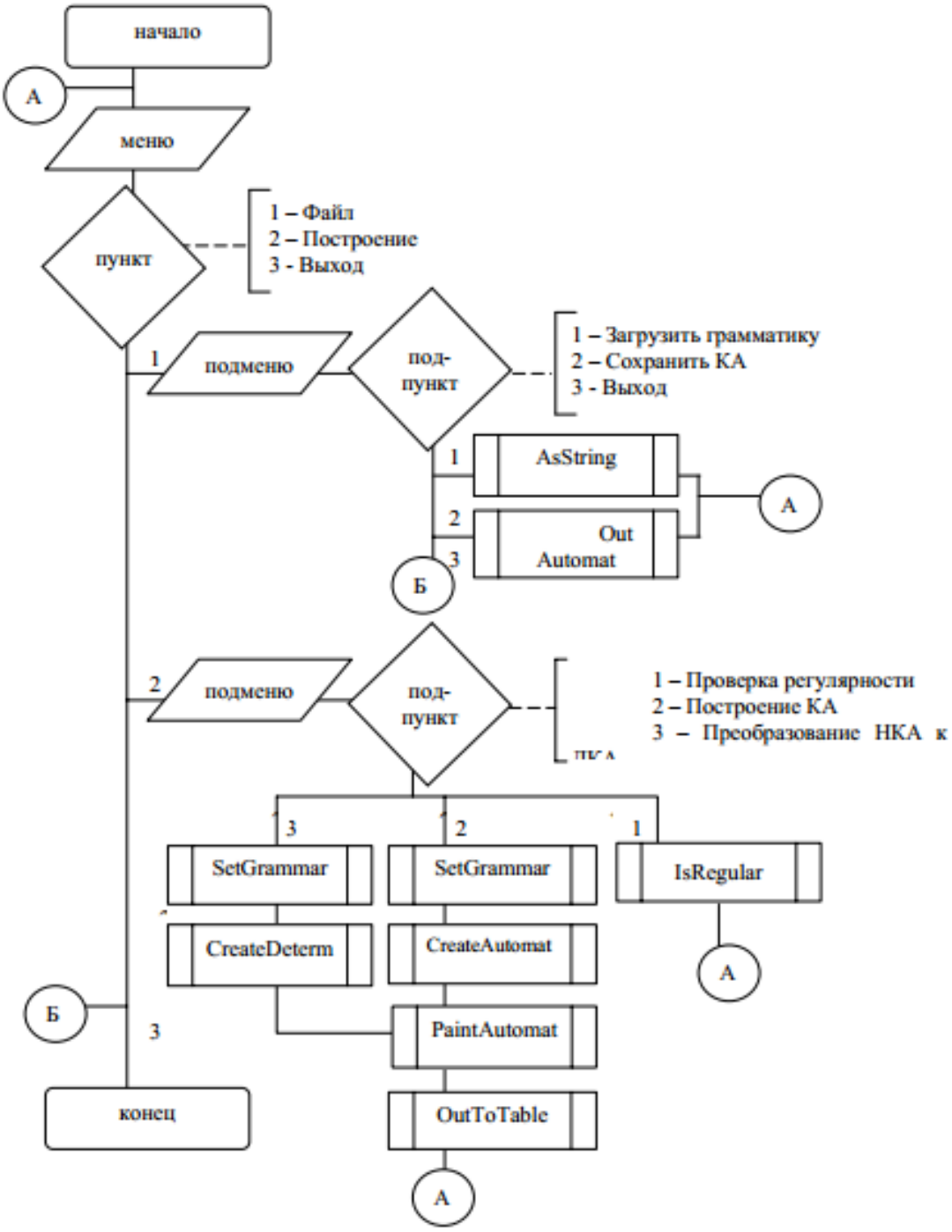


Рисунок 6.1 – Укрупненная схема алгоритма программного средства

Приложение Б

(обязательное)

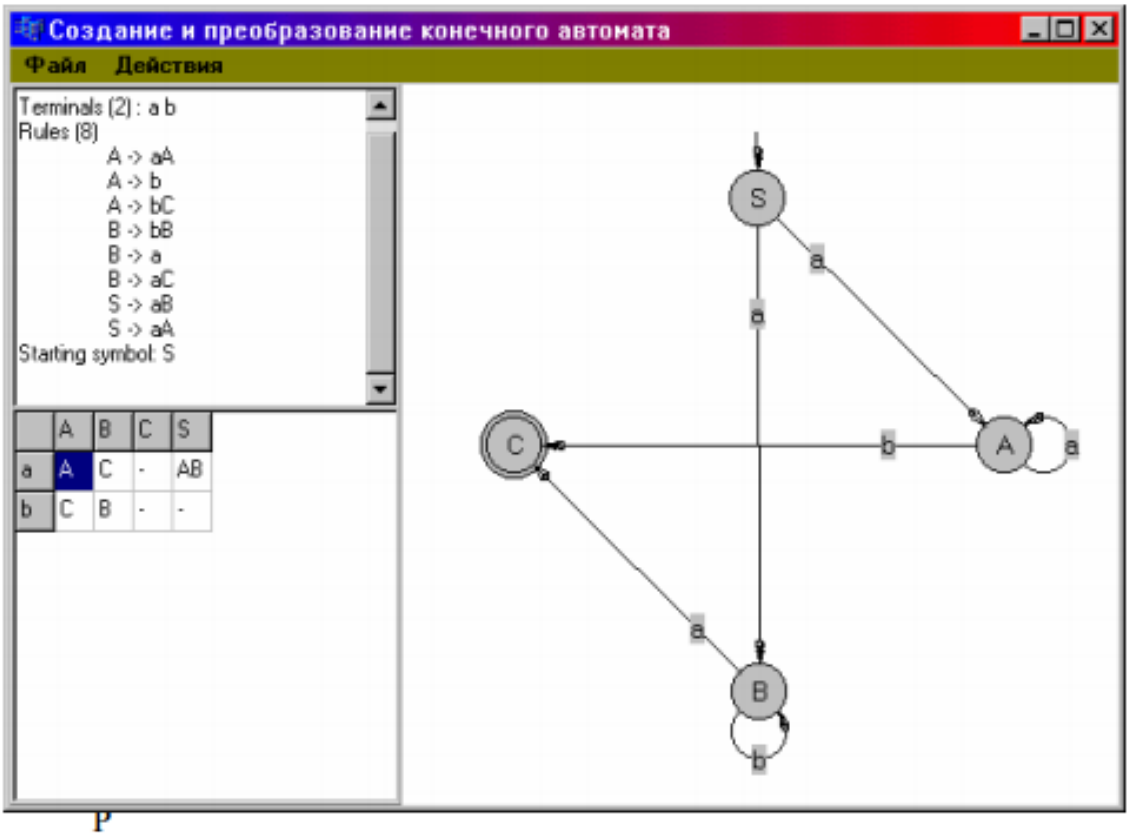
Пример оформления приложений отчета лабораторной работы

Приложение А

(обязательное)

Контрольный пример

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



Р

Рисунок А.1 – Исходная КС-грамматика и НКА

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

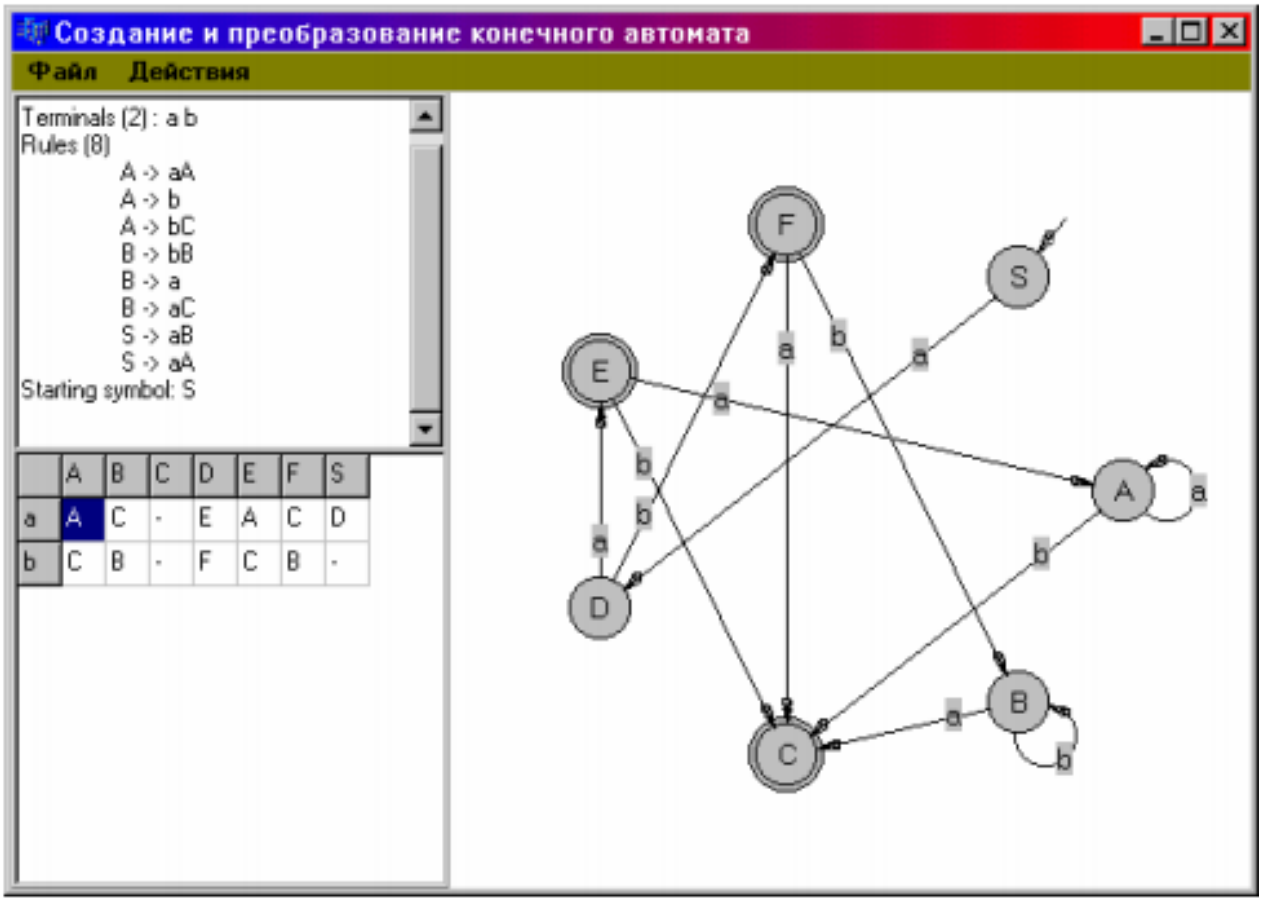


Рисунок А.2 – Результирующий ДКА

Приложение Б

(обязательное)

Текст программы

Unit2.h

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

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

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

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

// класс грамматики
class Grammar{
        public:
                charset N; // мн-во нетерминалов
                charset T; // множество терминалов
                rulemap P; // множество правил
                char S; // начальный символ
      int IsRegular(); // проверка грамматики на регулярность
      void InGrammar(char *fname); // ввод грамматики из файла
      string AsString(); // вывод грамматики в строку
      void OutGrammar(char *fname); // вывод грамматики в файл
};

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

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

Unit2.cpp

```
#include "Unit2.h"

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

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

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

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

   res += "Starting symbol: ";
   res += S;
   res += "\n";
   return res;
}

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

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

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

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	**09.04.02 Информационные системы и технологии**
Направленность (профиль)	**«Технологии работы с данными и знаниями, анализ информации»**
Квалификация выпускника	Магистр

Пятигорск, 2025

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 данныхметодов 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 заочной формы обучения:

Коды реализуемых компетенций, индикатора(ов)	Вид деятельности студентов	Средства и технологии оценки	Обьем часов, в том числе		
			СРС	Контактная работа с преподавателем	Всего
3 семестр					
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
Итого за 3 семестр			**110,7**	**12,3**	**123**

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 поп 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 леволинейной грамматики.
10. Построение леволинейной 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 НКА.
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 λ-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 однопроходные 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. Настраивающий 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. Макровызов, 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 леволинейной грамматики. Построение леволинейной 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 λ-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. Макровызов, 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. Этого же мнения 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 собойавтоматную модель, в основе которой лежит анализ процесса выполнения алгоритма каксовокупности

набора инструкций.Третий тип алгоритмических моделей - это преобразования слов в произвольныхалфавитах, в которых элементарными операциями являются подстановки, т.е. замены частислова (подслова) другим словом. Преимущество этого типа состоит в его максимальнойабстрактности и возможности применить понятие алгоритма к объектам произвольнойприроды. 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

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 ai никакого выходного слова bj (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 b1, b2, ..., bn:

а) алгоритм вычисления по формуле:

$$C=(...((A - b_1) - b_2) - ... - b_n)$$

б) алгоритм вычисления по формуле:

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

2. Разработать алгоритм вычисления по формуле:

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

3. Разработать алгоритм вычисления по формуле:

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

4. Разработать алгоритм поиска максимального (минимального) элемента из последовательности, заданной в виде одномерного массива A = {a1, a2, ... , ak}.
5. Разработать алгоритм определения количества одинаковых чисел в последовательности, заданной в виде одномерного массива A={a1, a2, . . . , ak}.
6. Разработать алгоритм подсчета количества одинаковых элементов в матрице ***B*** размерностью n × 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 = {a1, a2, . . ., ak} и B= {b1, b2}.

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 черезззначения определяемой функции для меньших значений аргументов.Применение рекурсивных функций в теории алгоритмов основано на идее нумерациислов в произвольном алфавите последовательными натуральными числами. Наиболее простотакую нумерацию можно осуществить, располагая слова в порядке возрастания их длин, аслова, имеющие одинаковую длину, - в произвольном порядке.После нумерации входных и выходных слов в произвольном алфавитном операторе этотоператор превращается в функцию y = f(x), в которой аргумент x и функция y принимаютнеотрицательные целочисленные значения. Функция f(x) может быть определена не для всехзначений 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(x1, x2, ..., xn) = 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 n m 1, x2, . . ., xm, . . . , xn) = xm - 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(x1, x2, ..., xm), f1(x1, x2, ..., xn), f2(x1, x2, ..., xn), . . ., fm(x1, x2, ..., xn) можно получить новую функцию: Sm+1(f, f1, f2, ..., fm) = g(x1, x2, ..., xn) = f(f1(x1, x2, ..., xn), f2(x1, x2, ..., xn), ..., fm(x1, ..., xn)). (1)

В операции суперпозиции Sm+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(x2, x1, x3, . . ., xn) = f(I22(x1, x2), I12(x1, x2), x3, . . ., xn);

f(x1, x1, x3, . . ., xn) = f(I12(x1, x2), I12(x1, x2), x3, . . ., xn).

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 Rn 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(x1, x2, ..., xn, y) получается оператором примитивной рекурсии из функций g(x1, x2, ..., xn) и функции h(x1, x2, ..., xn, y, z), если:

f(x1, x2, ..., xn, 0) = g(x1, x2, ..., xn); (2)

f(x1, x2, ..., xn, y+1) = h(x1, x2, ..., xn, y, f(x1, x2, ..., xn, 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 x1, x2, ..., xn на момент применения схемы (2) зафиксированы и играют рольпараметров.

Оператор минимизации

Отношение P(x1, x2, ..., xn) называется примитивно-рекурсивным, если примитивно рекурсивна его характеристическая функция:

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

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

Функция f(x1, x2, ..., xn) получается посредством оператора минимизации из предиката P(x1, x2, ..., xn, z), если в любой точке значением функции f(x1, x2, ..., xn) является минимальное значение z, обращающее предикат P(x1, x2, ..., xn, z) в значение «истина»:

f(x1, x2, ..., xn) = μz (P(x1, x2, ..., xn, z)),

где μz – оператор минимизации.

2.2.4. Ограниченный оператор минимизации

Функция f(x1, x2, ..., xn) получается ограниченным оператором минимизации из предиката P(x1, x2, ..., xn, z) и функции U(x1, x2, ..., xn), если в любой точке значение этой функции определено следующим образом:

1) для любого z < U(x1, x2, ..., xn) такого, что P(x1, x2, ..., xn, z) = "истина", значение функции f(x1, x2, ..., xn) = μz(P(x1, ..., xn, z)),

2) не для любого z < U(x1, x2, ..., xn) такого, что P(x1, x2, ..., xn, z) = "истина", значение функции f(x1, x2, ..., xn) = U(x1, x2, ..., xn).

Данное определение записывается следующим образом:f(x1, x2, ..., xn) = μ z < U(x) (P(x1, x2, ..., xn, z)).Ограничение z в ограниченном операторе минимизации дает гарантию окончаниявычислений, поскольку оно оценивает сверху число вычислений предиката P. Возможностьоценить сверху количество вычислений является существенной особенностью примитивнорекурсивных 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 примитивнорекурсивной:

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 некоторыхусловий:

1) если исходные данные имеют рекурсивную структуру, то процедуры анализа такихструктур будут более эффективны, если они сами рекурсивны;

2) если алгоритм обработки некоторого набора данных построить, разбивая данные начасти 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(I3(x, y, f(x, y))).

Пример 2. Доказать примитивную рекурсивность функции f(x, y) = x · y.

Решение. Рассмотрим способ рекурсивного определения данной функции:

f(x, 0) = 0, f(x, y+1) = x · (y + 1) = x · y + x = f(x, y) +x, из которого следует, что Z(x) = x · 0. Обозначим x · 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)), 1 3 I 3 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 · 2x =2 ·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) = μz(2·x + z = y+1), откуда предикат P(x, y, z) = 2·x + z = y + 1.

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

Sg(Sg((2 · x + z) ÷(y + 1)) + Sg((y +1) ÷ (2 · x + z)).

Найдем значение функции в точке (1, 5):

При z = 0: P(1, 5, 0) = 2 · 1 + 0 = 5 + 1 - "ложь",

z = 1: P(1, 5, 1) = 2 · 1 + 1 = 5 + 1 - "ложь",

z = 2: P(1, 5, 1) = 2 · 1 + 2 = 5 + 1 - "ложь",

z = 3: P(1, 5, 1) = 2 · 1 + 3 = 5 + 1 - "ложь",

z = 4: P(1, 5, 4) = 2 · 1 + 4 = 5 + 1 - "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

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

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

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

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 x1 > x2 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 отличатьсятолько 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 5x5. 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 NxN 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 управляющееустройство. Управляющее устройство может находиться в одном из состояний, образующихконечное множество Q = {q0, q1, ..., qn}. Множество Q называют внутренним алфавитоммашины Тьюринга

Принципиальное отличие машины Тьюринга от вычислительных машин состоит в том,что ее запоминающее устройство представляет собой бесконечную ленту, из-за которойневозможна ее физическая реализация. Лента разделена на ячейки, в каждой из которыхможет быть записан один из символов конечного алфавита A = {a0, a1, . . . , am}, которыйназывают входным алфавитом машины Тьюринга. Во время функционирования машиныТьюринга может быть заполнено конечное число ячеек. Считывающая головка в каждыймомент времени обозревает ячейку ленты, в зависимости от символа в этой ячейке исостояния управляющего устройства записывает в ячейку новый символ или оставляет егобез изменения, сдвигается на ячейку влево или вправо или остается на месте. При этомуправляющее устройство переходит в новое состояние или остается в старом. Средисостояний управляющего устройства выделены начальное состояние q0 и заключительноесостояние qz.

Таким образом, за один такт работы машина Тьюринга может считать символ, записатьвместо него новый или оставить его без изменения и сдвинуть головку на одну ячейку влевоили 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 |. Длинапустой цепочки равна нулю.

Конкатенацией цепочек X и Y называют цепочку, полученную приписыванием символовцепочки Y справа к цепочке X.

Машиной Тьюринга называется семерка видаT = (Q, A, δ, p0, pz, a0, a1), где
Q – конечное множество состояний, в которых может находиться управляющееустройство;

A – входной алфавит;

δ – функция переходов, δ = Q ·A → Q ·A ·S, где

S = {R, L, E} - направления сдвига;

p0 – начальное состояние; p0 ∈Q;

pz – заключительное состояние; pz ∈Q;

a0 – символ для обозначения пустой ячейки, a0 ∈A;

a1 – специальный символ - разделитель цепочек на ленте, a1 ∈A.

Командой машины Тьюринга называется элемент функции переходов qa → pbr, где q ирpQ; a и bA; rS. Каждая команда описывает один такт работы машины Тьюринга.Конфигурация машины Тьюринга представляется следующим образом: t = <CqaB>, гдеC - 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 <CqaB> непосредственно переходит в конфигурацию <CnqnanBn>, еслиновая конфигурация получена в результате применения одной команды к исходной конфигурации.Обозначим непосредственный переход из одной конфигурации в другую следующимобразом: <CqaB>⇒<CnqnanBn>.

Конфигурация, содержащая начальное состояние, называется начальной, а

содержащаязаключительное состояние - заключительной. Если цепочка С в конфигурации пустая, тоначальная и заключительная конфигурации называются стандартными.Машина Тьюринга перерабатывает цепочку х в цепочку у, если, действуя из начальнойконфигурации и имея на ленте цепочку х, машина Тьюринга переходит в заключительнуюконфигурацию, имея на ленте цепочку у. Если начальная и заключительная конфигурациявляются стандартными, то процесс переработки х в у является правильной переработкой.

Способы представления машины Тьюринга

Существует три способа представления машины Тьюринга: совокупностью команд,ввиде 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 пустуюячейку 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 соответствоватьпереходу в то состояние, которое соответствует началу цикла;
3) соседним шагам алгоритма соответствует переход в смежные состояния, которыесоответствуют 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, ε}, где

символ ε соответствует пустой ячейке, а число состояний устройства управления задано в виде множества Q = {q0, q1, q}

Если, например, начальная конфигурация имеет вид q0110011, то конечная конфигурация после завершения операции инвертирования должна иметь вид qz001100. Длярешения задачи машиной будет порождена следующая последовательность команд:

q01 → q00R, q10 → q10L,

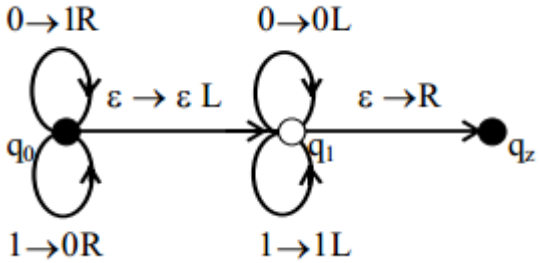
q00 → q01R, q11 → q11L,

q0 ε → q1εL, q1ε → qzεR.

В стандартной начальной конфигурации головка стоит над первым символом слева, иустройство управления находится в начальном состоянии. На следующем такте машинаТьюринга, не меняя своего состояния, заменяет символ 0 на 1 или 1 на 0 и сдвигается вправона один символ. После просмотра всей цепочки под головкой оказывается символ,указывающий на пустую ячейку. В этом случае машина Тьюринга переходит в новоесостояние и сдвигается влево на один символ. На последующих тактах управляющееустройство не меняет своего состояния, оставляет без изменения символ под головкой иперемещается влево до тех пор, пока не встретит пустую ячейку. Встретив пустую ячейку,машина Тьюринга переходит в заключительное состояние и перемещается вправо на одинсимвол, 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 изсимволов 0 и 1, будет представлена в виде графа следующим образом:



Начальная и конечная вершины графа обычно выделяются; на рисунке они отмеченычерным кружком. Если на очередном такте работы машины Тьюринга символ на ленте неизменяется, то в правой части команды его можно не писать (ε на ребре в состояние qz).

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	ε
q_0	q_0 1R	Q_0 0R	q_1 ε L
q_1	q_1 0L	Q_1 1L	q_z εL

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(x1, x2, ..., xn), если выполняютсяследующие условия:

1) для любых x1, x2, ..., xn, 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 x1, x2, ..., xn, не принадлежащих области определения функции, машинаТьюринга 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 существованиеалгоритма, необходимо построить машину Тьюринга, реализация которой на практикезачастую представляет собой трудоемкую задачу. В связи с этим возникает необходимостьразбиения алгоритма на отдельные задачи, каждая из которых будет решаться отдельноймашиной Тьюринга. Если объединить программы этих машин, то получится новаяпрограмма, позволяющая решить исходную задачу.

Машины Тьюринга могут вычислять искомую функцию с восстановлением и

безвосстановления. Вычисление функции с восстановлением означает работу машиныТьюринга с сохранением исходных данных:

p0 1X1* . . * 1Xn ⇒* p z 1f(X1, X2, . . . , Xn) *1X1* . . * 1Xn.

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

p0 1X1* . . * 1Xn ⇒* 1X1* . . * 1Xn * p z 1f(X1, X2, . . . , Xn) .

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

p0 1X1* . . * 1Xn ⇒* p z 1f(X1, X2, . . . , Xn) .

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

Операции над машинами Тьюринга

1. Композиция машин Тьюринга. Пусть машины T1 и T2 имеют программы P1 и P2.Предположим, что внутренние алфавиты этих машин не пересекаются; пусть qz1 - заключительное состояние машины T1, а q02 - начальное состояние машины T2. Заменим всюду в программе P1 заключительное состояние qz1 на начальное состояние q02 машины T2и полученную программу объединим с программой P2. Новая программа P определяет машину T, называемую композицией машин T1 и T2 по паре состояний (qz1, q02).

Композиция машин может быть обозначена T1 ·T2 или T1T2. Более подробно композициямашин записывается следующим образом:

T = T(T1, T2, (qz1, q02)),

где T1 = (Q1, A1, δ1, p01, pz1, a01, a11),

T2= (Q2, A2, δ2, p02, pz2, a02, a12).

Пусть a01 = a02 = a0 и a11 = a12 = a1. Внешний алфавит композиции T1T2 являетсяобъединением внешних алфавитов машин T1 и T2:

p02

T= (Q1 ∪Q2 \ {pz1}, A1 ∪A2, | (δ1,∪δ2), p01, pz2, a0, a1).

pz1.

Операция композиции, выполняемая над алгоритмами, позволяет получать новые, болеесложные 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. Заменим всюду в программе P машины T состояние qz haqn . 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 имеющаязаключительного состояния.

3. Разветвление машин Тьюринга. Пусть машины Тьюринга T1, T2 и T3 задаются программами P1, P2 и P3 соответственно. Считаем, что внутренние алфавиты этих машин попарно не пересекаются. Пусть qz11 и qz12 - какие-либо различные заключительные состояния машины T1. Заменим всюду в программе P1 состояние qz11 начальным состояниемq02 машины T2, а состояние qz12 начальным состоянием q03 машины T3. Затем новуюпрограмму объединим с программами P2 и P3. Получим программу P, задающую машинуТьюринга и обозначаемую:

T = T(T1, (qz11, q02), T2, (qz12, q03), T3).

Машина T называется разветвлением машин T2 и T3, управляемым машиной T1.

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

правиламсложения в двоичной системе сложения, выберем входной алфавит:

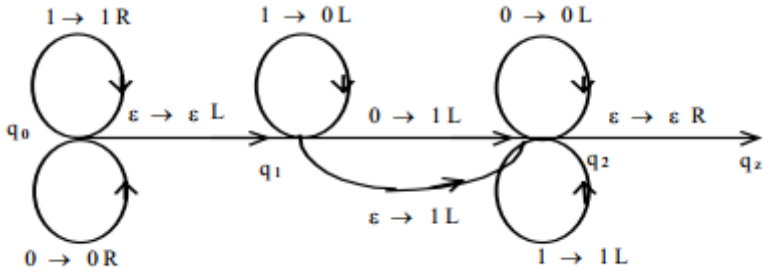
A ={0, 1, ε}.

Представим машины Тьюринга таблицей соответствия и графом.

Таблица соответствия:

	0	1	ε
q_0	q_0 0R	q_0 1R	Q_1 ε L
q_1	q_2 1L	q_1 0L	q_2 1L
q_2	q_2 0L	Q_2 1L	q_z ε R

Представление машины Тьюринга в виде графа:



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

1) q01 → q01R ε111ε 6) q11 → q10L ε110ε
2) q01 → q01R ε111ε 7) q11 → q10L ε100ε
3) q01 → q01R ε111ε 8) q1ε → q21L ε000ε
4) q0ε → q1εL ε111ε 9) q2 ε → qz εR ε1000ε
5) q11 → q10L ε111ε

Из примера видно, что машина Тьюринга из стандартной начальной конфигурации, имеяна 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 имеющуювходной алфавит A = {0, 1, ε}.

Таблица соответствия данной машины Тьюринга будет иметь следующий вид:

	0	1	ε		0	1	ε
q_0	q_0 0R	q_0 1R	q_1 ε L	q_2	q_3 1L	q_2 0L	q_3 1L
q_1	q_3 ε L	q_2 ε L		q_3	q_2 0L	q_2 1L	q_z ε 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,ε}. Представим данную машину таблицей соответствия:

	0	1	*	ε
q_0	$q_1$1L	$q_1$0R	q_0*L	q_zεR
q_1		$q_1$1R	q_2*R	q_2*R
q_2		$q_2$1R		$q_3$1L
q_3	$q_0$0R	$q_3$1L	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) q01 → q10R ε111ε
2) q11 → q11R ε011ε
3) q11 → q11R ε011ε
4) q1ε → q2*R ε011ε
5) q2ε → q31L ε011*ε
6) q3* → q3*L ε011*1ε
7) q31 → q31L ε011*1ε
8) q31 → q31L ε011*1ε
9) q30 → q00R ε011*1ε
10) q01 → q10R ε011*1ε
11) q11 → q11R ε001*1ε
12) q1* → q2*R ε001*1ε
13) q21 → q21R ε001*1ε
14) q2ε → q31L ε001*1ε
15) q31 → q31L ε001*11ε
16) q3* → q3*L ε001*11ε
17) q31 → q31L ε001*11ε
18) q30 → q00R ε001*11ε
19) q01 → q10R ε000*11ε
20) q1* → q2*R ε000*11ε
21) q21 → q21R ε000*11ε
22) q21 → q21R ε000*11ε
23) q2ε → q31L ε000*11ε
24) q31 → q31L ε000*111ε
25) q31 → q31L ε000*111ε
26) q3* → q3*L ε000*111ε
27) q30 → q00R ε000*111ε
28) q0* → q0*L ε000*111ε
29) q00 → q01L ε000*111ε
30) q00 → q01L ε001*111ε
31) q00 → q01L ε011*111ε
32) q0ε → qz1R ε111*111ε

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 ε111*111ε.

Машина Тьюринга с полулентой

В рассмотренных выше определениях машины Тьюринга использовалась бесконечная вобе стороны лента. Ограничим ленту с одной стороны и покажем, что машина Тьюринга справой или левой полулентой эквивалентна машине Тьюринга с бесконечной лентой.Говорят, что функция, правильно вычислимая на машине Тьюринга с обычной лентой,правильно вычислима и на машине Тьюринга с правой полулентой, т.е. для любой машиныТьюринга 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 иподвижным - справа;

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

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

Машина Тьюринга может вычислять функцию с восстановлением, то есть с

сохранениемисходных данных. Это позволяет получить на ленте сначала результат, а затем – исходныеданные или наоборот.

Рассмотрим пример построения машины Тьюринга с правой полулентой, вычисляющейфункцию f(x) = 2x. Алгоритм вычисления данной функции состоит в приписывании нуля квходной цепочке справа.

Таблица соответствия данной машины Тьюринга будет иметь следующий вид:

	<	0	1	>	ε
q_0		$q_0 0R$	$q_0 1R$	$q_1 0R$	
q_1					$q_2 >L$
q_2	$q_z < R$	$q_2 0L$	$q_2 1L$		

Здесь символ “<“ обозначает левый маркер, а символ “>“ - правый маркер. МашинаТьюринга, начав работу из стандартной начальной конфигурации, после выполнениясовокупности команд переходит в стандартную заключительную конфигурацию.

Полученный результат располагается между маркерами и сдвинут вплотную к левомумаркеру.

Задания для самостоятельной работы

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

4.1.

$q_0 1 \to q_0 1R$, $q_1 0 \to q_z 1E$,

$q_0 0 \to q_1 1R$, $q_1 1 \to q_1 1L$,

а) $K = 1101q_0 01$; б) $K = 101q_0 010$.

4.2.

$q_0 0 \to q_0 1L$, $q_1 1 \to q_0 0R$,

$q_0 1 \to q_1 1R$, $q_1 0 \to q_z 0L$,

а) $K = 1q_1 0111$; б) $K = 1q_1 1111$.

4.3.

$q_0 0 \to q_1 0L$, $q_1 0 \to q_1 1L$,

$q_0 1 \to q_0 0R$, $q_1 1 \to q_z 0R$,

а) $K = 1000q_1 01$; б) $K = 11q_1 11101$.

5. Построить машину Тьюринга, которая во входной цепочке, заданной в алфавите A ={0, 1, ε}, переставляет единицы и нули так, чтобы все единицы были в начале, а нули в концецепочки.
6. Построить композицию $T_1 \cdot T_2$ машин Тьюринга T_1 и T_2 по паре состояний (q_{1z}, q_{20}) инайти результат применения композиции $T_1 \cdot T_2$к слову D.

6.1. Машины T1 и T2 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 T1 и T2 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	$q_{21}0L$	$q_{22}0L$	$q_{2z}0R$
1	$q_{20}1L$	$q_{21}1L$	$q_{22}1L$

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 qz1 и qz2. 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 ≥ 1; б) i = 1, D = 1^{3k+2}, k ≥ 1.

7.2. МашинаТьюрингазаданатаблицейсоответствия:

T:

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

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. Компьютерныесети. Принципы, технологии, протоколы: учеб. пособие / В. Г. Олифер, Н. А. Олифер. - 4-е изд. - СПб.: Питер, 2011.
4. Таненбаум, Э. С. Архитектуракомпьютера / 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	**09.04.02 Информационные системы и технологии**
Направленность (профиль)	**«Технологии работы с данными и знаниями, анализ информации»**
Квалификация выпускника	Магистр

Пятигорск, 2025

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 данныхметодов 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 содержит 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 b1, b2, ..., bn:

а) алгоритм вычисления по формуле:

$$C=(...((A - b_1) - b_2) - ... - b_n)$$

б) алгоритм вычисления по формуле:

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

2. Разработать алгоритм вычисления по формуле:

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

3. Разработать алгоритм вычисления по формуле:

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

4. Разработать алгоритм поиска максимального (минимального) элемента из
последовательности, заданной в виде одномерного массива A = {a1, a2, ... , ak}.

5. Разработать алгоритм определения количества одинаковых чисел в последовательности, заданной в виде одномерного массива A={a1, a2, . . . , ak}.

Повышенный уровень

1. Разработать алгоритм подсчета количества одинаковых элементов в матрице ***B***
размерностью n × 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 = {a1, a2, . . ., ak} и B= {b1, b2}.

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

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

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

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

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 x1 > x2 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 5x5. 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 NxN 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, *, ε}.
8. Построить машину Тьюринга, выполняющую операцию копирования входной цепочки, заданной в алфавите A = {1, *, ε }, 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.

$q_0 1 \to q_0 1R$, $q_1 0 \to q_z 1E$,
$q_0 0 \to q_1 1R$, $q_1 1 \to q_1 1L$,
а) $K = 1101q_0 01$; б) $K = 101q_0 010$.

4.2.

$q_0 0 \to q_0 1L$, $q_1 1 \to q_0 0R$,
$q_0 1 \to q_1 1R$, $q_1 0 \to q_z 0L$,
а) $K = 1q_1\ 0111$; б) $K = 1q_1 1111$.

4.3.

$q_0 0 \to q_1 0L$, $q_1 0 \to q_1 1L$,
$q_0 1 \to q_0 0R$, $q_1 1 \to q_z 0R$,
а) $K = 1000q_1\ 01$; б) $K = 11q_1 11101$.

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

Повышенный 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 T1 и T2 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}
0	$q_{21}1L$	$q_{2z}0R$
1	$q_{21}1L$	$q_{20}0L$

T_2:

а) D = 111100111011; б) D = 11010111.

1.2.Машины T1 и T2 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	$q_{21}0L$	$q_{22}0L$	$q_{2z}0R$
1	$q_{20}1L$	$q_{21}1L$	$q_{22}1L$

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 qz1 и qz2. 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:

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 ≥ 1; б) i = 1, D = 1^{3k+2}, k ≥ 1.

2.2.7.2. Машина Тьюринга задана таблицей соответствия:

T:

	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 ≥ 1; б) i = 1, D = 1^{3k+2}, k ≥ 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^nb^n для n>0.
3. Построить КС-грамматику, порождающую язык $(ab)^nc^*a^{n+m}$ для n>0 и m>0.
4. Построить КС-грамматику, порождающую язык $(ab)^*ca^*b^{n+m}$ для n>0 и m>0.
5. Построить КС-грамматику, порождающую язык $a^nb^{3m}c^ma^{2n} = a^n(bbb)^mc^m(aa)^n$ для n>1 и m>1.

Повышенный уровень

1. Построить КС-грамматику, порождающую язык $(ab)^n(ca)^*b^{n+2}(abc)^*$для n>0.

2. Для леворекурсивной грамматики G: S→SabA | ba; A→AbA|bAa|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 → Aab | bSb | abB;

A→ abS | ba | bbA;

B → bB | Da | Aa;

D →Dbb|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. Этого же мнения 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 собойавтоматную модель, в основе которой лежит анализ процесса выполнения алгоритма каксовокупности набора инструкций.Третий тип алгоритмических моделей - это преобразования слов в произвольныхалфавитах, в которых элементарными операциями являются подстановки, т.е. замены частислова (подслова) другим словом. Преимущество этого типа состоит в его максимальнойабстрактности и возможности применить понятие алгоритма к объектам произвольнойприроды. 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 ai никакого выходного слова bj (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 черезззначения определяемой функции для меньших значений аргументов.Применение рекурсивных функций в теории алгоритмов основано на идее нумерациислов в произвольном алфавите последовательными натуральными числами. Наиболее простотакую нумерацию можно осуществить, располагая слова в порядке возрастания их длин, аслова, имеющие одинаковую длину, - в произвольном порядке.После нумерации входных и выходных слов в произвольном алфавитном операторе этотоператор превращается в функцию y = f(x), в которой аргумент x и функция y принимаютнеотрицательные целочисленные значения. Функция f(x) может быть определена не для всехзначений 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(x1, x2, ..., xn) = 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 n m 1, x2, . . ., xm, . . . , xn) = xm - 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(x1, x2, ..., xm), f1(x1, x2, ..., xn), f2(x1, x2, ..., xn), . . ., fm(x1, x2, ..., xn) можно получить новую функцию: Sm+1(f, f1, f2, ..., fm) = g(x1, x2, ..., xn) = f(f1(x1, x2, ..., xn), f2(x1, x2, ..., xn), ..., fm(x1, ..., xn)). (1)

В операции суперпозиции Sm+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(x2, x1, x3, . . ., xn) = f(I22(x1, x2), I12(x1, x2), x3, . . ., xn);

f(x1, x1, x3, . . ., xn) = f(I12(x1, x2), I12(x1, x2), x3, . . ., xn).

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 Rn 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(x1, x2, ..., xn, y) получается оператором примитивной рекурсии из функций g(x1, x2, ..., xn) и функции h(x1, x2, ..., xn, y, z), если:

f(x1, x2, ..., xn, 0) = g(x1, x2, ..., xn); (2)

f(x1, x2, ..., xn, y+1) = h(x1, x2, ..., xn, y, f(x1, x2, ..., xn, 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 x1, x2, ..., xn на момент применения схемы (2) зафиксированы и играют рольпараметров.

Оператор минимизации

Отношение P(x1, x2, ..., xn) называется примитивно-рекурсивным, если примитивно рекурсивна его характеристическая функция:

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

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

Функция f(x1, x2, ..., xn) получается посредством оператора минимизации из предиката P(x1, x2, ..., xn, z), если в любой точке значением функции f(x1, x2, ..., xn) является минимальное значение z, обращающее предикат P(x1, x2, ..., xn, z) в значение «истина»:

f(x1, x2, ..., xn) = μz (P(x1, x2, ..., xn, z)),

где μz – оператор минимизации.

2.2.4. Ограниченный оператор минимизации

Функция f(x1, x2, ..., xn) получается ограниченным оператором минимизации из предиката P(x1, x2, ..., xn, z) и функции U(x1, x2, ..., xn), если в любой точке значение этой функции определено следующим образом:

1) для любого z < U(x1, x2, ..., xn) такого, что P(x1, x2, ..., xn, z) = "истина", значение функции f(x1, x2, ..., xn) = μz(P(x1, ..., xn, z)),

2) не для любого z < U(x1, x2, ..., xn) такого, что P(x1, x2, ..., xn, z) = "истина", значение функции f(x1, x2, ..., xn) = U(x1, x2, ..., xn).

Данное определение записывается следующим образом:f(x1, x2, ..., xn) = μ z < U(x) (P(x1, x2, ..., xn, z)).Ограничение z в ограниченном операторе минимизации дает гарантию окончаниявычислений, поскольку оно оценивает сверху число вычислений предиката P. Возможностьоценить сверху количество вычислений является существенной особенностью примитивнорекурсивных 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 примитивнорекурсивной:

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 некоторыхусловий:

1) если исходные данные имеют рекурсивную структуру, то процедуры анализа такихструктур будут более эффективны, если они сами рекурсивны;

2) если алгоритм обработки некоторого набора данных построить, разбивая данные начасти 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: $f(x, 0) = x = I(x)$, $f(x, y+1) = f(x, y) + 1 = S(f(x, y)) = S(I3(x, y, f(x, y)))$.

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:

$p(x, 0) = Z(x)$; $p(x, y+1) = p(x, y) +x = S(p(x, y), x) = S(I (x, y, p(x, y)), 1\ 3\ I\ 3\ 3 (x, y, p(x, y)))$.

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 · 2x =2 ·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) = μz(2·x + z = y+1), откуда предикат P(x, y, z) = 2·x + z = y + 1.

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

Sg(Sg((2 · x + z) ÷(y + 1)) + Sg((y +1) ÷ (2 · x + z)).

Найдем значение функции в точке (1, 5):

При z = 0: P(1, 5, 0) = 2 · 1 + 0 = 5 + 1 - "ложь",

z = 1: P(1, 5, 1) = 2 · 1 + 1 = 5 + 1 - "ложь",

z = 2: P(1, 5, 1) = 2 · 1 + 2 = 5 + 1 - "ложь",

z = 3: P(1, 5, 1) = 2 · 1 + 3 = 5 + 1 - "ложь",

z = 4: P(1, 5, 4) = 2 · 1 + 4 = 5 + 1 - "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

О б щ и е с в е д е н и я

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 алгоритмоврешения различных задач.Описывая различные алгоритмы для машин Тьюринга и доказывая реализуемостьвсевозможных композиций алгоритмов, Тьюринг убедительно показал разнообразиевозможностей предложенной им конструкции, что позволило ему выступить со следующимтезисом: “Всякий алгоритм может быть реализован соответствующей машиной Тьюринга”.

Доказать тезис Тьюринга нельзя, так как в его формулировке не определено понятие “всякийалгоритм”. Его можно только обосновать, представляя различные алгоритмы в виде машинТьюринга. Было доказано, что класс функций, вычислимых на этих машинах, совпадает склассом частично рекурсивных функций.

Неформальное определение машины Тьюринга

Машина Тьюринга представляет собой автомат,имеющий бесконечную в обе стороны ленту, считывающую головку и управляющееустройство. Управляющее устройство может находиться в одном из состояний, образующихконечное множество Q = {q0, q1, ..., qn}. Множество Q называют внутренним алфавитоммашины Тьюринга

Принципиальное отличие машины Тьюринга от вычислительных машин состоит в том,что ее запоминающее устройство представляет собой бесконечную ленту, из-за которойневозможна ее физическая реализация. Лента разделена на ячейки, в каждой из которыхможет быть записан один из символов конечного алфавита A = {a0, a1, . . . , am}, которыйназывают входным алфавитом машины Тьюринга. Во время функционирования машиныТьюринга может быть заполнено конечное число ячеек. Считывающая головка в каждыймомент времени обозревает ячейку ленты, в зависимости от символа в этой ячейке исостояния управляющего устройства записывает в ячейку новый символ или оставляет егобез изменения, сдвигается на ячейку влево или вправо или остается на месте. При этомуправляющее устройство переходит в новое состояние или остается в старом. Средисостояний управляющего устройства выделены начальное состояние q0 и заключительноесостояние qz.

Таким образом, за один такт работы машина Тьюринга может считать символ, записатьвместо него новый или оставить его без изменения и сдвинуть головку на одну ячейку влевоили 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 |. Длинапустой цепочки равна нулю.

Конкатенацией цепочек X и Y называют цепочку, полученную приписыванием символовцепочки Y справа к цепочке X.

Машиной Тьюринга называется семерка видаT = (Q, A, δ, p0, pz, a0, a1), где
Q – конечное множество состояний, в которых может находиться управляющееустройство;

A – входной алфавит;

δ – функция переходов, δ = Q ·A → Q ·A ·S, где

S = {R, L, E} - направления сдвига;

p0 – начальное состояние; p0 ∈Q;

pz – заключительное состояние; pz ∈Q;

a0 – символ для обозначения пустой ячейки, a0 ∈A;

a1 – специальный символ - разделитель цепочек на ленте, a1 ∈A.

Командой машины Тьюринга называется элемент функции переходов qa → pbr, где q ирq∈Q; a и b∈A; r∈S. Каждая команда описывает один такт работы машины Тьюринга.Конфигурация машины Тьюринга представляется следующим образом: t = <CqaB>, гдеC - 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 <CqaB> непосредственно переходит в конфигурацию <CnqnanBn>, еслиновая конфигурация получена в результате применения одной команды к исходной конфигурации.Обозначим непосредственный переход из одной конфигурации в другую следующимобразом: <CqaB>⇒<CnqnanBn>.

Конфигурация, содержащая начальное состояние, называется начальной, а содержащаязаключительное состояние - заключительной. Если цепочка C в конфигурации пустая, тоначальная и заключительная конфигурации называются стандартными.Машина Тьюринга перерабатывает цепочку x в цепочку y, если, действуя из начальнойконфигурации и имея на ленте цепочку x, машина Тьюринга переходит в заключительнуюконфигурацию, имея на ленте цепочку y. Если начальная и заключительная конфигурацииявляются стандартными, то процесс переработки x в y является правильной переработкой.

Способы представления машины Тьюринга

Существует три способа представления машины Тьюринга: совокупностью команд,ввиде 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 пустуюячейку 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 соответствоватьпереходу в то состояние, которое соответствует началу цикла;
7) соседним шагам алгоритма соответствует переход в смежные состояния, которыесоответствуют 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, ε}, где
символ ε соответствует пустой ячейке, а число состояний устройства управления задано в виде множества Q = {q0, q1, q}

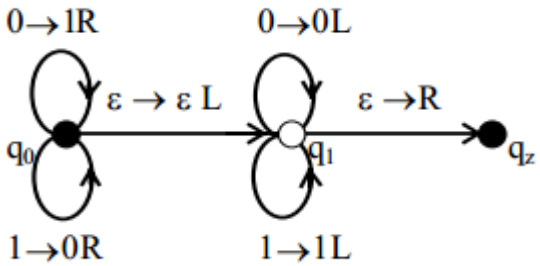
Если, например, начальная конфигурация имеет вид q0110011, то конечная конфигурация после завершения операции инвертирования должна иметь вид qz001100. Длярешения задачи машиной будет порождена следующая последовательность команд:

q01 → q00R, q10 → q10L,
q00 → q01R, q11 → q11L,
q0 ε → q1εL, q1ε → qzεR.

В стандартной начальной конфигурации головка стоит над первым символом слева, иустройство управления находится в начальном состоянии. На следующем такте машинаТьюринга, не меняя своего состояния, заменяет символ 0 на 1 или 1 на 0 и сдвигается вправона один символ. После просмотра всей цепочки под головкой оказывается символ,указывающий на пустую ячейку. В этом случае машина Тьюринга переходит в новоесостояние и сдвигается влево на один символ. На последующих тактах управляющееустройство не меняет своего состояния, оставляет без изменения символ под головкой иперемещается влево до тех пор, пока не встретит пустую ячейку. Встретив пустую ячейку,машина Тьюринга переходит в заключительное состояние и перемещается вправо на одинсимвол, 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 изсимволов 0 и 1, будет представлена в виде графа следующим образом:



Начальная и конечная вершины графа обычно выделяются; на рисунке они отмеченычерным кружком. Если на очередном такте работы машины Тьюринга символ на ленте неизменяется, то в правой части команды его можно не писать (ε на ребре в состояние qz).

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	ε
q_0	q_0 1R	Q_0 0R	q_1 ε L
q_1	q_1 0L	Q_1 1L	q_z εL

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(x1, x2, ..., xn), если выполняютсяследующие условия:

3) для любых x1, x2, ..., xn, 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 x1, x2, ..., xn, не принадлежащих области определения функции, машинаТьюринга 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 существованиеалгоритма, необходимо построить машину Тьюринга, реализация которой на практикезачастую представляет собой трудоемкую задачу. В связи с этим возникает необходимостьразбиения алгоритма на отдельные задачи, каждая из которых будет решаться отдельноймашиной Тьюринга. Если объединить программы этих машин, то получится новаяпрограмма, позволяющая решить исходную задачу.

Машины Тьюринга могут вычислять искомую функцию с восстановлением и безвосстановления. Вычисление функции с восстановлением означает работу машиныТьюринга с сохранением исходных данных:

p0 1X1* . . * 1Xn ⇒* p z 1f(X1, X2, . . . , Xn) *1X1* . . * 1Xn.

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

p0 1X1* . . * 1Xn ⇒* 1X1* . . * 1Xn * p z 1f(X1, X2, . . . , Xn) .

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

p0 1X1* . . * 1Xn ⇒* p z 1f(X1, X2, . . . , Xn) .

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

Операции над машинами Тьюринга

1. Композиция машин Тьюринга. Пусть машины T1 и T2 имеют программы P1 и P2.Предположим, что внутренние алфавиты этих машин не пересекаются; пусть qz1 - заключительное состояние машины T1, а q02 - начальное состояние машины T2. Заменим всюду в программе P1 заключительное состояние qz1 на начальное состояние q02 машины T2и полученную программу объединим с программой P2. Новая программа P определяет машину T, называемую композицией машин T1 и T2 по паре состояний (qz1, q02).

Композиция машин может быть обозначена T1 ·T2 или T1T2. Более подробно композициямашин записывается следующим образом:

T = T(T1, T2, (qz1, q02)),

где T1 = (Q1, A1, δ1, p01, pz1, a01, a11),

T2= (Q2, A2, δ2, p02, pz2, a02, a12).

Пусть a01 = a02 = a0 и a11 = a12 = a1. Внешний алфавит композиции T1T2 являетсяобъединением внешних алфавитов машин T1 и T2:

p02

T= (Q1 ∪Q2 \ {pz1}, A1 ∪A2, | (δ1,∪δ2), p01, pz2, a0, a1).

pz1.

Операция композиции, выполняемая над алгоритмами, позволяет получать новые, болееосложные 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. Заменим всюду в программе P машины T состояние qz haqn . 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 имеющаязаключительного состояния.

3. Разветвление машин Тьюринга. Пусть машины Тьюринга T1, T2 и T3 задаются программами P1, P2 и P3 соответственно. Считаем, что внутренние алфавиты этих машин попарно не пересекаются. Пусть qz11 и qz12 - какие-либо различные заключительные состояния машины T1. Заменим всюду в программе P1 состояние qz11 начальным состояниемq02 машины T2, а состояние qz12 начальным состоянием q03 машины T3. Затем новуюпрограмму объединим с программами P2 и P3. Получим программу P, задающую машинуТьюринга и обозначаемую:

T = T(T1, (qz11, q02), T2, (qz12, q03), T3).

Машина T называется разветвлением машин T2 и T3, управляемым машиной T1.

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 правиламсложения в двоичной системе сложения, выберем входной алфавит:

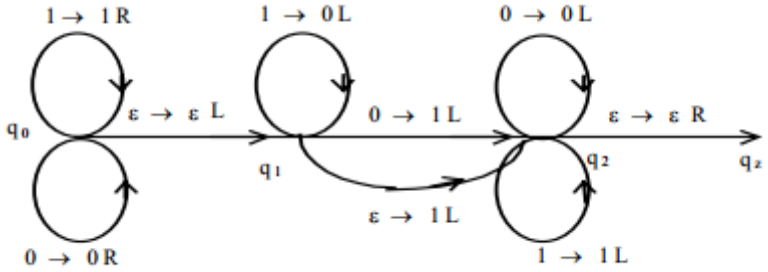
A ={0, 1, ε}.

Представим машины Тьюринга таблицей соответствия и графом.

Таблица соответствия:

	0	1	ε
q_0	q_0 0R	q_0 1R	Q_1 ε L
q_1	q_2 1L	q_1 0L	q_2 1L
q_2	q_2 0L	Q_2 1L	q_z ε R

Представление машины Тьюринга в виде графа:



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

6) q01 → q01R ε111ε 6) q11 → q10L ε110ε
7) q01 → q01R ε111ε 7) q11 → q10L ε100ε
8) q01 → q01R ε111ε 8) q1ε → q21L ε000ε
9) q0ε → q1εL ε111ε 9) q2 ε → qz εR ε1000ε
10) q11 → q10L ε111ε

Из примера видно, что машина Тьюринга из стандартной начальной конфигурации, имеяна 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 имеющуювходной алфавит A = {0, 1, ε}.

Таблица соответствия данной машины Тьюринга будет иметь следующий вид:

	0	1	ε		0	1	ε
q_0	q_0 0R	q_0 1R	q_1 ε L	q_2	q_3 1L	q_2 0L	q_3 1L
q_1	q_3 ε L	q_2 ε L		q_3	q_2 0L	q_2 1L	q_z ε 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,ε}. Представим данную машину таблицей соответствия:

	0	1	*	ε
q_0	$q_1$1L	$q_1$0R	q_0*L	q_zεR
q_1		$q_1$1R	q_2*R	q_2*R
q_2		$q_2$1R		$q_3$1L
q_3	$q_0$0R	$q_3$1L	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) q01 → q10R ε111ε
18) q11 → q11R ε011ε
19) q11 → q11R ε011ε
20) q1ε → q2*R ε011ε
21) q2ε → q31L ε011*ε
22) q3* → q3*L ε011*1ε
23) q31 → q31L ε011*1ε
24) q31 → q31L ε011*1ε
25) q30 → q00R ε011*1ε
26) q01 → q10R ε011*1ε
27) q11 → q11R ε001*1ε
28) q1* → q2*R ε001*1ε
29) q21 → q21R ε001*1ε
30) q2ε → q31L ε001*1ε
31) q31 → q31L ε001*11ε
32) q3* → q3*L ε001*11ε

17) q31 → q31L ε001*11ε
18) q30 → q00R ε001*11ε
19) q01 → q10R ε000*11ε
20) q1* → q2*R ε000*11ε
21) q21 → q21R ε000*11ε
22) q21 → q21R ε000*11ε
23) q2ε → q31L ε000*11ε
24) q31 → q31L ε000*111ε
25) q31 → q31L ε000*111ε
26) q3* → q3*L ε000*111ε
27) q30 → q00R ε000*111ε
28) q0* → q0*L ε000*111ε
29) q00 → q01L ε000*111ε
30) q00 → q01L ε001*111ε
31) q00 → q01L ε011*111ε
32) q0ε → qz1R ε111*111ε

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 ε111*111ε.

Машина Тьюринга с полулентой

В рассмотренных выше определениях машины Тьюринга использовалась бесконечная вобе стороны лента. Ограничим ленту с одной стороны и покажем, что машина Тьюринга справой или левой полулентой эквивалентна машине Тьюринга с бесконечной лентой.Говорят, что функция, правильно вычислимая на машине Тьюринга с обычной лентой,правильно вычислима и на машине Тьюринга с правой полулентой, т.е. для любой машиныТьюринга 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 иподвижным - справа;

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

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

Машина Тьюринга может вычислять функцию с восстановлением, то есть с

сохранениемисходных данных. Это позволяет получить на ленте сначала результат, а затем – исходныеданные или наоборот.

Рассмотрим пример построения машины Тьюринга с правой полулентой, вычисляющейфункцию f(x) = 2x. Алгоритм вычисления данной функции состоит в приписывании нуля квходной цепочке справа.

Таблица соответствия данной машины Тьюринга будет иметь следующий вид:

	<	0	1	>	ε
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. МашинаТьюринга, начав работу из стандартной начальной конфигурации, после выполнениясовокупности команд переходит в стандартную заключительную конфигурацию.

Полученный результат располагается между маркерами и сдвинут вплотную к левомумаркеру.

5.4. ФОРМАЛЬНЫЕ ГРАММАТИКИ И ЯЗЫКИ

Общие сведения

В последние три десятилетия появилось большое количество работ по общей теориияязыков и грамматик. Можно выделить четыре научных направления, которые удалосьобъединить 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 былисформулированы вопросы машинного перевода. Эта проблема требовала формализациипонятий словарь, грамматика, язык, их классификации и умения относить конкретныесловари, грамматики, языки к определенному классу.Интерес к задачам такого рода еще более усилился с появлением искусственных языковпрограммирования. 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 какомулибо 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 автоматики.

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.

a. Основные понятия порождающих грамматик

Алфавит - это непустое конечное множество. Элементы алфавита называются символами. Цепочка над алфавитом Σ={a1, a2, …, an} есть конечная последовательность элементов ai. Множество всех цепочек над алфавитом Σ обозначается Σ*.

Длина цепочки x равна числу ее элементов и обозначается |x|. Цепочка нулевой длиныназывается пустой и обозначается символом ε. Соответственно, непустая цепочкаопределяется как цепочка ненулевой длины. Пусть имеется алфавит Σ={a, b}. Тогдамножество всех цепочек определяется следующим образом: Σ*={ ε, a, b, aa, ab, ba, bb, aaa,aab, aba, . . .}.

Цепочки x и y равны, если они одинаковой длины и совпадают с точностью до порядкасимволов, 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 Σ представляет собой множество цепочек над Σ. Необходимо различать пустой язык L=0 и язык, содержащий только пустую цепочку: L={ε}≠0. Формальный язык L над алфавитом Σ - это язык, выделенный с помощью конечного множества некоторых формальных правил.

Пусть M и L - языки над алфавитами. Тогда конкатенация LM = {xy|x∈ L, y∈ M}. В частности, {ε}L = L{ε}=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:

L0 = {ε};

L1 = L;

Ln+1 = Ln L = L Ln;

{ε}L=L{ε}=L.

Например, еслизаданязык$L=\{a\}$, тогда$L^*=\{\varepsilon, a, aa, aaa, \ldots\}$, $L+=\{a, aa, aaa, \ldots\}$. 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 $x = \alpha u \beta$, $y = \alpha v \beta$ и $u \rightarrow v \in P$, т.е. цепочка y непосредственно выводится из x, что обозначается x => y.

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

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

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

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

S → ab;

S → aSb => aabb;

S→ aSb =>aaSbb =>aaabbb; . . .

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 → aSb | 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) леволинейные (леворекурсивные), правила вывода для которых имеют следующий вид: A → Aa | a, где $A \in VN$;

б) праволинейные (праворекурсивные), правила вывода для которых имеют следующий вид: A → Aa | 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 → aB, B → Cd | dC, C → e}. Выписать терминальные цепочки, порожденные данной грамматикой, и определить длину их вывода.

Решение. Получим следующие терминальные цепочки:

S → aB → aCd → aed,

S → aB → adC → 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 → ed, B → Ab, C → Bc, C → dD, D → Ae, E → bc}. Определить,принадлежит ли языку L(G) цепочка eadabcb.

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

C → Bc → Abc → edbc,

C → dD → dAe → 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 → 0S1, S → 1S0, S → ε, S → S01, S → S10}.

Пример 4. Описать язык, порождаемый следующими правилами: S → bSS | a.

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

S → a;

S → bSS → baa;

S → bSS → bbSSS → bbaaa;

S → bSS → bbSSbSS → bbaabaa;

S → bSS → bbSSbSS → bbbSSbSSbbSSbSS → ...

Изполученныхцепочеквидно, 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.

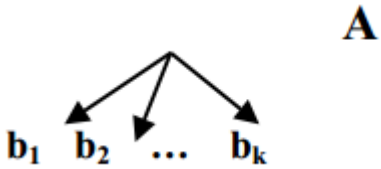
Исходя из этих выводов, определим язык, порождаемый заданными правилами, следующим образом:

L(G) = {α | α ∈ {a, b}*}, |a| = |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 → ScS|b|a возможны следующие эквивалентные выводы терминальной цепочки acb: S → ScS →Scb → acb и S → ScS → acS → acb.

Деревом вывода цепочки x в КС-грамматике G = (VT, VN, P, S) называется упорядоченное дерево, каждая вершина которого помечена символом из множества V ∪VN∪ {ε} так, что каждому правилу A → b1b2b3 ... bk, использующемуся при выводе цепочки x, в дереве вывода соответствует поддерево с корнем A и прямыми потомками b1, b2, b3, ..., bk, использующемуся при выводе цепочки x, в дереве вывода соответствует поддерево с корнем A и прямыми потомками b1, b2, b3, ..., bk, которое приведено на рисунке:



В силу того, что цепочка x ∈ L(G) выводится из аксиомы S, корнем вывода всегдаявляется аксиома. Внутренние узлы вывода соответствуют нетерминальным символам.Концевые вершины дерева вывода - листья - это вершины, не имеющие потомков. Такиевершины соответствуют либо терминалам, либо пустым символам (ε) при условии, что средиправил грамматики имеются правила с пустой правой частью. При чтении слева направоконцевые вершины дерева вывода образуют цепочку x.

Дерево вывода часто называют деревом грамматического разбора, илисинтаксическим деревом, а процесс построения дерева вывода - грамматическим разбором(синтаксическим анализом). Одной цепочке языка может соответствовать больше одногодерева, 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 ∈ L(G), имеющая два или более дерева вывода.

Рассмотрим пример.

Пусть даны две КС-грамматики:

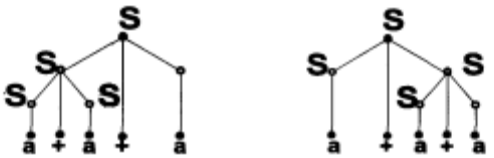
G1 = (VT, VN, P1, S), VN = {S},

P1={S → S+S | S | S· S | (S) | a};

G2= (VT, VN, P2, S), VN = {S, A, B},

P2 = {S → S + A | A | A, A → A· B | A / B | B, B → a| (S)}, содержащие в множестве терминальных символов знаки операций, круглые скобки и символ a. Определить, являютсяли грамматики однозначными. Если какая-либо из них неоднозначна, привести примерцепочки, для которой существует два различных дерева вывода.

Решение: Грамматика G1 является неоднозначной, так как она имеет правила с правойчастью, содержащей два вхождения нетерминала S и знак арифметической операции.Построим два дерева вывода для простейшей цепочки a + a + a:



Грамматика G2 является однозначной, так как не содержит правил с двойнымвхождением нетерминального символа. Так, для цепочки a + a + a она имеет только однодерево вывода.

В некоторых случаях неоднозначность в грамматиках может устраняться путемэквивалентных преобразований. Однако в общем случае проблема устранениянеоднозначности неразрешима. На практике от неоднозначности избавляются путемзадания словесных ограничений, называемых контекстными условиями языка, которыевключаются в его семантику.

Различают две стратегии грамматического разбора: восходящую и нисходящую, которыесоответствуют способу построения синтаксических деревьев. При нисходящей стратегииразбора дерево строится от корня аксиомы вниз, к терминальным вершинам. Главнойзадачей при нисходящем разборе является выбор того правила, которое следует применитьна рассматриваемом шаге. При восходящем разборе дерево строится от терминальныхвершин 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 (любыецепочки, 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 → aSa | bSb | c}.

Преобразования КС-грамматик

Часто требуется изменить грамматику таким образом, чтобы она удовлетворялаопределенным требованиям, не изменяя при этом порождаемый грамматикой язык. Дляэтого используются эквивалентные преобразования КС-грамматик, некоторые из которыхрассмотрены ниже.

Удаление правил вида A → B

Преобразование первого типа состоит в удалении правил A → B, или <нетерминал>→<нетерминал>.

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

Пусть имеется КС-грамматика G=(VT, VN, P, S), где множество нетерминалов VN={A1,A2, . . . , An}. Разобьем P на два непересекающихся множества: P = P1 ∪P2. В P1 включенывсе правила вида Ai → Ak, в P2 включены все остальные правила, т.е. P2 = P \ P1. Затем длякаждого Ai определим множество правил P(Ai), включив в него все такие правила Ai→ϕ, чтоAi →* Aj и Aj → ϕ, где Aj → ϕ ∈ P2. Построим эквивалентную КС-грамматику Gэ = (VT, VN,Pэ, S), в которой множества терминальных и нетерминальных символов, а также аксиомасовпадают с теми же объектами исходной грамматики G, а множество правил Pэ полученообъединением правил множества P2 и правил P(Ai) для всех 1≤ i ≤n:

Пример. Пусть задана грамматика G со следующими правилами вывода S → aFb | A; A→ aA | B; B → aSb | S; F → bc | bFc.Построим множества правил P2, P(S), P(A), P(B), P(F).

Определим правила для P2: P2 = {S → aFb; A → aA; B → aSb; F → bc | bFc}.Определим правила для P(S): S => A => B или S =>*A; S=>B, где =>* обозначаетнепосредственную выводимость. P(S) = {S → aA; S → aSb).Определим правила для P(A): A => B => S или A =>* B; A => S. P(A) = {A → aSb; A →aFb}.

Определим правила для P(B): B => S => A или B =>* S; B =>*A. P(B) = {B → aFb; B →aA}.

Определим правила для P(F): так как непосредственно выводимых нетерминалов несуществует, то P(F) = 0.

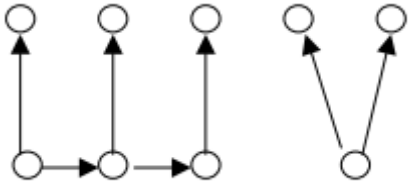
Объединив полученные правила, можно записать грамматику Gэ, эквивалентнуюисходной:

S → aFb | aSb | aA; A → aA | aSb | aFb;B → aA | aSb | aFb; F → bc | bFc.

Графическая модификация метода

Аналитическое преобразование по рассмотренному алгоритму оказывается довольносложным. При автоматизированном преобразовании грамматик проще применитьграфическую модификацию этого метода. С этой целью каждому нетерминальному символуи каждой правой части правил из множества P2 поставлена в соответствие вершина графа.

Из вершины с меткой U в вершину с меткой V направлено ребро, если в грамматикесуществует правило U → V.



В эквивалентную грамматику будут включены правила вида A → w, A∈ VN; w∈ (VT ∪VN)*, если из вершины с меткой A существует путь в вершину с меткой w.

S → aFb | aSb | aA;

A → aA | aSb | aFb;

B → aA | aSb | aFb;

F → bc | bFc.

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

Построение неукорачивающей грамматики

Грамматика, не содержащая правил с пустой правой частью, называетсянеукорачивающей грамматикой.

В грамматике с правилами вида A→ε длина выводимой цепочки при переходе от k-го шага к (k+1)-му уменьшается. Поэтому грамматики с правилами вида A→ε называютсяукорачивающими. Восходящий синтаксический разбор в укорачивающих грамматикахсложнее по сравнению с разбором в неукорачивающих грамматиках, т.к. при редукциинеобходимо отыскать такой фрагмент входной цепочки, в которую можно вставить пустойсимвол.

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

W1= {A | A→ε ∈ P},

Wn+1 = Wn ∪ {B | B→ϕ ∈ P, ϕ ∈ W*n }.

Затем найдем множество правил эквивалентной грамматики в два этапа:

а) удалив из множества P исходной грамматики правила с пустой правой частью P1' P\{A→ε | A→ε ∈ P};

б) получив новые правила A→ϕ' после удаления из каждого правила исходнойграмматики A→ϕ ∈ P те нетерминалы, которые вошли в множество Wn по правилу:P1" = { A→ϕ' | A→ϕ ∈ P'; ϕ =ϕ1Bϕ2 | B∈ Wn; ϕ'=ϕ1Bϕ2}.

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

Пример. Пусть задана грамматика G со следующими правилами вывода: S → AbA | cAb |Bb; A → aAb | ε; B→AA|a. Необходимо:

1) построить множество нетерминалов, из которых выводится ε;
2) построить неукорачивающую грамматику, эквивалентную исходной.

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

W1 = {A | A → ε ∈ P};

Wm+1 = Wm ∪ {B | B → ϕ ∈ P, ϕ ∈ W*m}.

Так как мы имеем правило A → ε ∈ P, то можно построить множество W1 = {A},включающее нетерминал A.

Построим множество W2. С нетерминалом A связан нетерминал B, т.е. существуетправило B → AA и A ∈ W1. Следовательно, W2={A,B}.W3 = W2, т. к. множество W3={B|B → ϕ ∈ P, ϕ ∈ W*m } является пустым.

Исключив правило, содержащее пустую цепочку в правой части, получимнеукорачивающую грамматику G1 следующего вида:

S → AbA | cAb | Bb | bA | Ab | cb | b;

A → aAb | ab;
B→ AA | A | a.

Построение грамматики с продуктивными нетерминалами

Нетерминальный символ A называется непродуктивным (непроизводящим), если он непорождает ни одной терминальной цепочки, т.е. не существует вывода A →* x, где x∈ V*T.

Поэтому представляет интерес удаление из грамматики всех непродуктивныхнетерминальных символов. Рассмотрим, как для произвольной КС-грамматики можнопостроить эквивалентную КС-грамматику, все нетерминальные символы которойпродуктивны. С этой целью выделяется множество W1={A | A → ϕ ∈ P, ϕ ∈ V*T}. Затемстроится множество W1, W2, . . . , Wn+1 по следующим правилам:Wn+1 = Wn ∪ {B□B → x ∈ P, x ∈ (VT ∪ Wn)*

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

W1 = {A}, т.к. в множестве P есть правило A → bb;

W2 = {A, S}, т.к. имеется правило S → bAb и A ∈ W1;

W3 = W2.

Все символы множества VN\Wn являются непродуктивными, не используются в выводеникакой терминальной цепочки и их можно удалить из грамматики вместе со всемиправилами, в которые они входят. Грамматика G1, эквивалентная исходной грамматике,будет включать следующее множество правил:

S → SA | bAb; A → aSa | bb.

В грамматике G1 все нетерминалы продуктивны.

Построение грамматики, аксиома которой зависит от всех нетерминалов

Существует еще один тип нетерминальных символов, которые можно удалять изграмматики вместе с правилами, в которые они входят. Например, в грамматике G, заданнойсистемой правил P: S → aSb | ba; A → aAa | bba, нетерминал A не участвует ни в какомвыводе, т.к. из аксиомы нельзя вывести цепочку, содержащую этот нетерминал. Поэтому иззаданной грамматики можно удалить нетерминал A. Рассмотрим, как для произвольной КСграмматики можно построить эквивалентную КС-грамматику, аксиома которой зависит отвсех нетерминальных символов.

Для этого построим множество нетерминалов, от которых зависит аксиома. С этойцелью выделим множества W1, W2, . . . , Wn+1 по следующим правилам:

W1 = {S},

Wn+1 = Wn∪ {B | A→ xBy∈ P, A∈ Wn}.

Нетерминал B∈ Wn тогда и только тогда, когда аксиома S зависит от B. Все нетерминалы,не содержащиеся в Wn, можно удалить из грамматики вместе с правилами, в которые онивходят.

Пример. Пусть дана КС-грамматика G:

S → abS | ASa | ab; A → abAa | ab; B → bAab | bB.

Найдем нетерминалы, от которых зависит аксиома:

W1 = {S},

W2 = {S, A}, т.к. имеется правило S → Asa и S ∈ W1;

W3 = W2.

Эквивалентная грамматика G1, аксиома которой зависит от всех нетерминальныхсимволов:

S → abS | ASa | ab; A → abAa | ab.

Удаление правил с терминальной правой частью

Пусть в грамматике G имеется с терминальной правой частью A→β, где β ∈ V*T. Тогдалюбой вывод с использованием этого правила имеет вид: S →* xAy → xβy. Здесьнетерминал A в сентенциальной форме xAy появился на предыдущем шаге вывода B →uAv.

Если в это правило вместо A подставить β, получим правило B → uβv. Тогда длина выводанекоторой цепочки сократится на один шаг. Следовательно, для того чтобы удалитьтерминальное правило грамматики A→β, необходимо учесть следующие правила:

а) если для нетерминала A больше нет правил, тогда во всех правых частях A заменяетсяна β;

б) если для нетерминала A есть другие правила, тогда добавляются новые правила, вкоторых A заменяется на β.

Пример. Пусть дана КС-грамматика G:

S → aSb | bAa | B; A → ABa | b | ε; B→ ab | ba.

Удалим правила для нетерминала B. Тогда эквивалентная грамматика G1 будет включатьследующие правила:

S→ aSb | bAa | ab | ba; A→ Aaba | Abaa| b | ε.

Построение эквивалентной праворекурсивной КС-грамматики

Некоторые специальные методы грамматического разбора неприменимы клеворекурсивным или праворекурсивным грамматикам, поэтому рассмотрим устранениелевой или правой рекурсии. В общем случае избавиться от рекурсии в правилах грамматикиневозможно, т.к. бесконечные языки порождаются грамматиками с конечным числом правилтолько благодаря рекурсии. Поэтому можно говорить лишь о преобразовании одного видарекурсии в другой. Рассмотрим, как для любой леворекурсивной КС-грамматики построитьэквивалентную праворекурсивную КС-грамматику.

Пусть нетерминал A - леворекурсивен, т.е. для него имеются правила следующего вида:A → Ax1 | Ax2 | . . . | Axp | w1| . . . | wk, где xi и wj - цепочки над множеством VT ∪VN.Введем дополнительные нетерминалы B и D и указанные правила заменим наэквивалентные им:

A → AB | D;

B → x1 | x2 | . . . | xp;

D → w1| w2 | . . . | wk.

В результате замены получим вывод, который имеет вид:

A → AB→ ABB → ABBB → . . . → AB* → DB*.

Таким образом, для нетерминала A можно определить эквивалентные правила:

A → DK;

K → BK | ε.

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

A → w1K| w2K | . . . | wkK;

K → x1K | x2K | . . . | xpK | ε.

Пример. Пусть задана грамматика G со следующими правилами вывода: S → Sa | ba.

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

S → SB | D;

B → a; D → ba. Вывод из S имеет вид: S → SB→ SBB→ . . . →DB*.

Запишем эквивалентные правила для S:

S → DK; K → BK | ε.

Подставим в эти правила B и D и получим эквивалентную праворекурсивнуюграмматику G1:

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

Рассмотрим вывод цепочки baaaa в исходной леворекурсивной грамматике G:

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

Эту же цепочку можно вывести в эквивалентной праворекурсивной грамматике G1:

$S \rightarrow baK \rightarrow baaK \rightarrow baaaK \rightarrow baaaaK \rightarrow baaaa$.

5.5. ТЕОРИЯ АВТОМАТОВ

Понятие автомата. Типы автоматов

Автомат - это алгоритм, определяющий некоторое множество и, возможно,преобразующий его в другое множество. Неформальное описание автоматов выглядитследующим образом: автомат имеет входную ленту, управляющее устройство с конечнойпамятью 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 вцепочку y при условии, что $x \in L$.

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

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

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

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

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

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, котораявключает 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 длякаждой

конфигурации существует конечное множество возможных следующих тактов,любой из которых автомат может сделать, исходя из этой конфигурации. Управляющееустройство будет детерминированным, если для каждой конфигурации будет возможентолько 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 собойлинейную 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, δ, γ), где

K - множество состояний (алфавит состояний);

X - входной алфавит;

Y - выходной алфавит;

δ - функция переходов, задающая отображение K· X→K;

γ - функция выходов, задающая отображение K· X→У.

Функционирование автомата можно задать множеством команд вида qx → py, где q и p∈ K, x ∈ X, y ∈ Y.

Пусть на некотором такте ti управляющее устройство находится в состоянии q, а извходной ленты читается символ x. Если в множестве команд есть команда qx → py, то натакте ti на выходную ленту записывается символ y, а к следующему такту ti+1 управляющееустройство перейдет в состояние p, т.е.:

y(t) = γ(q(t), x(t)), q(t+1) =δ(q(t), x(t)).

Если же команда qx → py отсутствует, то автомат оказывается блокированным и нереагирует на символ, принятый в момент ti, а также перестает воспринимать символы впоследующие моменты времени.

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

Если зафиксировано некоторое начальное состояние, то такой автомат называютинициальным, т.е. q(0) = q0.

Инициальный автомат - это шестерка вида A = (K, X, Y, δ, γ, q0), где

K - множество состояний (алфавит состояний);

X - входной алфавит;

Y - выходной алфавит;

δ - функция переходов (отображение K · X → K);

γ - функция выходов (отображение K · X → У);

q0 - начальное состояние.

Распознаватели. Языки и автоматы

Задача грамматического разбора заключается в нахождении вывода цепочки в заданнойграмматике 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. Простейшимтипом автоматов являются конечные автоматы.Конечный автомат имеет входную ленту, с которой за один такт может быть считан одинвходной символ. Возврат по входной ленте не допускается.

Конечным автоматом называется пятерка вида A = (K, ∑, δ, p0, F), где

K - конечное множество состояний;

∑ - алфавит;

δ - функция переходов;

p0 - 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 такжематрицей переходов.

Регулярные множества

Регулярные множества образуют класс языков, имеющих важное значение для теорииформальных языков. Рассмотрим несколько методов задания языков, каждый из которыхопределяет регулярные множества. Среди них - регулярные выражения, праволинейныеграмматики, детерминированные и недетерминированные конечные автоматы.

Пусть Σ - некоторый алфавит. Регулярное множество в алфавите Σ определяетсярекурсивно следующим образом:

1) 0 - пустое множество;

2) {ε} - множество из пустой цепочки;

3) {a} - регулярное множество для каждого элемента a ∈ Σ;

4) если P и Q - регулярные множества в алфавите Σ, то регулярными являютсямножества:

а) P ∪ Q

б) PQ

в) P*.

Других регулярных множеств в алфавите Σ нет. Таким образом, некоторое множество цепочек в заданном алфавите Σ называется регулярным тогда и только тогда, когда либо оно является одним из множеств: 0, {ε} или {a} для некоторого a ∈ Σ, 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, читаемыхавтоматом при переходе из начального состояния в одно из заключительных состояний:

$L(A)=\{a_1 a_2 \ldots a_n \mid p_0a_1 \rightarrow p_1, p_1a_2 \rightarrow p_2, \ldots, p_{n-1}a_n \rightarrow p_n; p_n \in F\}$.

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, ∑, δ, p0, F) - произвольный конечный автомат. Построим автомат:

A1 = (K ∪ {q0}, Σ, δ ∪ {q0a → pi | p0a → pi ∈ δ}, q0, F ∪ {q0 | p0 ∈ F}).

Любая цепочка x = a1 a2 ... ak принадлежит языку L(A) тогда и только тогда, когда существует следующая последовательность команд автомата A:

p0a1 → p1; p1a2 → p2; . . . ; ph-1ah → ph, ph ∈ F

и соответствующая ей последовательность команд автомата A1:

q0a1 → p1; p1a2 → p2; . . . ; pk-1ak → ph.

Таким образом, имеем: A = A1.

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, ∑, δ, p0, F) новый автомат A1: A1 = (K∪ {f}, Σ, δ∪ {qja → f | pja → pi∈ δ& pi∈ F}, p0, {f}∪ {p0 | p0 ∈ F}).

В результате имеем: A = A1.

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 выполненияперечисленных 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 {ε}∪L∪L2∪. . . =L*.

2. Операция произведения над L(A1) и L(A2) выполняется с помощью двух преобразований: а) удаляются циклы из начального состояния A2 и заключительного состояния A1; б) каждому заключительному состоянию A1 ставим в соответствие свой экземпляр A2 и объединяем заключительные состояния A1 с начальным состоянием соответствующего экземпляра A2.

3. Объединение L(A1) и L(A2) строится с помощью удаления циклов в начальных состояниях A1 и A2 и объединения полученных начальных состояний.

4. Усеченная итерация может быть построена двумя способами:

а) L(A1)+ = L(A1)* L(A1), б) L(A1)+ = L(A1) L(A1)*.

5. Рассмотрим дополнение L(A1) до Σ*. Пусть автомат A1 детерминированный, тогда любая цепочка x=a1a2 . . an распознается по единственному маршруту:

p0a1 → p1 p1a2 → p2

. . .

pn-1an→ pn, pn∈ F.

Автомат не распознает только те цепочки, которые:

1) либо представляют собой начальную часть цепочки a1a2 . . aj, при чтении которой автомат переходит в состояние, не являющееся заключительным;

2) либо имеют вид y = a1a2 . . akbc1c2 . . cm (k < n), где начало цепочки a1a2 . . ak совпадает с началом цепочки x∈ L(A1), но за символом ak стоит такой символ b, что автомат A1 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 символамалфавита, не читаемым в этом состоянии; в) в построенном дополнительном состоянии построить петли для всех символов алфавита, чтобы обеспечить чтение произвольного окончания цепочки c1c2 . . cm. 6. Разность L(A1) и L(A2) строится в соответствии со следующим преобразованием: $L(A1) \setminus L(A2) = L(A1) \cap L(A2)$.

7. Операция пересечения строится в соответствии со следующим преобразованием: $L(A1) \cap L(A2) = L(A1) \cup L(A2)$. 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→aB поставим в соответствие команду Aa → B, а каждому терминальному правилу A → a поставим в соответствие команду Aa → F.

Таким образом, выводу цепочки в грамматике

S ⇒a1A1 ⇒a1a2A2 ⇒... ⇒a1a2...ak-1Ak-1⇒a1a2...ak взаимно однозначно соответствует последовательность команд построенного автомата A:

Aa1 → A1; A1a2 → A2; . . . ; Ak-1ak → F.

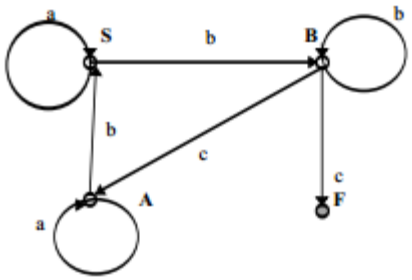
Таким образом, язык L(G) = L(A).

Пример. Пусть для заданной грамматики G требуется построить конечный автомат.

S → aS | bB;

A → aA | bS;

B → bB | c | 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 → S - в начальном состоянии при поступлении на вход терминального символа a автомат остается в том же состоянии S;

Sb → B - из начального состояния при поступлении на вход терминала b автомат переходит в состояние B;

Bb → B - в состоянии B при поступлении на вход терминала b автомат остается в том же состоянии B;

Bc → F - из состояния B при поступлении на вход терминала c автомат переходит в заключительное состояние F;

Bc → A - из состояния B при поступлении на вход терминала c автомат переходит в состояние A;

Aa → A - в состоянии A при поступлении на вход терминала a автомат остается в этом же состоянии A;

Ab → 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 соответствиенетерминал грамматики, причем начальному состоянию будет соответствовать аксиома. Тогда для каждой команды Ac → B в множество правил грамматики включим правило A→ cB, причем в случае, если B - заключительное состояние, добавим правило A → c. Эквивалентность исходного конечного автомата и построенной грамматики очевидна.

Теорема. Для каждой леворекурсивной грамматики существует эквивалентныйконечный 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 → Ba поставим в соответствие команду автомата Ba →A. Тогда каждому выводу в грамматике:

S⇒A1ak⇒A2ak-1ak⇒... ⇒Ak-1a2a3...ak ⇒a1a2 . . . ak однозначно соответствует следующая последовательность команд построенного автомата A:

Na1 → Ak-1; . . .; A2ak-1 → A1; A1ak → 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 → B в множество правил включим соответствующее правило B → Aa, причем в том случае, если B - заключительное состояние, дополнительно введем правило S → Aa, а если A - начальное состояние, то дополнительно введем правило B → a.

Тогда последовательности команд:

A0a1 → A1; A1a2 → A2; . . . ;

Ak-1ak → F

соответствует следующий вывод:

S⇒Ak-1ak⇒... ⇒A1a2a3...ak⇒a1a2a3...ak.

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 → aA;

A → bS | b.

Проверим, действительно ли эта грамматика порождает язык L(A). Для этого построим несколько выводов возможных вариантов цепочек:

1) S ⇒aA ⇒ab;

2) S ⇒aA ⇒abS ⇒abaA ⇒abab;

3) S ⇒aA ⇒abS ⇒abaA ⇒ababS ⇒ababaA ⇒abababab; 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 → A - из состояния S при поступлении на вход терминала a автомат переходит в состояние A;

Ab → S - из состояния A при поступлении на вход терминала a автомат переходит в состояние S;

Ab → 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, Σ, Г, δ, p0, F, B0), где

K - конечное множество состояний;

Σ - алфавит; Г - алфавит магазина;

δ - функция переходов;

p0 - начальное состояние;

F - множество заключительных состояний;

B0 - символ из множества Г для обозначения маркера дна магазина.

В общем случае данное определение соответствует недетерминированному автомату. В отличие от конечного автомата для произвольного МП-автомата нельзя построитьэквивалентный 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, Г, δ, p0, F, B0), где Г = VT ∪VN ∪{B0},

K = {p0, F}, F = {f}.

Функция переходов δ будет содержать следующие команды:

а) p0, a, ε → p0, a - длялюбых a ∈VT;

б) p0, ε, ϕ’ → p0, A - для всех правил A → ϕ ∈P, где ϕ’ - зеркальное отображение ϕ;

в) p0, ε, SB0 → f, B0. В общем случае команда выглядит так:

pi, σ, γ → pj, λ , где pi ∈K – состояние автомата до выполнения команды, σ∈Vt – символ на входной ленте, γ∈Г- символ верхушки магазина, pj ∈K – состояние автомата после выполнения команды, λ∈Г- 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.

Пустьданаграмматика G:

S → S+A | S/A | A A → a | (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:
   p0, a, ε → p0, a ;
   p0, +, ε → p0, + ;
   p0, /, ε → p0, / ;
   p0, (, ε → p0, (;
   p0,), ε → p0,).

Эти команды обеспечивают занесение терминального символа из входной ленты в магазин.

2. Команды редукции по правилам грамматики:
   p0, ε, A+S → p0, S ;
   p0, ε, A/S → p0, S ;
   p0, ε, A → p0, S ;
   p0, ε,)S(→ p0, A ;
   p0, ε, a → p0, 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:
   p0, ε, SB0 → f, B0.

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 а г а з и н	B_0	a B_0	A B_0	S B_0	/ S B_0	(/ S B_0	a (/ S B_0	A (/ S B_0	S (/ S B_0	+ S (/ S B_0	a + S (/ S B_0	A + S (/ S B_0	S (/ S B_0 |) S (/ S B_0	A / S B_0	S B_0	B_0
p	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	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:
   p0, ε, ε → p1, S;
2) для каждого правила A→ϕ∈ P нетерминал в верхушке магазина заменяется на правую часть правила с помощью команды: p1, ε, A → p1, ϕ;
3) для каждого терминала a∈ VT выполняется сравнение символа на входной ленте с символом в верхушке магазина и его поглощение: p1, a, a →p1, ε
4) разбор заканчивается по команде: p1,ε,B0 → f, B0.

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: p0, ε, ε → p0, S;
2) команды замены нетерминала правой частью правила:
p1, ε, S → p1, S+A
p1, ε, S → p1, S/A
p1, ε, S → p1, A
p1, ε, A → p1, a
p1, ε, A→ p1, (S);
3)команды сравнения и поглощения символа с входной ленты и символа в верхушке магазина:
p1, a, a → p1, ε
p1, +, + → p1, ε
p1, /, / → p1, ε
p1, (, (→ p1, ε p1,),) → p1, ε;

4)команда завершения разбора: p1, ε, B0 → f, B0. Процесс разбора цепочки представлен в таблице 2.

Таблица 2

вход			a	/		(				a	+		a	)	
магазин	B_0	S B_0	S / A B_0	a / A B_0	a / A B_0	/ A B_0	A B_0	(S) B_0	S) B_0	S + A) B_0	A + A) B_0	a + A) B_0	+ A) B_0	A) B_0	a) B_0	) B_0	B_0
p	p_0	p_1	p_1	p_1	p_1	p_1	p_1	p_1	p_1	p_1	p_1	p_1	p_1	p_1	p_1	p_1	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 придетсяиспользовать 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, q0), где K - конечное множество состояний;

X - входной алфавит;

Y - выходной алфавит;

f - функция переходов;

g - функция выходов;

q0 - 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, q0) - конечный преобразователь. Тогда отображение S(x) = g(q0, 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(q0, x) зафиксируем начальное состояние q0, 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 скоторого 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, порождаемые 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 = {q0, q1, q2, q3, q4},

2. X = {a,+, −}, 3. Y = X.

Преобразователь M начинает работу в состоянии q0 и, чередуя состояния q0 и q4 на входном символе «−», определяет, четное или нечетное число знаков − предшествует первому символу a. Когда появляется a, преобразователь M переходит в состояние q1, 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 q2 и q3. Единственное различие между парами q2, q3 и q0, q4 состоит в том, что если символу a предшествует четное числоминусов, то первая из них выдает +a, а не только a.

Таблица переходов выглядит следующим образом:

	Y			K		
K\X	a	+	–	a	+	–
q_0	a	ε	ε	q_1	q_0	q_4
q_1	-	ε	ε	-	q_2	q_3
q_2	+a	ε	ε	q_1	q_2	q_3
q_3	–a	ε	ε	q_1	q_3	q_2
q_4	–a	ε	ε	q_1	q_4	q_0

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 · X → K);

h - функция выходов (отображение K · X → 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: ∀ a ∈ X, ∀ b ∈ X, ∀ z ∈ 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 одногосостояния 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 | qZ ∈ K};

L(U) = {ht | ht ∈ 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 = x1 x2 ... xn - входная цепочка, тогда множества реакций для автоматов M и U будут соответственно представлены следующим образом:

q0 / y1, x1 → q1 / y2, x2 → ... → qn-1 / yn, xn;

q0, x1 / y1 → q1, x2 / y2 → ... → qn-1, xn / yn.

Теорема. Для любого автомата Мили можно построить эквивалентный автомат Мура.

Доказательство. В качестве множества K1 автомата Мура возьмем K1 = K · Y. Для обеспечения равносильности автоматов M = U функции переходов и выходов определим следующим образом:

f1 (p · y, a) = {qb | f(p, a) = q, b ∈ X};

h(p · y) = y.

Если реакция автомата M на входную цепочку вида w = x1 x2 ... xn из состояния q0 имеет вид

q0, x1 / y1 → q1, x2 / y2 → ... → qk-1, xk / yk (1),

то существует такое состояние q0· x1 недетерминированного автомата U, что, начиная работу из этого состояния, автомат U выполняет следующие действия:

q0 · x1 / y1 → q1 · x2 / y2 → ... → qk-1 · xk / yk, xk (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 оканчивающиесясимволом "т", т.е. такие, как "бит", "байт", "батут" и т.д. При построении автомата всебуквы 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. Компьютерныесети. Принципы, технологии, протоколы: учеб. пособие / В. Г. Олифер, Н. А. Олифер. - 4-е изд. - СПб.: Питер, 2011.
6. Таненбаум, Э. С. Архитектуракомпьютера / 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