Федеральное агентство по образованию
Государственное образовательное учреждение высшего профессионального образования
Амурский государственный университет
(ГОУВПО «АмГУ»)

УТВЕРЖДАЮ
Зав. кафедрой ИиУС
_____ А.В. Бушманов
«___» _____ 2007 г.

МАТЕМАТИЧЕСКАЯ ЛОГИКА И ТЕОРИЯ АЛГОРИТМОВ

Учебно-методический комплекс дисциплины

для специальности

230102 – Автоматизированные системы обработки
информации и управления;

Составитель:
Семичевская Н.П.

2007 г.

Печатается по решению
редакционно-издательского совета
факультета математики
и информатики
Амурского государственного
университета

Математическая логика и теория алгоритмов для специальности 230102 «Автоматизированные системы обработки информации и управления»: учебно-методический комплекс дисциплины. / Семичевская Н.П. – Благовещенск. Изд-во Амурского гос. ун-та, 2007. 109с.

ОГЛАВЛЕНИЕ

1. ВЫПИСКА ИЗ ГОСУДАРСТВЕННОГО ОБРАЗОВАТЕЛЬНОГО СТАНДАРТА ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

Направление подготовки дипломированного специалиста

654600 – Информатика и вычислительная техника

Специальность

230102 – Автоматизированные системы обработки информации и управления.

Квалификация – *инженер*

Индекс	Наименование дисциплин и их основные разделы	Всего часов
ЕН.Ф.01.04	*Математическая логика и теория алгоритмов.*	
	Логика высказываний; логика предикатов; исчисления; непротиворечивость; полнота; синтаксис и семантика языка логики предикатов. Клаузальная форма. Метод резолюций в логике предикатов. Принцип логического программирования. Темпоральные логики; нечеткая и модальные логики; нечеткая арифметика; алгоритмическая логика Ч. Хоара. Логика высказываний. Логическое следование, принцип дедукции. Метод резолюций. Аксиоматические системы, формальный вывод. Метатеория формальных систем. Понятие алгоритмической системы. Рекурсивные функции. Формализация понятия алгоритма; Машина Тьюринга. Тезис Черча; Алгоритмически неразрешимые проблемы. Меры сложности алгоритмов. Легко и трудноразрешимые задачи. Классы задач P и NP. NP – полные задачи. Понятие сложности вычислений; эффективные алгоритмы. Основы нечеткой логики. Элементы алгоритмической логики.	100

2. РАБОЧАЯ ПРОГРАММА

Федеральное агентство по образованию РФ
АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
(ГОУВПО «АмГУ»)

УТВЕРЖДА
Ю
Проректор по УНР
_____Е.С.Астапова
«___»_____2005 г.

РАБОЧАЯ ПРОГРАММА

по дисциплине "Математическая логика и теория алгоритмов"
для специальности 230102 Автоматизированные системы обработки информации и управления
курс 2 семестр 3

Лекции 36 (час.) Экзамен

Практические (семинарские) занятия 18 (час.) Зачет 3 семестр

Лабораторные занятия___(час.)

Самостоятельная работа 36 (час.)

Всего часов 90 час.

Составитель Семичевская Наталья Петровна

Факультет Математики и информатики

Кафедра Информационных и управляющих систем

2005 г.

Рабочая программа составлена на основании Государственного образовательного стандарта ВПО по специальности 230102 «Автоматизированные системы обработки информации и управления»

Рабочая программа обсуждена на заседании кафедры Информационных и управляющих систем

« _____» _____2005 г., протокол №_____

Заведующий кафедрой_____ А.В.Бушманов

Рабочая программа одобрена на заседании УМС 230102 «Автоматизированные системы обработки информации и управления»
«_____» _____2005 г., протокол №_____

Председатель_____ А.В.Бушманов

СОГЛАСОВАНО
Начальник УМУ
_____Г.Н.Торопчина

«____»_____2005 г.

СОГЛАСОВАНО
Председатель УМС факультета
_____Е.Л.Еремин

«_____»_____2005 г.

СОГЛАСОВАНО
Заведующий выпускающей кафедрой
_____А.В.Бушманов

«____»_____2005 г.

Математическая логика и теория алгоритмов

1. Цели и задачи дисциплины, ее место в учебном процессе.

В современной математической науке исследования в областях, традиционно относящихся к дискретной математике, занимают все более заметное место. Это объясняется необходимостью создания и эксплуатации современных электронных вычислительных машин, средств передачи и обработки информации, автоматизированных систем управления и проектирования.

Цель преподавания

Цель преподавания дисциплины «Математическая логика и теория алгоритмов» научить студентов основам математической логики и теории алгоритмов. В настоящее время наряду с такими классическими разделами математики, как математический анализ, дифференциальные уравнения в учебных планах технических и инженерных специальностей появились разделы по математической логике, алгебре, комбинаторике, теории графов и сетей. В связи с этим представляется целесообразным преподавание для студентов специальности «Автоматизированные системы обработки информации и управления» дисциплины, которая бы объединила в себе два раздела из дискретной математики, которые расширили бы рамки курса «Дискретная математика».

В курсе «Математическая логика и теория алгоритмов» излагаются основы логики высказываний и логики предикатов, принципы построения формальных теорий: исчисления высказываний и исчисления предикатов. С помощью машин Тьюринга и частично рекурсивных функций уточняются понятия алгоритма и вычислимости.

Перечень дисциплин, усвоение которых студентами необходимо при изучении дисциплины: «Математический анализ» (теория функций), «Линейная алгебра и геометрия» (алгебра матриц), «Теория графов», «Дискретная математика» (булева алгебра, алгебра логики, логика предикатов).

2. Содержание дисциплины

2.1. Федеральный компонент

Дисциплина «Математическая логика и теория алгоритмов» является дисциплиной, входящей в блок общематематических и естественно научных дисциплин федерального компонента для специальности 230202 «Автоматизированные системы обработки информации и управления». Государственный стандарт – ЕН.Ф.01.04.

2.2 Содержание лекций - 36 час.

Математическая логика

Тема 1. История логики. Основные разделы логики. Темпоральные логики. Нечеткая и модальные логики. (2ч.)

Тема 2. Аксиоматические системы. Формальный вывод. Принципы построения формальных теорий. (2ч.)

Тема 3. Исчисление высказываний. Исчисление высказываний и алгебра логики. Логика высказываний. Логическое следование, принцип дедукции. Метод резолюций. (4ч.)

Тема 4. Исчисление предикатов и теории первого порядка. Выводимость и истинность. Предваренная нормальная форма. Метод резолюций в логике предикатов. Теории первого порядка: исчисление с равенством; исчисление частичного нестрогого порядка. Формальная арифметика. (6ч.)

Тема 5. Метатеория логических исчислений. Непротиворечивость. Полнота, разрешимость. Теорема Геделя. Аксиоматический метод. (4ч.)

Теория алгоритмов

Тема 6. Понятие алгоритма. Основные требования к алгоритмам. Блок-схемы алгоритмов. Меры сложности алгоритмов. Понятие алгоритмической системы. Легко и трудно разрешимые задачи. (4ч.)

Тема 7. Машина Тьюринга. Основные определения, представления машины Тьюринга. Примеры машин. Операции над машинами Тьюринга. Композиция машин Тьюринга. Понятие Универсальной машины Тьюринга. Тезис Тьюринга. Проблема остановки. Алгоритмически неразрешимые проблемы. (6ч.)

Тема 8. Рекурсивные функции. Примитивно-рекурсивные функции. Примитивно-рекурсивные операторы. Общерекурсивные и частично-рекурсивные функции. Связь рекурсивных функций с машинами Тьюринга. Тезис Черча. (6ч.)

Тема 9. Понятие сложности вычислений; эффективные алгоритмы. Основы нечеткой логики. Элементы алгоритмической логики. (2ч.)

2.3. Практические занятия - 18час.

1. Логика высказываний. Логическое следование, принцип дедукции. Метод резолюций. (2ч.)
2. Исчисление высказываний. Аксиомы и правила вывода. (2ч.)
3. Основные метатеоремы исчисления высказываний. Теорема дедукции. (2ч.)
4. Исчисление предикатов и теории первого порядка. Аксиомы и правила вывода. Правила переименования свободных и связанных переменных. (2ч.)
5. Префиксная нормальная форма в исчислении предикатов. (2ч.)
6. Примеры машин Тьюринга. Описание машин T_{+}, $T_{коп}$, T_{2x}, T_{++}. (2ч.)
7. Композиция машин Тьюринга. Пример машин T_{++}, $T_{\times}$. Проблема остановки.(2ч.)
8. Рекурсивные функции. Примитивно-рекурсивные функции (сложение, умножение, возведение в степень, урезанное вычитание, min, max). (2ч.)
9. Примитивно-рекурсивные операторы (оператор условного перехода, ограниченный оператор наименьшего числа (μ -оператор)). (2ч.)

2.4. Самостоятельная работа студентов - 36 ч.

В течение семестра студентами должны быть выполнены и сданы следующие промежуточные виды контрольных работ:

1. Контрольная работа по «Математической логике» №1
2. Контрольная работа по «Теории алгоритмов» №2
3.Коллоквиум по «Математической логике»

Вопросы к коллоквиуму по «Математической логике»

1. Понятие формальной системы.
2. Принципы построения формальных теорий.
3. Вывод формулы В. Доказательство формулы В.
4. Теоремы теории Т. Доказательство теорем в теории.
5. Исчисление высказываний. Принципы построения теории.
6. Формулы схемы формул исчисления высказываний.
7. Аксиомы исчисления высказываний. Аксиоматический метод.
8. Правила вывода в исчислении высказываний.
9. Теорема дедукции. Применение теоремы дедукции.
10. Применение теоремы дедукции. Метод доказательства от противного.
11. Теорема о выводимости формулы в исчислении высказываний. Пример.
12. Теоремы о тождественной истинности в исчислении высказываний. Примеры.
13. Исчисление предикатов и теории первого порядка. Принципы построения.
14. Аксиомы исчисления предикатов.
15. Правила вывода в исчислении предикатов.
16. Выводимость и истинность в исчислении предикатов.
17. Эквивалентности, выводимые в исчислении предикатов. Предваренная форма или префиксная нормальная форма ПНФ.
18. Пример теории первого порядка исчисления с равенством.
19. Теория первого порядка исчисления частичного нестрогого порядка.
20. Формальная арифметика.

Вопросы к зачету
по дисциплине «Математическая логика и теория алгоритмов»

Часть I. Математическая логика.

1. Понятие формальной системы.
2. Принципы построения формальных теорий.
3. Вывод формулы В. Доказательство формулы В.
4. Теоремы теории Т.
5. Исчисление высказываний. Принципы построения теории.
6. Формулы схемы формул исчисления высказываний.
7. Аксиомы исчисления высказываний.
8. Правила вывода в исчислении высказываний.
9. Теорема дедукции. Применение теоремы дедукции.
10. Применение теоремы дедукции. Метод доказательства от противного.
11. Теорема о выводимости формулы в исчислении высказываний. Пример.
12. Теоремы о тождественной истинности в исчислении высказываний. Примеры.
13. Полнота и замкнутость теории. Теорема Геделя.

14. Исчисление предикатов и теории первого порядка. Принципы построения.
15. Аксиомы исчисления предикатов.
16. Правила вывода в исчислении предикатов.
17. Выводимость и истинность в исчислении предикатов.
18. Эквивалентности, выводимые в исчислении предикатов. Предваренная форма или префиксная нормальная форма ПНФ.
19. Пример теории первого порядка исчисления с равенством.
20. Теория первого порядка исчисления частичного нестрогого порядка. Формальная арифметика (принцип индукции).

Часть II. Теория алгоритмов.

21. Понятие алгоритма. Основные требования к алгоритмам.
22. Определение машины Тьюринга. Пример.
23. Конфигурация или полное состояние машины Тьюринга. Стандартная начальная конфигурация, Стандартная заключительная конфигурация.
24. Память машины Тьюринга, данные Машины Тьюринга, детерминированность машины Тьюринга.
25. Представления машин Тьюринга. Система команд, построение таблицы переходов, построение диаграммы переходов машин Тьюринга. Примеры.
26. Понятие функции правильно вычислимой по Тьюрингу.
27. Машина Тьюринга вычисляющая сложение (T_{+}).
28. Машина Тьюринга вычисляющая копирование ($T_{коп}$).
29. Машина Тьюринга - композиция машин, вычисляющая функцию $f(x)=2x$ ($T_{+}(T_{коп})$).
30. Машина Тьюринга с правой полулентой. Пример.
31. Вычисление предикатов на машинах Тьюринга.
32. Алгоритмически неразрешимые проблемы Проблема остановки. Тезис Тьюринга.
33. Рекурсивные функции. Примитивно-рекурсивные функции (f_{+}, $f_{\times}$, $f_{\div}$).
34. Рекурсивные функции. Примитивно-рекурсивные функции (f_{min}, f_{max}).
35. Примитивно-рекурсивные операторы.
36. Связь рекурсивных функций с машинами Тьюринга. Тезис Черча.

Зачет принимается по двум частям: 1) Математическая логика; 2) Теория алгоритмов. На зачете надо ответить на два теоретических вопроса и решить одну практическую задачу. Студенты, которые сдали 1 часть на коллоквиуме, сдают зачет только по второй части: один вопрос теоретический и практическая задача.

Критерий оценки ответа на зачете: зачет ставится за знание теории и умение решать задачи по математической логике и теории алгоритмов; не зачет получают студенты, которые в течение семестра 1) не сдавали работы по текущему контролю в семестре, 2) не ответили на теоретические зачетные вопросы, 3) не смогли решить практическую задачу.

3. Учебно-методические материалы по дисциплине

Основная литература

1. Москинова Г.И. Дискретная математика. Математика для менеджера в примерах и упражнениях: Уч.пособие. – М.: Логос, 2000. - 240с.
2. Асеев Г. Г. Дискретная математика [Текст]: учеб. пособ. / Г.Г. Асеев, О.М. Абрамов, Д.Э. Ситников. - Ростов н/Д: Феникс; Харьков: Торсинг, 2003. - 144 с.
3. Элементы теории алгоритмов и язык программирования С [Текст]: учеб. пособие: Рек. УМО Моск. физико-техн. ин-та / В.Я. Митницкий. - М.: Изд-во Моск. физико-техн. ин-та, 2001. - 180 с.
4. Игошин В. И. Задачи и упражнения по математической логике и теории алгоритмов [Текст]: учеб. пособ.: рек. Мин. обр. РФ / В. И. Игошин. - М.: Академия, 2005. -304 с.
5. Нефедов В.Н., Осипова В.А. Курс дискретной математики. М., 1992.

Дополнительная литература

1. Никифоров А. Книга о логике. М.: «Гнозис», 1996.
2. Манин Ю.И. Доказуемое и недоказуемое. М.: «Советское радио», 1979.
3. Смаллиан Р. Алиса в стране смекалки. М.: Мир, 1987.
4. Гарднер М. А ну-ка, догадайся! М.: Мир, 1984.
5. Кориков А.М., Сафьянова Е.Н. Основы системного анализа и теории систем: Учебное пособие. – Томск: изд-во Том. ун-та, 1989. – 207 с.
6. Основы кибернетики. Математические основы кибернетики / Под. ред. К.А. Пупкова. – М.: Высш. школа, 1974. – 416 с.
7. Мальцев А.И. Алгоритмы и рекурсивные функции. 2-е изд. – М.: Наука, 1986. –368 с.
8. Кузин Л.Т. Основы кибернетики.: В 2 т. Т. 2. Основы кибернетических моделей. – М.: Энергия, 1979. – 584 с.
9. Риордан Дж. Введение в комбинаторный анализ. – М. ИЛ, 1963. – 288 с.
10. Рыбников К.А. Введение в комбинаторный анализ. – М.: Изд-во МГУ, 1985. – 312 с.
11. Шевелев Ю.П. Высшая математика 5. Дискретная математика. Ч.1: Теория множеств. Булева алгебра (для автоматизированной технологии обучения): Учебное пособие. – Томск: Том. гос. ун-т систем управления и радиоэлектроники, 1998. – 114 с.
12. Шевелев Ю.П. Высшая математика 6. Дискретная математика. Ч.2: Теория конечных автоматов. Комбинаторика. Теория графов (для автоматизированной технологии обучения): Учебное пособие. – Томск: Том. гос. ун-т систем управления и радиоэлектроники, 1999. – 120 с.

4.УЧЕБНО-МЕТОДИЧЕСКАЯ (ТЕХНОЛОГИЧЕСКАЯ) КАРТА ДИСЦИПЛИНЫ

«Математическая логика и теория алгоритмов»

Номер недели	Номер темы	Вопросы, изучаемые на лекции	Занятия (номера)		Используемые нагляд. и метод. пособия	Самостоятельная работа студентов		Форма контроля
			Практич (семин.)	Лаборат.		Содержание	часы	
1	2	3	4	5	6	7	8	9
1	1	1,2,3,4	1					
2	2	1,2,3					2	к/р№1
3	3	1,2	2					
4	3	3,4,5						
5	4	1,2	3					
6	4	3,4						
7	4	5,6	4					
8	5	1,2						Тест
9	5	3,4,5	5				10	
10	6	1,2,3				Коллокв.	10	
11	6	4,5,6	6					
12	7	1,2,3					2	к/р№2
13	7	4,5,6	7					
14	7	7,8,9						
15	8	1,2	8					К/р№3
16	8	3,4					10	
17	9	5,6	9					Тест
18	9	1,2,3					2	Зач.

К.р. № 1 «Исчисление высказываний, исчисление предикатов»

К.р. № 2 «Машины Тьюринга. Протокол машины Тьюринга»

К.р. №3 «Примитивно-рекурсивные функции»

3. ГРАФИК САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТОВ

Содержание	Объем в часах	Сроки и форма контроля
Контрольная работа №1	2	
Контрольная работа №2	2	
Выполнение домашних работ	10	
Подготовка к тестированию	8	
Подготовка к коллоквиуму	10	
Подготовка к зачету	4	
Итого	36	

4. МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ПРОВЕДЕНИЮ САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТОВ

По каждой практической работе студенты выполняют домашнюю работу, требования к выполнению домашних работ указаны в методических разработках к практическим работам.

В течение семестра студенты выполняют контрольные работы по тематике предложенной в рабочей программе. Подготовка к контрольной работе предусматривает изучение материалов лекции и демонстрацию умения решать предложенные задачи в контрольной работе.

5. ПЕРЕЧЕНЬ УЧЕБНИКОВ, УЧЕБНЫХ ПОСОБИЙ

Основная литература

1. Москинова Г.И. Дискретная математика. Математика для менеджера в примерах и упражнениях: Уч.пособие. – М.: Логос, 2000. - 240с.
2. Асеев Г. Г. Дискретная математика [Текст]: учеб. пособ. / Г.Г. Асеев, О.М. Абрамов, Д.Э. Ситников. - Ростов н/Д: Феникс; Харьков: Торсинг, 2003. - 144 с.
3. Элементы теории алгоритмов и язык программирования С [Текст]: учеб. пособие: Рек. УМО Моск. физико-техн. ин-та / В.Я. Митницкий. - М.: Изд-во Моск. физико-техн. ин-та, 2001. - 180 с.
4. Игошин В. И. Задачи и упражнения по математической логике и теории алгоритмов [Текст]: учеб. пособ.: рек. Мин. обр. РФ / В. И. Игошин. - М.: Академия, 2005. -304 с.
5. Нефедов В.Н., Осипова В.А. Курс дискретной математики. М., 1992.

Дополнительная литература

6. Никифоров А. Книга о логике. М.: «Гнозис», 1996.
7. Манин Ю.И. Доказуемое и недоказуемое. М.: «Советское радио», 1979.
8. Смаллиан Р. Алиса в стране смекалки. М.: Мир, 1987.
9. Гарднер М. А ну-ка, догадайся! М.: Мир, 1984.
10. Кориков А.М., Сафьянова Е.Н. Основы системного анализа и теории систем: Учебное пособие. – Томск: изд-во Том. ун-та, 1989. – 207 с.
11. Основы кибернетики. Математические основы кибернетики / Под. ред. К.А. Пупкова. – М.: Высш. школа, 1974. – 416 с.
12. Мальцев А.И. Алгоритмы и рекурсивные функции. 2-е изд. – М.: Наука, 1986. –368 с.
13. Кузин Л.Т. Основы кибернетики.: В 2 т. Т. 2. Основы кибернетических моделей. – М.: Энергия, 1979. – 584 с.
14. Риордан Дж. Введение в комбинаторный анализ. – М. ИЛ, 1963. – 288 с.
15. Рыбников К.А. Введение в комбинаторный анализ. – М.: Изд-во МГУ, 1985. – 312 с.
16. Шевелев Ю.П. Высшая математика 5. Дискретная математика. Ч.1: Теория множеств. Булева алгебра (для автоматизированной технологии обучения): Учебное пособие. – Томск: Том. гос. ун-т систем управления и радиоэлектроники, 1998. – 114 с.
17. Шевелев Ю.П. Высшая математика 6. Дискретная математика. Ч.2: Теория конечных автоматов. Комбинаторика. Теория графов (для автоматизированной технологии обучения): Учебное пособие. – Томск: Том. гос. ун-т систем управления и радиоэлектроники, 1999. – 120 с.

6.КОНСПЕКТ ЛЕКЦИЙ

Тематический план лекций

№	Содержание лекций	Часы
	Математическая логика	
Тема 1	История логики. Основные разделы логики. Темпоральные логики. Нечеткая и модальные логики.	2
Тема 2	Аксиоматические системы. Формальный вывод. Принципы построения формальных теорий.	2
Тема 3	Исчисление высказываний. Исчисление высказываний и алгебра логики. Логика высказываний. Логическое следование, принцип дедукции. Метод резолюций.	4
Тема 4	Исчисление предикатов и теории первого порядка. Выводимость и истинность. Предваренная нормальная форма. Метод резолюций в логике предикатов. Теории первого порядка: исчисление с равенством; исчисление частичного нестрогого порядка. Формальная арифметика.	6
Тема 5	Метатеория логических исчислений. Непротиворечивость. Полнота, разрешимость. Теоремы Геделя. Аксиоматический метод.	4
	Теория алгоритмов	
Тема 6	Понятие алгоритма. Основные требования к алгоритмам. Блок-схемы алгоритмов. Меры сложности алгоритмов. Понятие алгоритмической системы. Легко и трудно разрешимые задачи.	4
Тема 7	Машина Тьюринга. Основные определения, представления машины Тьюринга. Примеры машин. Операции над машинами Тьюринга. Композиция машин Тьюринга. Понятие Универсальной машины Тьюринга. Тезис Тьюринга. Проблема остановки. Алгоритмически неразрешимые проблемы.	6
Тема 8	Рекурсивные функции. Примитивно-рекурсивные функции. Примитивно-рекурсивные операторы. Общерекурсивные и	6

	частично-рекурсивные функции. Связь рекурсивных функций с машинами Тьюринга. Тезис Черча.	
Тема 9	Понятие сложности вычислений; эффективные алгоритмы. Основы нечеткой логики. Элементы алгоритмической логики.	2
ИТОГО		36

Лекция 1

История логики.

История логики насчитывает около 2-х с половиной тысячелетий.

I этап 384-322 до н.э. до второй половины XIX в.-нач. XX в.: логика Аристотеля развивалась очень медленно.

II этап начало XX в. до нашего времени: на этом этапе в логике произошла научная революция, на смену традиционной логике пришла современная (математическая) логика. В основе последней лежат идеи Г. Лейбница (1646-1716) о возможности представить доказательство как математическое вычисление. Д. Буль (1815-1864) истолковал умозаключение как результат решения логических равенств, в результате чего теория умозаключения приняла вид алгебры, отличающейся от обычной алгебры лишь отсутствием численных коэффициентов и степеней. С работ Г. Фреге (1848-1925) начинается применение логики для исследования оснований математики.

В 20-е годы XX века предмет логических исследований существенно расширился, с этого периода начали складываться различные направления логики.

Многозначная логика предполагающая, что утверждения являются не только истинными или ложными, но могут иметь и другие истинностные значения («неопределенно», «возможно», «бессмысленно»). В зависимости от множества истинностных значений различают конечнозначные и бесконечнозначные логики (трехзначная логика Я. Лукасевича).

Логика эпистемическая исследует высказывании, содержащие разные теоретико-познавательные понятия: «непроверяемо», «фальцифицируемо», «полагает», «сомневается», «отвергает».

Деонтическая логика изучает логические связи нормативных высказываний.

Логика оценок занимается аксиологическими модальностями (понятия «хорошо», «плохо», «безразлично»).

Логика времени занимается временными модальностями, изучает логические связи высказываний о прошлом и будущем.

Паранепротиворечивая логика, непозволяющая выводить из противоречий все что угодно.

Логика изменения, говорящая об изменении и становлении нового.

Логика причинности, изучающая утверждения о детерминизме и причинности.

Парафальсифицирующая логика, не позволяющая отвергать положения, хотя бы одно следствие которых оказалось ложным.

Нечеткая и модальная логики.

Модальная логика

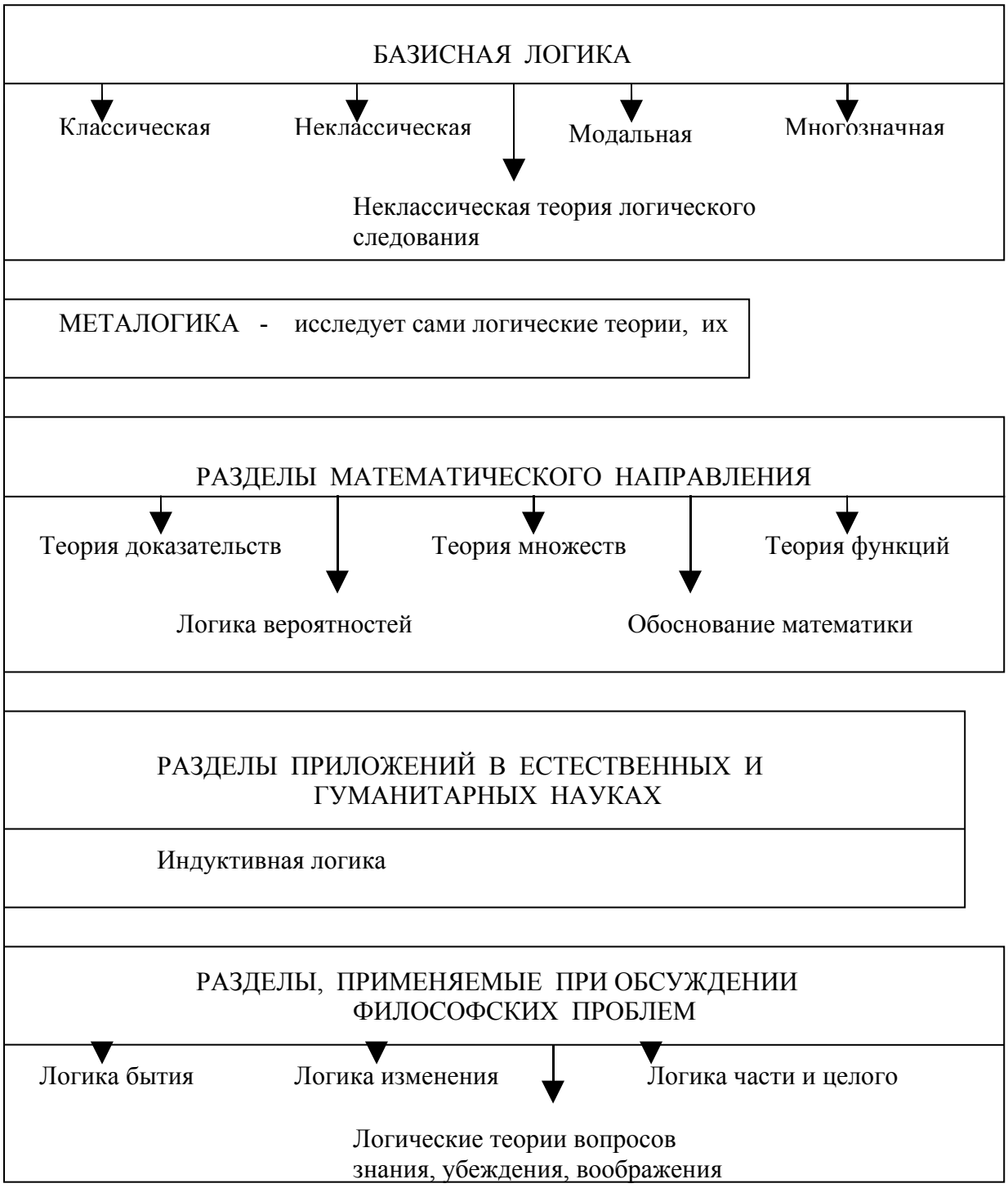
Для классической логики вещь существует или не существует, и нет других вариантов. Язык классической логики слишком беден, чтобы на нем удалось передать рассуждения не только о реальных событиях, имеющих место в действительном мире, но и о возможных событиях или необходимых событиях. В естественном языке имеется большее число логических связок, которые не сводятся к стандартным “и”, “не”, “или”, например “возможно”, “обязательно”, “всегда”, “здесь”, “завтра”. Связки такого рода называются модальностями. Изучение модальностей, начатое Аристотелем, до начала 20-го века оставалось предметом философии и филологии. К середине века модальная логика стала разделом математической логики. С конца 70-х г.г. эта область начала стремительно развиваться, появились глубокие математические результаты, в информатике, в лингвистике и в других разделах математической логики возникли многочисленные приложения модальных логик.

Модальная логика рассматривает понятия необходимости, возможности, случайности. Исследуются логические связи модальных высказываний, т.е. высказываний, включающих модальности. Теория логических модальностей изучает логическое поведение высказываний, включающих модальные понятия «логически необходимо», «логически возможно», «логически случайно».

Модальная логика и теория доказательств – приложение модальной логики к исследованию аксиоматических теорий и автоматическому построению доказательств. Направление было инициировано Геделем, который заметил, что доказуемость можно рассматривать как модальность.

Временная логика – изучение временных модальностей “будет”, “было”, “сейчас” и т.п.

Основные разделы (ветви) логики



Понятие логики

Логика (logos- слово, понятие, рассуждение разум)

Формальная логика – наука о законах и операциях правильного мышления. Согласно основному принципу логики правильность рассуждения (вывода) определяется только его логической формой или структурой, и не зависит от конкретного содержания входящих в него утверждений. Различие между формой и содержанием может быть сделанным явным с помощью особого языка, или символики, оно относительно и зависит от выбора языка.

Отличительная особенность *правильного вывода* в том, что от истинных предпосылок он всегда ведет к истинному заключению. Такой вывод позволяет из истин получать новые истины с помощью чистого рассуждения, без обращения к опыту, интуиции и т.д. *Неправильные выводы* могут от истинных посылок вести, как к истинным, так и к ложным заключениям.

Логика занимается не только связями высказываний в правильных выводах, но и

смыслом и значением выражений языка,

отношениями между терминами,

операциями определения и логического деления понятий, вероятностными и статистическими рассуждениями,

парадоксами и логическими ошибками.

Логика не перечисляет некоторые схемы правильных рассуждений, она выявляет различные типы таких схем;

устанавливает общие критерии их правильности;

выделяет исходные схемы, из которых по определенным правилам могут быть построены другие схемы данного типа;

исследует проблему взаимной совместимости логических схем.

В современной логике логические процессы изучаются путем их отображения в языках *формализованных или логических исчислений*.

Логический синтаксис – это формальное строение логических исчислений, правила образования и преобразования входящих в них выражений.

Логическая семантика – это отношения между исчислениями и содержательными областями, служащими моделями.

Лекция 2

ФОРМАЛЬНЫЕ ТЕОРИИ ИСЧИСЛЕНИЕ ВЫСКАЗЫВАНИЙ

Формальные системы – это системы операций над объектами, понимаемыми как последовательности символов (как слова в фиксированных алфавитах); сами операции также являются операциями над символами. Термин «формальный» подчеркивает, что объекты и операции над ними рассматриваются формально, без каких бы то ни было содержательных интерпретаций символов. Предполагая, что между символами не существует никаких связей и отношений, кроме тех, которые явно описаны средствами самой формальной системы.

Принципы построения формальных теорий.

Всякая точная теория определяется:1) языком, т.е. множеством высказываний, имеющих смысл с точки зрения этой теории; 2) совокупностью теорем - подмножеством языка, состоящим из высказываний, истинных в данной теории.

Формальная теория (или исчисление) строится следующим образом.

1. Определяется множество формул, или правильно построенных выражений, образующее язык теории. Это множество задается конструктивными средствами (индуктивным определением) и перечислимо. Обычно оно и разрешимо. Множество M разрешимо (рекурсивно), если существует алгоритм A, который по любому объекту a дает ответ, принадлежит a множеству M или нет.
2. Выделяется подмножество формул, называемых аксиомами теории. Множество может быть и бесконечным; во всяком случае, оно должно быть разрешимо.
3. Задаются правила вывода теории. Правило вывода $R(F_1, \ldots, F_n, G)$ –это вычислимое отношение на множестве формул.

Опр. 1 Если формулы $F_1, \ldots, F_n, G$ находятся в отношении R, то формула G называется непосредственно выводимой из $F_1, \ldots, F_n$ по правилу R. Правило $R(F_1, \ldots, F_n, G)$ записывается в виде $\frac{F_1, \ldots, F_n}{G}$

Формулы $F_1, \ldots, F_n$ называются посылками правила R, G – его следствием.

Опр.2 *Выводом* формулы B из формул $A_1, \ldots, A_n$ называется последовательность формул $F_1, \ldots, F_m$, такая, что F_m=B, а любая F_i (i=1,…, m) есть либо аксиома, либо одна из исходных формул $A_1, \ldots, A_n$, либо непосредственно выводима из формул $F_1, \ldots, F_{i-1}$ (или какого-то их подмножества) по одному из правил вывода. Если существует вывод B из $A_1, \ldots, A_n$, то говорят, что B выводима из $A_1, \ldots, A_n$ и обозначается как $A_1, \ldots, A_n|$—B. Формулы $A_1, \ldots, A_n$ называются гипотезами или посылками вывода.

Опр.3 *Доказательством* формулы B в теории T называется вывод B из пустого множества формул, т.е. вывод, в котором в качестве исходных формул используются только аксиомы. Формула B, для которой существует доказательство, называется теоремой теории T; факт доказуемости B обозначается |—B.

!Присоединение формул к гипотезам не нарушает выводимости. Поэтому если |—B, то A|—B, и если $A_1, \ldots, A_n$ |—B, то $A_1, \ldots, A_n, A_{n+1}$|—B для любых A и A_n.

Аксиоматический метод

Аксиоматический метод – способ построения научной теории, при котором какие-то положения теории избираются в качестве исходных, а все остальные ее положения выводятся из них логическим путем, посредством доказательств.

Положения, доказываемые на основе аксиом, называются теоремами.

Аксиоматический метод зародился еще в античности и приобрел большую известность благодаря «Началам» Евклида, появившимся около 330 – 320 гг. до н.э.. Д. Гильберт (1862 - 1943) рассматривал аксиоматическую теорию как *формальную теорию*, устанавливающую соотношения между ее элементами (знаками) и описывающую любые множества объектов, удовлетворяющих ей.

Опр.1 *Аксиоматические теории* формулируются как формализованные системы, содержащие точное описание логических средств вывода теорем из аксиом. *Доказательство* в такой теории представляет собой последовательность формул, каждая из которых либо является аксиомой, либо получается из предыдущих формул последовательности по одному из принятых *правил вывода*.

К аксиоматической формальной системе предъявляются требования *непротиворечивости*, *полноты*, независимости систем аксиом и т.д.

Как показал известный математик и логик К. Гедель (1906 - 1978), достаточно богатые научные теории (арифметика натуральных чисел) не допускают полной аксиоматизации. Это свидетельствует об ограниченности аксиоматического метода и невозможности полной формализации научного знания.

Теорема Геделя. (Теорема о неполноте арифметики 1931г.) *Если система Z (содержащая арифметику натуральных чисел) непротиворечива, то в ней существует такое предложение A, что ни само A, ни его отрицание не могут быть доказаны средствами Z.*

Как показывает теорема Геделя даже арифметику натуральных чисел невозможно формализовать полностью, ибо в формализованной арифметике существуют истинные предложения, которые оказываются неразрешимыми. Г.т. показывает, невозможность полной формализации человеческого знания.

Полнота теории.

Опр.2 *Полнота* (в логике и дедуктивных науках) – логико-методологическое требование, предъявляемое к аксиоматической теории и характеризующее достаточность для определенных целей ее выразительных и дедуктивных средств.

Аксиоматическая теория является полной, если все ее формулы, истинные при рассматриваемой интерпретации доказуемы. Полная система содержит все возможные теоремы, не противоречащие интерпретации.

Для уточнения семантического понимания полноты может быть выдвинуто требование, чтобы либо само предложение, либо его отрицание было теоремой, т.е. чтобы предложение было или доказуемо, или опровержимо.

В 1931 г. К. Гедель показал, что достаточно богатые аксиоматические системы (включающие арифметику натуральных чисел) в принципе не могут быть полными: в них имеются предложения, которые не могут быть ни доказаны, ни опровергнуты.

Требование полноты не является необходимым;
неполные аксиоматические системы могут представлять и теоретический, и практический интерес.

Непротиворечивость

Опр.3 *Непротиворечивость* – это свойство системы предложений некоторой теории (для аксиоматической теории – система ее аксиом), заключающееся в невыводимости из них противоречия.

Если отрицание какого-то предложения может быть доказано в теории, то о самом предложении говорится, что оно опровержимо в ней.

Непротиворечивость теории означает, что никакое предложение не может быть в ней и доказано, и опровергнуто.

Требование непротиворечивости является обязательным требованием к научной и, в частности, логической теории. Противоречивая теория заведомо несовершенна: наряду с истинными положениями она включает также ложные, в ней что-то одновременно доказывается, и опровергается.

Во многих теория имеет место закон *Дунса Скота*. В этих условиях доказуемость противоречия означает, что становится «доказуемым» все что угодно и понятие доказательства теряет смысл.

Закон Дунса Скота. Ложное высказывание влечет (имплицирует) любое высказывание.

Закон Дунса Скота выражается формулой:

$\neg p \rightarrow (p \rightarrow q)$ или эквивалентной ей в классической логике формулой:

$(p \& \neg p) \rightarrow q$.

Для простых теорий, таких как исчисление высказываний, доказательство непротиворечивости не представляет труда. В более сложных теориях оно обычно сводится к интерпретации в терминах теории множеств.

Для сложных теорий, например, арифметики и самой теории множеств, отыскание подходящей теории, которая сама была бы непротиворечивой и вместе с тем могла бы использоваться для доказательства их непротиворечивости, представляется задачей скорее всего безнадежной.

Это указывает на нетривиальность проблемы непротиворечивости.

Лекция 3

Исчисление высказываний.

В исчислении высказываний мы будем иметь дело с формулами алгебры логики. Но здесь формулы рассматриваются не как способ представления функций, а как составные высказывания, образованные из элементарных высказываний с помощью логических операций или, как говорят в логике, связок дизъюнкции, конъюнкции, отрицания, импликации. При этом особое внимание уделяется тождественно-истинным (т.-и.) высказываниям, поскольку они должны входить в любую теорию в качестве общелогических законов.

Исчисление высказываний определяется следующим образом:

1. *Алфавит* вычисления высказываний состоит из переменных высказываний (пропозиционных букв): A, B, C, …, знаков логических связок дизъюнкции, конъюнкции, отрицания, импликации ($\neg$, $\vee$, &, $\rightarrow$) и скобок.

2. *Формулы*

a) переменное высказывание есть формула;

b) если $\mathfrak{I}$ и $\mathfrak{R}$ – формулы, то $(\mathfrak{I}\vee\mathfrak{R})$, $(A\ \&\mathfrak{R})$ $(\mathfrak{I}\rightarrow\mathfrak{R})$ и $\neg\mathfrak{R}$ - формулы;

c) других формул нет.

3. *Аксиомы*. Приведем две системы аксиом.

Система аксиом I.

1. $A \rightarrow (B \rightarrow A)$
2. $(A \rightarrow B) \rightarrow ((A \rightarrow (B \rightarrow C)) \rightarrow (A \rightarrow C))$
3. $(A \,\&B) \rightarrow A$
4. $(A\& B) \rightarrow B$
5. $A\rightarrow (B \rightarrow (A \,\&B))$
6. $A \rightarrow (A \vee B)$
7. $B \rightarrow (A \vee B)$
8. $(A \rightarrow C) \rightarrow ((B \rightarrow C) \rightarrow ((A \vee B) \rightarrow C))$
9. $(A \rightarrow B) \rightarrow ((A \rightarrow \neg B) \rightarrow \neg A)$
10. $\neg\neg A \rightarrow A$

Система аксиом II.

1. $A \rightarrow (B\rightarrow A)$
2. $(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$
3. $(\neg A \rightarrow \neg B) \rightarrow ((\neg A \rightarrow B) \rightarrow A)$.

!Приведенные системы аксиом равносильны в том смысле, что порождают одно и тоже множество формул. Такое утверждение нуждается в доказательстве, которое заключается в том, что показывает выводимость всех аксиом системы II из аксиом системы I и, наоборот системы I из системы II.

Возможны и другие системы аксиом, равносильные первым двум системам.

4. *Правила вывода:*

1) правило подстановки. Если $\mathfrak{I}$ – выводимая формула, содержащая букву A (обозначим $\mathfrak{I}$(A)), то выводима формула $\mathfrak{I}(\mathfrak{R})$, получающаяся из $\mathfrak{I}$ заменой всех вхождений A на произвольную формулу;

2) правило заключения (Modus Ponens). Если $\mathfrak{I}$ и $\mathfrak{I}\rightarrow \mathfrak{R}$ – выводимые формулы, то $\mathfrak{R}$ выводима.

В этом описании исчисления высказываний аксиомы являются формулами исчисления; формулы же, использованные в правилах вывода, это «метаформулы», или *схемы формул*. Схема формул, например $\mathfrak{I}\rightarrow\mathfrak{R}$, обозначает множество всех тех формул исчисления, которые получаются, если ее метапеременные заменить формулами исчисления.

Примеры вывода в исчислении высказываний.

Лекция 4

Основные метатеоремы исчисления высказываний.

Теорема 1 (дедукции). Если Г (множество формул), $\mathfrak{I}$ |–$\mathfrak{R}$, то Г |– $\mathfrak{I}\rightarrow \mathfrak{R}$ ($\mathfrak{I}$,$\mathfrak{R}$ - формулы).

Будем исходить из системы аксиом II и рассматривать их как схемы аксиом (не пользоваться правилом подстановки).

Следствие 1 (из теоремы дедукции) (правило силлогизма)

A⊂B, B⊂C |– A⊂C.

Построим вывод правила силлогизма:

1. A⊂B - гипотеза;
2. B⊂C - гипотеза;
3. A - гипотеза;
4. B применяем правило *т.р*. 1. 3.
5. C - применяем правило *т.р*. 2. 4.
6. Тогда A⊂B, B⊂C, A|– C. По теореме о дедукции A⊂B, B⊂C |– A⊂C.

Следствие 2 *(из теоремы дедукции)* A ⊂ (B⊂C), B |– A⊂C.

В результате двукратного применения правила *т.р*. получим A, A ⊃ (B ⊃C), B |– C. Отсюда по теореме дедукции A ⊃ (B⊃C), B |– A⊃C.

Для любых формул A и B в исчислении высказываний верны утверждения:

Утверждение 1. |– ⌝⌝A ⊃ A.
Утверждение 2. |– A ⊃⌝⌝A.
Утверждение 3. |– ⌝A ⊃ (A ⊃ B).
Утверждение 4. |– (⌝B ⊃⌝A)⊃ (A ⊃ B).
Утверждение 5. |– (A ⊃ B)⊃ (⌝B ⊃⌝A).
Утверждение 6. |– A ⊃ (⌝B ⊃ ⌝(A ⊃ B)).
Утверждение 7. |– (A ⊃ B) ⊃ ((⌝A ⊃ B) ⊃ B).

Применение теоремы дедукции.

Пример1. Покажем, что аксиома II3 выводима из системы аксиом I.

1. Подставим в I9 ⌝A вместо A. Получим: (⌝A ⊃ B) ⊃ ((⌝A ⊃ ⌝B) ⊃ ⌝⌝A).
2. Двойное применение правила *m.p*. к шагу 1 дает:
⌝A ⊃ B, ⌝A ⊃ ⌝B |– ⌝⌝A.
3. Так как из аксиомы I 10 следует по правилу *m.p*., что ⌝⌝A |– A, то по транзитивности выводимости получим ⌝A ⊃ B, ⌝A ⊃ ⌝B |– A.
4. Переставим гипотезы в полученной выводимости (их порядок неважен, как видно из определения выводимости): ⌝A ⊃ ⌝B, ⌝A ⊃ B |– A.
5. Применив 2 раза к шагу 4 теорему дедукции, получим аксиому II3:
(⌝A ⊃ ⌝B) ⊃ ((⌝A ⊃ B) ⊃ A).

Пример 2. Очень распространенным методом математических доказательств является метод доказательства от противного: предположим, что A верно, и покажем, что 1) из A выводится B, 2) что из A выводится ⌝B, что невозможно и, следовательно, A неверно, т.е. верно ⌝A.

Этот метод формулируется как правило: «если Г, A |– B и Г, A |– ⌝B, то Г|– ⌝A». Докажем, что это правило в исчислении высказываний выполняется.

По теореме дедукции, если Г, A |– B и Г, A |– ⌝B, то Г|– A⊃B и Г|– A⊃⌝B. Из этих импликаций и аксиомы I9 двойным применением правила заключения {*m.p*.} получаем Г |–⌝A. Доказанное правило называется также *правилом введения отрицания*.

Пример 3. Докажем закон исключенного третьего |– A ∨⌝A.

1. ⌝(A ∨⌝A), A |– A ∨⌝A (аксиома I6 при B = ⌝A и *m.p*.).
2. ⌝(A ∨⌝A), A |– ⌝(A ∨⌝A) (очевидно)
3. Применяя к шагам 1 и 2 только что доказанное правило введения отрицания, получаем: ⌝(A ∨⌝A) |– ⌝A.
4. Аналогично доказывается ⌝(A ∨⌝A) |– ⌝⌝A.
5. Применяя к шагам 3 и 4 введение отрицания, получаем:
|– ⌝⌝(A ∨⌝A).
6. С помощью аксиомы I 10 и правила *m.p*. снимаем двойное отрицание в шаге 5 и получаем |– A ∨⌝A.

Лекция 5

Исчисление высказываний и алгебра логики.

Формула F исчисления высказываний содержательно интерпретируется как составное высказывание, истинность которого зависит от истинности входящих в него элементарных высказываний. Эта зависимость в точности соответствует зависимости значения логической функции, представляемой формулой F, от значений переменных этой функции. Если задана формула $F(A_1, \ldots, A_n)$ и распределение истинностей входящих в нее элементарных высказываний $A_1, \ldots, A_n$, то для выяснения истинности ее нужно вычислить как логическую функцию на наборе $(\sigma_1, \ldots, \sigma_n)$.

$$A_i^{\sigma} = \begin{cases} A_i, \text{ если } \sigma_i=1, \text{ т.е. если } A_i \text{ истинно;} \\ \neg A_i, \text{ если } \sigma_i=0, \text{ т.е. если } A_i \text{ ложно.} \end{cases}$$

Если $F(\sigma_1, \ldots, \sigma_n)=1$, то высказывание F истинно при данном распределении истинностей $A_1, \ldots, A_n$, если $F(\sigma_1, \ldots, \sigma_n)=0$, то высказывание F ложно.

Возникает вопрос, как связано такое содержательное, "истинностное" истолкование формул с их выводимостью в исчислении высказываний?

Теорема 12. Пусть формула $\Re(A_1, \ldots, A_n)$ определяет логическую функцию f от n переменных. Тогда, если $f(\sigma_1, \ldots, \sigma_n)=\sigma$, то в вычислении высказываний $A_1^{\sigma 1}, \ldots, A_n^{\sigma n} \vdash \Re^{\sigma}$.

Теорема 13. Всякая теорема исчисления высказываний является тождественно-истинным высказыванием.

Теорема 14. Всякая тождественно-истинная формула является теоремой исчисления высказываний.

Таким образом, исчисление высказываний действительно выполняет задачу порождения общелогических законов – тождественно истинных высказываний.

Лекция 6

Полнота и непротиворечивость исчисления высказываний.

Исчисление высказываний построено как аксиоматическая теория.

! Множество теорем исчисления высказываний совпадает с множеством т-и (тождественно-истинная) формул логики высказываний.

Теорема 1. Всякая выводимая (из пустой системы гипотез) формула исчисления высказываний т-и (тождественно-истинная).
Доказательство.

Теорема 2. Любая т-и (тождественно-истинная) формула выводима в исчислении высказываний, т.е. является теоремой.

Теорема о непротиворечивости

Исчисление высказываний непротиворечиво.

Действительно согласно теореме 1 всякая выводимая формула т.-и. Отрицание этой формулы не является т.-и. формулой. Следовательно, ни для какой формулы А невозможно, чтобы одновременно |– А и |– ⌉А.

Наряду с полнотой аксиоматической теории в широком смысле рассматривают ее полноту и в узком смысле, если добавление любой невыводимой формулы в качестве схемы аксиом приводит к противоречивой теории.

Теорема о полноте: Исчисление высказываний полно в узком смысле.

F-произвольная невыводимая формула (согласно теореме о непротиворечивости в качестве F можно взять любую не тождественно-истинную формулу), X_1, …X_n – список ее переменных, а $<\sigma_1, \ldots, \sigma_n>$ - распределение истинностей , на котором формула F принимает ложное значение.

Расширенная теория оказывается противоречивой.

Во всякой формальной теории возникает вопрос о независимости ее аксиом, т.е. вопрос о том, можно ли какую-нибудь аксиому вывести из остальных, применяя правила вывода данной системы.

! Система аксиом II исчисления высказываний независима.

Лекция 7

ИСЧИСЛЕНИЕ ПРЕДИКАТОВ И ТЕОРИИ ПЕРВОГО ПОРЯДКА

Аксиомы и правила вывода

1. А л ф а в и т исчисления предикатов состоит из предметных переменных x_1, x_2 …, предметных констант a_1, a_2 …, предикатных букв P^1_1, P^1_2, …, P^1_k… и функциональных букв f^1_1, f^1_2, …, f^1_k…, а также знаков логических связок ⌉, ∨, &, → , кванторов общности и существования (∀, ∃) и скобок ().

Опр.1 *n-местный предикат* – это функция $P(x_1, \ldots, x_n)$ от n переменных, принимающих значения из некоторых заданных предметных областей, так, что x_1 из M_1, x_2 из M_2 …, x_n из M_n, а функция P принимает логические

значения – «истинно» или «ложно». Предикат $P(x_1, \ldots, x_n)$ является функцией типа $P: M_1 \times M_2 \times \ldots \times M_n \to B$, где множества M_1, M_2 …, M_n называются предметными областями предиката; $x_1, \ldots, x_n$ – предметными переменными предиката; B – бинарное множество {И, Л} или {0, 1}.

Квантор собственные связки логики предикатов.

Верхние индексы предикатных и функциональных букв указывают число аргументов, их нижние индексы служат для обычной нумерации букв.

Переменные высказывания в исчислении предикатов вводятся либо непосредственно как пропозиционные буквы A_1, A_2, …, либо как 0-местные предикаты P^0_1, P^0_2, …, т.е. как предикаты без переменных.

2. Ф о р м у л ы. Понятие формулы определяется в два этапа.

1) Термы:

a) предметные переменные и константы являются термами;

b) если f^n – функциональная буква, а $t_1, \ldots, t_n$ – термы, то $f^n(t_1, \ldots, t_n)$ - терм.

2) Формулы:

a) если P^n – предикатная буква, а $t_1, \ldots, t_n$ – термы, то $P^n(t_1, \ldots, t_n)$ – формула; все вхождения предметных переменных в формулу вида называют свободными переменными;

b) если F_1, F_2 – формулы, то формулами являются $\neg F_1$, $(F_1 \to F_2)$, $(F_1 \& F_2)$, $(F_1 \vee F_2)$; все вхождения переменных, свободные в F_1, F_2, являются свободными и в указанных четырех видах формул;

c) если $F(x)$ – формула, содержащая свободные вхождения переменной x, то $\forall x\, F(x)$ и $\exists x\, F(x)$ - формулы; в этих формулах все вхождения переменной x называются связанными; вхождения остальных переменных в F остаются свободными.

3. А к с и о м ы и с ч и с л е н и я п р е д и к а т о в делятся на две группы:

1) аксиомы исчисления высказываний (любая из систем I или II);

2) предикатные аксиомы:

P1. $\forall x\, F(x) \to F(y)$

P2. $F(y) \to \exists x\, F(x)$

4. П р а в и л а в ы в о д а

1) правило заключения (Modus Ponens)

3) правило обобщения ($\forall$- введения):

$$\frac{F \to G(x)}{F \to \forall x G(x)}$$

4) правило $\exists$-введения:

$\underline{G(x)\to F}$
$\exists x G(x)\to F$

Нарушение этих требований могут привести к ложным выводам из истинных высказываний.

Пусть, например, $P(x)$ предикат «х – делится на 6», $Q(x)$ предикат «х – делится на 3». Высказывание $P(x) \to Q(x)$, очевидно, истинно для любого х, однако применение к нему правила обобщения дает высказывание $P(x)\to\forall x\, Q(x)$, не являющееся всегда истинным.

Если же к $P(x) \to Q(x)$ применить правило $\exists$-введения, получим $\exists x\, P(x)\to Q(x)$, из которого путем применения правила обобщения получим высказывание $\exists x\, P(x) \to \forall x\, Q(x)$, ложное на множестве натуральных чисел.

Возможны и другие системы аксиом и правил. Принцип минимизации числа логических операторов в исчислении высказываний оставляет лишь связки $\neg$ и $\to$ (что отражено в системе аксиом II). В исчислении предикатов этот принцип выражается в том, что квантор $\exists x$ не считается самостоятельным символом, а рассматривается как сокращение выражения $\neg\forall x\neg$.

Примеры вывода в исчислении предикатов.

Правило переименования свободных переменных
Правило переименования связных переменных

Пример 1.

Правило переименования свободных переменных.

В исчислении предикатов из выводимости формулы $F(x)$, содержащей свободные вхождения x, ни одно из которых не находится в области действия квантора по y, следует выводимость $F(y)$.

1. $\vdash F(x)$ (по условию)

2. $F(x)\to(G\to F(x))$ (акс.II 1; в качестве G выбираем любую доказуемую формулу, не содержащую свободных вхождений x: ее доказуемость понадобится на шаге 5, а ограничение на x на шаге 4)

3. $G\to F(x)$ (m.p. к шагам 1 и 2)

4. $G\to \forall x F(x)$ (правило обобщения к шагу 3)

5. $\forall x F(x)$ (m.p. к G и шагу 4)

6. $F(y)$ (m.p. к шагу 5 и аксиоме P1)

Пример 2.

Правило переименования связанных переменных.

В исчислении предикатов из выводимости формулы $\forall x F(x)$, следует выводимость $\forall y F(y)$ при условии, что $F(x)$ не содержит свободных вхождений y и содержит свободные вхождения x, ни одно из которых не входит в область действия квантора по y.

1. $\vdash \forall x\, F(x)$ (по условию)

2. $\forall x F(x) \to F(y)$ (акс. P1)

3. $\forall x F(x) \to \forall y F(y)$ (правило обобщения к шагу 2)

4. $\forall y F(y)$ (m.p. к шагам 1 и 3).

Лекция 8

Выводимость и истинность. Эквивалентные преобразования в исчислении предикатов.

Теорема 5. Всякая доказуемая формула исчисления предикатов тождественно-истинна (общезначима).

Непосредственно проверяется общезначимость аксиом и показывается, что правила вывода сохраняют общезначимость, т.е. их применение к общезначимым формулам дает общезначимые формулы.

Справедлива и обратная теорема.

Теорема 6. Всякая общезначимая предикатная формула доказуема в исчислении предикатов.

! Всякому эквивалентному соотношению F=G в булевой алгебре соответствует доказуемая эквивалентность $\vdash\!\!\dashv F \sim G$ в исчислении высказываний. Из теорем 5, 6 следует, что между соотношениями содержательной логики предикатов и формальными эквивалентностями в исчислении предикатов имеется аналогичное соответствие ($F \sim G$ рассматривается как сокращение $(F \to G)\&(G \to F)$). На нем имеет смысл остановиться подробнее.

Доказательства общезначимости в логике предикатов существенно сложнее, чем в логике высказываний, и поэтому формальный вывод эквивалентностей становится важным способом их получения.

Теорема 7. Пусть F(A) – формула, в которой выделено вхождение формулы A; F(B) – формула, полученная из F(A) заменой вхождения A формулой B. Тогда если ⊢—A~B, то ⊢—F(A)~F(B).

Эта теорема формулируется как правило для исчисления предикатов, аналогичное правилу замены эквивалентных подформул в алгебрах. Благодаря этому правилу можно получать доказуемые эквивалентности в исчислении, не строя непосредственного вывода.

При доказательстве теоремы используются следующие производные правила:

1) если ⊢—A~B, то ⊢— A→C~B→C и C→ A ~C→B;
2) если ⊢—A~B, то ⊢— A∨C~B∨C и ⊢—C∨ A ~C∨B;
3) если ⊢—A~B, то ⊢— A&C~B&C и ⊢—C& A ~C&B;
4) если ⊢—A~B, то ⊢— ¬A~¬B;
5) если ⊢— F(x)~G(x), то ∀ x F(x)~ ∀ x G(x);
6) если ⊢— F(x)~G(x), то ∃ x F(x)~ ∃ x G(x).

Первые четыре правила проверяются таблицами истинности.

С помощью этих шести правил теорема 7 доказывается индукцией по построению формулы F(A): показывается, что если A~B, то эта эквивалентность сохраняется на всех шагах построения F(A) из A и F(B) из B. Продемонстрируем доказательство на примере.

Пример1.

Пусть F(A)= ∀y ($P_1(y)$∨¬∃x $P_2(x, y)$), A= ¬∃x $P_2(x, y)$. В качестве эквивалентности A~B возьмем эквивалентность ¬∃x $P_2(x, y)$ ~ ∀x ¬$P_2(x, y)$, верную в логике предикатов и, следовательно, в силу теоремы 7 доказуемую в исчислении предикатов.

1. ¬∃x $P_2(x, y)$ ~ ∀x ¬$P_2(x, y)$ (исходная эквивалентность)
2. ($P_1(y)$ ∨ ¬∃x $P_2(x, y)$) ~ ($P_1(y)$ ∨∀x ¬$P_2(x, y)$) (правило 2)
3. ∀ y ($P_1(y)$ ∨ ¬∃x $P_2(x, y)$) ~ ∀ y ($P_1(y)$ ∨∀x ¬$P_2(x, y)$) (правило 5)

Формула из шага 3 и есть искомая эквивалентность F(A)~ F(B).

Эквивалентности, выводимые в исчислении предикатов

A&∀x F(x) ~∀x(A &F(x)); (1)
A∨∃x F(x) ~∃x(A ∨F(x)); (2)
A&∃x F(x) ~∃x(A &F(x)); (3)
A∨∀x F(x) ~∀x(A ∨F(x)); (4)
A→∀x F(x) ~∀x(A →F(x)); (5)
A→∃x F(x) ~∃x(A →F(x)); (6)
∀x F(x) →B ~∃x(F(x) →B); (7)
∃x F(x) →B ~∀x(F(x) →B). (8)

Эквивалентности позволяют выносить кванторы вперед. Используя при этом соотношения, полученные в алгебре логики предикатов ($\neg(\exists x\, P(x)) \sim \forall x(\neg P(x))$, $\neg(\forall x\, P(x)) \sim \exists x(\neg P(x))$), позволяющие заменять один квантор другим и «спускать» отрицание внутрь области действия квантора, а также правила переименования переменных, кванторы можно вынести вперед для любой формулы.

Префиксная нормальная форма

Опр.1 Формула, имеющая вид $Q_1x_1Q_2\, x_2 \ldots\, Q_nx_nF$, где $Q_1, Q_2 \ldots Q_n$ – кванторы, а F – формула, не имеющая кванторов (и являющаяся областью действия всех n кванторов), называется *предваренной формой*, или формулой в предваренной форме.

В исчислении предикатов для любой формулы F существует эквивалентная ей предваренная форма F^*, т.е. $\vdash\!\dashv F \sim F^*$.

Этапы приведения к ПНФ:

1.
2.
3.

Пример 2.

Теории первого порядка (прикладные исчисления предикатов).

Построенное ранее исчисление предикатов называется исчислением предикатов первого порядка. В исчислениях второго порядка возможны кванторы по предикатам, т.е. выражения вида $\forall P(P(x))$. Приложения таких исчислений встречаются гораздо реже, нами рассматриваться не будут.

Исчисление предикатов, не содержащее функциональных букв и предметных констант, называется чистым исчислением предикатов. До сих пор рассматривалось именно чистое исчисление предикатов, хотя язык исчисления (формулы) был определен с учетом его использования в прикладных исчислениях.

Прикладные исчисления (теории первого порядка) характеризуются тем, что в них к чисто логическим аксиомам добавляются собственные аксиомы, в которых участвуют конкретные (индивидуальные) предикатные буквы и константы. Примеры индивидуальных предикатных букв – предикаты =, <, функциональных букв – знаки арифметических операций, предметных констант – натуральные числа, единица в теории групп, пустое множество в теории множеств.

Важная особенность прикладных исчислений заключается в том, что в схемах аксиом P1 и P2 участвуют уже не предметные переменные, а произвольные термы. Более точно эти схемы аксиом принимают следующий вид:

P1*. $\forall x\, \mathrm{F}(x) \to \mathrm{F}(t)$

P2*. $\mathrm{F}(t) \to \exists x\, \mathrm{F}(x)$,

где $\mathrm{F}(t)$ – результат подстановки терма t в $\mathrm{F}(x)$ вместо всех свободных вхождений x, причем все переменные t должны быть свободными в $\mathrm{F}(t)$.

1. Исчисления с равенством.

Большинство прикладных исчислений содержит предикат равенства = и определяющие его аксиомы.

Аксиомы для =:

E1. $\forall x\, (x = x)$; (конкретная аксиома)

E2. $(x = y) \to (\mathrm{F}(x, x) \to \mathrm{F}(x, y))$ (схема аксиом),

где $\mathrm{F}(x, y)$ получается из $\mathrm{F}(x, x)$ заменой некоторых (необязательно всех) вхождений x на y при условии, что y в этих вхождениях также остается свободным. Всякая теория, в которой E1 и E2 являются теоремами или аксиомами, называется *теорией* (или исчислением) *с равенством*. Из E1 и E2 выводимы основные свойства равенства – рефлексивность, симметричность и транзитивность.

Теорема 8. В любой теории с равенством:

1) $\vdash t = t$ для любого терма t;

2) $\vdash x = y \to y = x$;

3) $\vdash x = y \to (y = z \to x = z)$.

2. Теория частичного строгого порядка

Пример прикладного исчисления – теория частичного строгого порядка, содержащая две конкретные аксиомы для предиката “<”:

NE1. $\forall x \neg (x < x)$;

NE2. $\forall x \forall y \forall z\, (x < y \to (y < z \to x < z))$.

Два сходных утверждения о транзитивности даются в разной форме: теорема 8, без кванторов, в открытой форме, а в NE2 – с кванторами, в замкнутой форме. Эти два способа задания аксиом равносильны: от первого ко второму переходим по правилу обобщений, а от второго к первому – по аксиоме P1* и правилу заключения.

Если к аксиомам NE1 и NE2 добавить аксиому с предметной константой *a*

NE3. $\forall x \neg(x < a)$, то получим теорию частичного порядка с минимальным элементом *a*.

3. Формальная арифметика.

Наиболее изученной формальной теорией, которая играет фундаментальную роль в основаниях математики, является формальная арифметика. Перечислим ее схемы аксиом:

A1. $F(0)\&\forall x(F(x) \to F(x')) \to F(x)$ (принцип индукции)

A2. $t_1' = t_2' \to t_1 = t_2$;

A3. $\neg(t_1' = 0)$;

A4. $t_1 = t_2 \to (t_1 = t_3 \to t_2 = t_3)$;

A5. $t_1 = t_2 \to t_1' = t_2'$;

A6. $t_1 + 0 = t_1$;

A7. $t_1 + t_2' = (t_1 + t_2)'$;

A8. $t_1 \cdot 0 = 0$;

A9. $t_1 \cdot t_2' = t_1 \cdot t_2 + t_1$.

В этих аксиомах использованы три функциональных символа +, ·, ', один индивидуальный предикат (предикатная буква) = и одна предметная константа 0. Они придают схемам аксиом A2 – A9 конкретный вид: все предикатные и функциональные буквы в них зафиксированы, и единственный способ их варьировать – это подставлять различные термы вместо метапеременных t_1 и t_2.

Схемы A6 – A9 – это просто предикат равенства, в который подставлены термы определенного вида. Схема A1 имеет вид: $F(x)$ – метаобозначение употребляемое в том же смысле, в каком оно использовалось в чистом исчислении предикатов. Схемы A2 – A9 можно заменить конкретными аксиомами (т.е. формулами арифметики), A2 – A9 можно получить с помощью правила обобщения и схемы аксиом P1.

Лекция 9

МЕТАТЕОРИЯ ЛОГИЧЕСКИХ ИСЧИСЛЕНИЙ

Под метатеорией логических исследований понимается изучение их общих свойств и соответствия этих свойств целям, ради которых исчисления создавались. Некоторые задачи такого рода (связь между доказуемостью и истинностью) уже рассматривались. Здесь они будут систематизированы и изучены более подробно.

Интерпретация и модели.

Ценность всякой формальной теории, в конечном счете, определяется ее способностью описывать какие-то объекты и связи между ними. Поэтому один из первых для любой теории вопросов – это вопрос о том, для описания каких объектов пригодна данная теория. Конечно, если ставить его как общую проблему соответствия научных знаний о мире самому миру, то он не может обсуждаться средствами лишь математики (поскольку математика в отличие от естественных наук имеет дело не непосредственно с миром, а с формальными описаниями, его различных фрагментов) и становится фундаментальной проблемой философии и методологии науки. Речь пойдет о более простом случае, когда множество объектов, для которого строится формальная теория, само по себе представляет достаточно строго описанный предмет исследований. Более точно проблема адекватности формальной теории и описываемых ею объектов будет рассматриваться как математическая задача о соответствии между содержательно построенной теорией, рассматриваемой как множество объектов с операциями и отношениями на нем (т. е. как алгебраическая система), и множеством высказываний об этой теории, построенном как формальное исчисление. При такой постановке задачи интуитивно ясное понятие интерпретации приобретает точный математический смысл.

Интерпретация формальной теории состоит из множества *M* и однозначного отображения, которое каждой предикатной букве P_i^n ставит в соответствие *n*-местное отношение на *M* (интерпретирует ее как отношение на *M),* каждой функциональной букве f_j^n – *n*-местную операцию на *M,* каждой предметной константе – элемент *M.* Постоянные термы исчисления (не содержащие предметных переменных) при таком определении также отобразятся в элементы *M.* Таким образом, множество *M* (называемое областью интерпретации) рассматривается как основное множество алгебраической системы.

Всякая замкнутая, т. е. не содержащая свободных переменных, формула теории представляет собой высказывание об элементах, отношениях и функциях *M,* которое может быть истинным или ложным. Значения истинности составных формул вычисляются в соответствии с входящими в них логическими операциями. Открытая формула соответствует некоторому отношению на *M,* при подстановке предметных констант она превращается в высказывание о том, что между элементами *M,* соответствующими подставленным константам выполняется данное отношение. Открытая формула называется *выполнимой* в данной интерпретации, если существует такая подстановка предметных констант, при которой она превращается в истинное высказывание. Формула называется *истинной в данной интерпретации,* если она выполняется (т. е. превращается в истинное высказывание) при любой подстановке констант. Формула называется ложной в данной интерпретации, если она невыполнима.

Интерпретация (а иногда область интерпретации M) называется *моделью* для множества формул Г, если любая формула Г истинна в данной интерпретации. Интерпретация называется *моделью теории T,* если она является моделью множества всех теорем теории *T,* т. е. если всякая формула, доказуемая в *T,* истинна в данной интерпретации.

Если в T доказуема некоторая открытая формула F (которая, строго говоря, высказыванием не является), то в модели теории T должны быть истинными все высказывания, получающиеся из F всеми возможными подстановками констант на место свободных переменных формулы *F,* и, следовательно, должно быть истинно высказывание $\forall x_1 \ldots \forall x_n F$, где $x_1, \ldots, x_n$ – свободные переменные формулы F.

Пример. В теории строгого частичного порядка (с аксиомами NE1 и NE2) моделью будет любая интерпретация, при которой предикатная буква < интерпретируется отношением «быть меньше». Однако почти столь же очевидно из аксиом NE1 и NE2 (хотя и выглядит парадоксально), что моделью этой теории будет и интерпретация, при которой символ "<" интерпретируется как отношение «быть больше».

Пример иллюстрирует существо формального подхода: в символы исчисления (даже самые привычные) не вкладывается никакого смысла, пока не введена их явная интерпретация. Но и введенная интерпретация, вообще говоря, не относится к числу средств самого исчисления: она позволяет осмыслить формулы исчисления, но не участвует в формальном выводе теорем.

О формальных свойствах самого исчисления, его формул и формальных преобразований над ними принято говорить как о *синтаксисе исчисления;* свойства исчисления, описанные в терминах его интерпретаций, – это *семантика исчисления.*

Непротиворечивость.

Формула называется *общезначимой,* если она истинна в любой интерпретации, и *противоречивой,* если она ложна в любой интерпретации, т. е. если ее отрицание общезначимо. Для любого множества общезначимых формул и, в частности, для чистого исчисления предикатов (по теореме 6.5) любая алгебраическая система и вообще любое множество является моделью. Напротив, для любого множества формул, содержащего хотя бы одну противоречивую формулу, моделей не существует.

Аксиома E1 теории с равенством не является общезначимой: существуют интерпретации (в смысле, указанном в начале), в которых она ложна. Примером такой интерпретации может служить всякое отображение, которое предикатной букве = ставит в соответствие отношение <. (Здесь символы = и < используются на разных уровнях: символ = – как формальная предикатная буква, не имеющая смысла до ее

интерпретации, а символ < – как символ отношения на множестве, имеющий всем известный смысл «быть меньше».) Собственные аксиомы прикладных исчислений вообще не общезначимы – иначе по теореме об общезначимой формуле в исчислении предикатов они были бы доказуемы в чистом исчислении
предикатов.

Введенное ранее определение противоречивой формулы является семантическим, т.е. связывающим непротиворечивость с истинностью. Исходя из него, можно сформулировать понятие семантически непротиворечивой теории: теория *семантически непротиворечива,* если ни одна из ее теорем не является противоречивой, т.е. ложной в любой интерпретации. Исчисление высказываний и исчисление предикатов семантически непротиворечивы в силу теорем (о теореме исчисления высказываний и о доказуемой формуле исчисления предикатов): все их теоремы общезначимы и, следовательно, непротиворечивы.

С помощью введенных понятий ответ на общий вопрос, поставленный в начале лекции, формулируется так: теория *T* пригодна для описания тех множеств, которые являются ее моделями; модель для теории *T* существует тогда и только тогда, когда *T* семантически непротиворечива.

Чисто логические теории – *исчисление высказываний* и *исчисление предикатов* – в силу теорем (о теореме исчисления высказываний и о доказуемой формуле исчисления предикатов) пригодны для описания любых множеств, что соответствует общенаучному принципу универсальности законов логики. Лейбниц формулировал его как выполнимость логических законов «во всех мыслимых мирах». Такой критерий пригодности теорий по существу известен уже давно; отыскание модели для теории до возникновения оснований математики было единственным общепризнанным методом доказательства законности теории. Одно из важных достижений оснований математики заключается в том, что аналог этого критерия сформулирован в терминах самих формальных теорий, без привлечения семантических понятий. Таким аналогом является понятие формальной, или дедуктивной непротиворечивости.

Теория *T* называется *формально непротиворечивой,* если не существует формулы *F*, такой, что *F* и $\neg F$ являются теоремами теории *T*, т. е. в *T* не выводимы одновременно формула и ее отрицание. Аналогичное определение можно сформулировать и для произвольного множества формул.

Для любой теории, содержащей исчисление высказываний, из определения непосредственно следует, что если она формально противоречива, то в ней доказуема тождественно-ложная формула и, следовательно, она семантически противоречива.

Обратное утверждение (которое также оказывается верным), если понимать его конструктивно, гораздо глубже. Смысл его в том, что по всякому формально непротиворечивому множеству формул можно построить его модель. Доказательство этого факта довольно сложно и заключается в изложении метода такого построения. Вместе оба утверждения образуют важную метатеорему.

Теорема 1. Множество формул формально непротиворечиво, если и только если оно семантически непротиворечиво (т.е. имеет модель).□

Таким образом, понятие формальной непротиворечивости оказывается эквивалентным более привычному в математике понятию семантической непротиворечивости; однако оно сформулировано в синтаксических терминах и с конструктивной точки зрения более надежно.

Неприятие современниками неевклидовых геометрий Лобачевского– Бойаи объясняется именно тем, что законность этих теорий обосновывалась отсутствием в них противоречий – аргументом, по существу совпадающим с современным понятием формальной непротиворечивости. Геометрические модели для этих теорий, доказывающие их семантическую непротиворечивость, были найдены позднее.

Полнота, разрешимость, аксиоматизируемость.

Пусть имеется, с одной стороны, содержательная теория *S,* сформулированная в семантических терминах, т.е. совокупность истинных высказываний о некоторой алгебраической системе *M;* с другой стороны — формальная теория *T,* т. е. множество выражений, выводимых из аксиом теории *T* с помощью правил вывода теории *T.* В каких случаях можно утверждать, что *T* является удовлетворительной формализацией S? Каковы признаки удовлетворительности формализации?

Очевидно, необходимым признаком является условие, чтобы S была моделью теории *T,* т. е. чтобы существовало отображение, При котором всякая теорема теории *T* отображается в истинное высказывание из *S.* Однако само по себе это условие явно недостаточно, иначе для формализации любой теории S хватило бы чистого исчисления предикатов: ведь для него любое множество является моделью. Исчерпывающей формализацией S будет служить такая теория *T,* для которой выполняется и обратное соответствие: каждое истинное высказывание теории S отображается в некоторую теорему теории *T.* Теория *T* с таким свойством называется *полной* относительно *S,* а иногда – *адекватной S.*

Из теорем (исчисления высказываний) следует, что исчисление высказываний полно относительно алгебры высказываний, а теоремы (исчисления предикатов) образуют известную *теорему Геделя о полноте исчисления предикатов* относительно логики предикатов, т. е. множества всех общезначимых предикатных формул.

Если для семантической (содержательной) теории S удается построить непротиворечивую и полную формальную теорию *T*, то S естественно назвать *аксиоматизируемой,* или *формализуемой теорией*.
Из предыдущих результатов следует, что логика высказываний и логика предикатов аксиоматизируемы с помощью соответствующих исчислений.

Еще одна важная характеристика формальной теории – это ее *разрешимость.* Формальная теория *T* называется *разрешимой,* если существует алгоритм, который для любой формулы языка определяет, является она теоремой в *T* или нет, иначе говоря, если предикат «быть теоремой в теории *T*» общерекурсивен.

Теорема 2. Исчисление высказываний разрешимо.

Разрешающий алгоритм для формулы *F* исчисления высказываний заключается в вычислении значений *F* на всех наборах значений ее переменных. Ввиду полноты исчисления высказываний *F* является его теоремой, если и только если она истинна на всех наборах.

Теорема 3 (теорема Черча). Исчисление предикатов неразрешимо.

Несмотря на полноту исчисления предикатов, разрешающий алгоритм, связанный с вычислением значений истинности предикатных формул, построить не удается из-за бесконечности предметной области, которая приводит в общем случае к бесконечным таблицам истинности. *Идея доказательства* теоремы Черча (которое здесь не приводится) состоит в том, чтобы в чистом исчислении предикатов описать предикат $Q(i, a, x)$: «машина Тьюринга с номером *i*, будучи применена к исходным данным *a*, закончит вычисление в момент *x*». Предикат $\exists x Q(i, a, x)$ – это предикат остановки, а $\exists x \mathrm{Q}(a, a, x)$ – предикат самоприменимости; оба предиката неразрешимы.

Отметим, что важный для приложений фрагмент исчисления предикатов разрешим: исчисление одноместных предикатов (т.е. исчисление, допускающее в формулах только одноместные предикатные буквы) разрешимо.

Еще тяжелее обстоит дело с формальной арифметикой, система аксиом которой приведена в конце лекции №__.

Теорема 4 (первая теорема Геделя о неполноте в форме Клини). Любая формальная теория *T,* содержащая формальную арифметику, неполна: в ней существует (и может быть эффективно построена) замкнутая формула *F,* такая, что $\neg F$ истинно, но ни *F,* ни $\neg \mathrm{F}$ не выводимы в *T*.

Теорема 5 (вторая теорема Геделя о неполноте). Для любой непротиворечивой формальной теории *T,* содержащей формальную арифметику, формула, выражающая непротиворечивость *T*, недоказуема в *T*.

Арифметика и теория чисел оказываются неаксиоматизируемыми теориями. В терминах теории алгоритмов сформулированные ранее результаты можно резюмировать как:

1) множество всех истинных высказываний логики высказываний перечислимо и разрешимо;

2) множество всех истинных высказываний логики предикатов перечислимо (ввиду его полноты), но неразрешимо;

3) множество всех истинных высказываний арифметики неперечислимо: если бы для него существовала перечисляющая процедура (и, следовательно, соответствующая машина Тьюринга), то по ней можно было бы построить полную формальную теорию, рассматривая правила перехода машины от одной конфигурации к другой как правила вывода, процесс вычисления – как вывод, а результаты вычисления – как теоремы.

Две знаменитые теоремы Геделя имеют важное методологическое значение. Из *первой теоремы* следует, что для достаточно богатых математических теорий не существует адекватных формализаций. Правда, любую неполную теорию *T* можно расширить, добавив, например, к ней в качестве новой аксиомы истинную, но не выводимую в *T* формулу. Однако по первой теореме Геделя новая теория *T* также будет неполна. *Вторая теорема* может быть истолкована как невозможность исследования метасвойств теории средствами самой формальной теории (опять невозможность самоприменимости!); иначе говоря, метатеория теории *T* для того, чтобы иметь возможность доказывать хотя бы непротиворечивость теории *T*, должна быть богаче *T*.

По существу при этом ставится под сомнение первоначальная, «максималистская» программа финитного подхода: нельзя построить математику как некоторую фиксированную совокупность средств, которые можно было бы объявить единственно законными и с их помощью строить метатеории любых теорий.

Не следует истолковывать результаты Геделя как крах формального подхода. Наличие алгоритмически неразрешимых проблем вовсе не бросает тень на теорию алгоритмов, а лишь сообщает «суровую правду» об устройстве мира, изучаемого этой теорией. Из этой правды не вытекает, что алгоритмический, конструктивный подход к решению проблемы не пригоден; хотя он чего-то и не может, но лишь потому, что этого не может никто. Точно так же невозможность полной формализации содержательно определенных теорий – это не недостаток подхода или концепции, а объективный факт, неустранимый никакой концепцией. Формальный подход остается основным конструктивным средством изучения множеств высказываний. Невозможность адекватной формализации теории означает, что надо либо искать формализуемые ее фрагменты, либо строить какую-то более сильную формальную теорию, которая, правда, снова будет неполна, но, быть может, будет содержать всю исходную теорию. В частности, методами, не формализуемыми в формальной арифметике, Генцен доказал непротиворечивость формальной арифметики.

Лекция 10

ТЕОРИЯ АЛГОРИТМОВ

1. ПРЕДВАРИТЕЛЬНОЕ ОБСУЖДЕНИЕ

Несколько вступительных слов. С алгоритмами, т. е. эффективными процедурами, однозначно приводящими к результату, математика имела дело всегда. Школьные методы умножения «столбиком» и деления «углом», метод исключения неизвестных при решении системы линейных уравнений, правило дифференцирования сложной функции, способ построения треугольника по трем заданным сторонам – все это алгоритмы. Однако, пока математика имела дело в основном с числами и вычислениями и понятие алгоритма отождествлялось с понятием метода вычисления, потребности в изучении самого этого понятия не возникало. Традиции организации вычислений складывались веками и стали составной частью общей научной культуры в той же степени, что и элементарные навыки логического мышления. Все многообразие вычислений комбинировалось из 10 – 15 четко определенных операций арифметики, тригонометрии и анализа. Поэтому понятие метода вычисления считалось изначально ясным и не нуждалось в специальных исследованиях.

До середины XIX в. единственной областью математики, работавшей с нечисловыми объектами, была геометрия, и как раз она, не имея возможности опираться на вычислительную интуицию человека, резко отличалась от остальной математики повышенными требованиями к строгости своих рассуждений. До сих пор любой современный шестиклассник, для которого математика — это мир вычислений (и в этом он мало отличается от типичного инженера), мучительно привыкает к понятиям доказательства и математического построения и никак не может понять, зачем доказывать равенство отрезков, когда их проще измерить, и зачем строить перпендикуляр с помощью циркуля и линейки, когда есть угольник с «готовым» прямым углом или транспортир.

Такое же мучительное привыкание к новым, более жестким требованиям строгости началось в математике во второй половине XIX в., оно стимулировалось в основном математикой нечисловых объектов – открытием неэвклидовых геометрий, появлением абстрактных алгебраических теорий типа теории групп и т.д. Одним из решающих обстоятельств, приведших к пересмотру оснований математики, т. е. принципов, лежащих в основе математических рассуждений, явилось создание Кантором теории множеств. Довольно быстро стало ясно, что понятия теории множеств в силу своей общности лежат в основе всего здания математики. Однако почти столь же быстро было показано, что некоторые кажущиеся вполне естественными рассуждения в рамках этой теории приводят к неразрешимым противоречиям – парадоксам теории множеств (которые упоминались). Все это потребовало

точного изучения принципов математических рассуждений (до сих пор казавшихся интуитивно ясными) математическими же средствами. Возникла особая отрасль математики – основания математики, или метаматематики.

Опыт парадоксов теории множеств научил математику крайне осторожно обращаться с бесконечностью и по возможности даже о бесконечности рассуждать с помощью финитных методов. Существо финитного подхода заключается в том, что он допускает только конечные комплексы действий над конечным числом объектов. Выяснение того, какие объекты и действия над ними следует считать точно определенными, какими свойствами и возможностями обладают комбинации элементарных действий, что можно и чего нельзя сделать с их помощью, – все это стало предметом теории алгоритмов и формальных систем, которая первоначально возникла в рамках метаматематики и стала важнейшей ее частью. Главным внутриматематическим приложением теории алгоритмов явились доказательства невозможности алгоритмического (т. е. точного и однозначного) решения некоторых математических проблем. Такие доказательства (да и точные формулировки доказываемых утверждений) неосуществимы без точного понятия алгоритма.

Пока техника использовала чисто вычислительные методы, эти высокие проблемы чистой математики ее мало интересовали. В технику термин «алгоритм» пришел вместе с кибернетикой. Если понятие метода вычисления не нуждалось в пояснениях, то понятие процесса управления пришлось вырабатывать практически заново. Понадобилось осознавать, каким требованиям должна удовлетворять последовательность действий (или ее описание), чтобы считаться конструктивно заданной, т. е. иметь право называться алгоритмом. В этом осознании огромную помощь инженерной интуиции оказала практика использования вычислительных машин, сделавшая понятие алгоритма ощутимой реальностью. С точки зрения современной практики алгоритм—это программа, а критерием алгоритмичности процесса является возможность его запрограммировать. Именно благодаря этой реальности алгоритма, а также потому, что подход инженера к математическим методам всегда был конструктивным, понятие алгоритма в технике за короткий срок стало необычайно популярным (быть может, даже больше, чем в самой математике).

Однако у всякой популярности есть свои издержки. В повседневной практике слово «алгоритм» употребляется слишком широко, теряя зачастую свой точный смысл. Приблизительные описания понятия «алгоритм» (вроде того, которое приведено в первой фразе этого параграфа) часто принимаются за точные определения. В результате за алгоритм зачастую выдается любая инструкция, разбитая на шаги. Появляются такие дикие словосочетания, как «алгоритм изобретения» (а ведь наличие «алгоритма изобретения» означало бы конец изобретательства как творческой деятельности).

Ясное представление о том, что такое алгоритм, важно, конечно, не только для правильного словоупотребления. Оно нужно и при разработке конкретных алгоритмов, особенно когда имеется в виду их последующее программирование. Однако оно еще важнее при наведении порядка в бурно растущем алгоритмическом хозяйстве. Чтобы ориентироваться в море алгоритмов, захлестнувшем техническую кибернетику, необходимо уметь сравнивать различные алгоритмы решения одних и тех же задач, причем не только по качеству решения, но и по характеристикам самих алгоритмов (числу действий, расходу памяти и т.д.). Такое сравнение невозможно без введения точного языка для обсуждения всех этих вопросов; иначе говоря, сами алгоритмы должны стать такими же предметами точного исследования, как и те объекты, для работы с которыми они предназначены.

Строгое исследование основных понятий теории алгоритмов начнется в следующем параграфе. Прежде чем приступить к нему, обсудим на неформальном уровне некоторые основные принципы, по которым строятся алгоритмы, и выясним, что же именно в понятии алгоритма нуждается в уточнении.

Основные требования к алгоритмам.

1. Первое, что следует отметить в любом алгоритме,− это то, что он применяется к исходным данным и выдает результаты. В привычных технических терминах это означает, что алгоритм имеет входы и выходы. Кроме того, в ходе работы алгоритма появляются промежуточные результаты, которые используются в дальнейшем. Таким образом, каждый алгоритм имеет дело с *данными* − входными, промежуточными и выходными. Поскольку мы собираемся уточнять понятие алгоритма, нужно уточнить и понятие данных, т. е. указать, каким требованиям должны удовлетворять объекты, чтобы алгоритмы могли с ними работать.

Ясно, что эти объекты должны быть четко определены и отличимы как друг от друга, так и от «необъектов». К алгоритмическим объектам относятся числа, векторы, матрицы смежностей графов, формулы. Изображения (например, рисунок графа) представляются менее естественными в качестве алгоритмических объектов. Если говорить о графе, то дело даже не в том, что в рисунке больше несущественных деталей и два человека один и тот же граф изобразят по-разному (в конце концов, разные матрицы смежности тоже могут задавать один и тот же граф с точностью до изоморфизма), а в том, что матрица смежности легко разбивается на элементы, причем из элементов всего двух видов (нулей и единиц) состоят матрицы любых графов, тогда как разбить на элементы рисунок гораздо труднее. Наконец, с такими объектами, как «хорошая книга» или «осмысленное утверждение», с которыми легко управляется любой человек (но каждый по-своему!), алгоритм работать откажется, пока они не будут описаны как данные с помощью других, более подходящих объектов.

Вместо того чтобы пытаться дать общее словесное определение четкой определенности объекта, в теории алгоритмов фиксируют конкретные конечные наборы исходных объектов (называемых элементарными) и конечный набор средств построения других объектов из элементарных. Набор элементарных объектов образует конечный *алфавит* исходных символов (цифр, букв и т. д.), из которых строятся другие объекты; типичным средством построения являются индуктивные определения, указывающие, как строить новые объекты из уже построенных. Простейшее индуктивное определение — это определение некоторого множества слов, классическим примером которого служит определение идентификатора в АЛГОЛе; идентификатор – это либо буква, либо идентификатор, к которому приписана справа буква или цифра. Слова конечной длины в конечных алфавитах (в частности, числа) – наиболее обычный тип алгоритмических данных, а число символов в слове (длина слова) – естественная единица измерения объема обрабатываемой информации. Более сложный случай алгоритмических объектов – формулы. Они также определяются индуктивно и также являются словами в конечном алфавите, однако не каждое слово в этом алфавите является формулой. В этом случае обычно основным алгоритмам предшествуют вспомогательные, которые проверяют, удовлетворяют ли исходные данные нужным требованиям. Такая проверка называется синтаксическим анализом.

2. Данные для своего размещения требуют *памяти.* Память обычно считается однородной и дискретной, т. е. состоит из одинаковых ячеек, причем каждая ячейка может содержать один символ алфавита данных. Таким образом, единицы измерения объема данных и памяти согласованы. При этом память может быть бесконечной. Вопрос о том, нужна ли одна память или несколько и, в частности, нужна ли отдельная память для каждого из трех видов данных (входных, выходных и промежуточных), решается по - разному.

3. Алгоритм состоит из отдельных э*лементарных* или *действий,* причем множество различных шагов, из которых составлен алгоритм, конечно. Типичный пример множества элементарных действий — система команд Обычно элементарный шаг имеет дело с фиксированным числом символов (это удобно, например, для измерения времени работы алгоритма числом проделанных шагов), однако это требование не всегда выполняется. Например, в ЭВМ третьего поколения есть команды типа «память - память», работающие с полями памяти переменной длины.

4. Последовательность шагов алгоритма *детерменирована,* т. е. после каждого шага либо указывается, шаг делать дальше, либо дается команда остановки, после чего работа алгоритма считается законченной.

5. Естественно от алгоритма потребовать *результативности* т. е. остановки после конечного числа шагов зависящего от данных) с указанием того, что считать результатом. В частности, всякий, кто предъявляет алгоритм решения некоторой задачи, например вычисления функции f(x), обязан показать, что алгоритм останавливается после конечного числа шагов (как говорят,

сходится) для, любого *x* из области задания. Однако проверить результативность (сходимость) гораздо труднее. В отличие от них сходимость обычно не удается установить простым просмотром описания алгоритма; общего же метода проверки сходимости годного для любого алгоритма *A* и любых данных будет показано далее, вообще не существует. Обсуждение трудностей, связанных с распознаванием сходимости далее.

6. Следует различать: а)описание алгоритма конструкцию или программу); б)механизм реализации алгоритма (например, ЭВМ), включающий средства остановки, реализации элементарных шагов, выдача результатов и обеспечения детерминированности, т.е. управления ходом вычисления; в) процесс реализации алгоритма, т. е. последовательность шагов, которая будет порождена при применении алгоритма к конкретным данным.

Будем предполагать, что описание алгоритма и механизм его реализации конечны (память, как уже говорилось, может быть бесконечной, но она не включается в механизм). Требования к конечности процесса реализации совпадают с требованиями результативности (см. п. 5).

Пример 1.

Рассмотрим следующую задачу: дана последовательность *P* из *n* положительных чисел (*n* — конечное, но произвольное число); требуется упорядочить их, т. е. построить последовательность R, в которой эти же числа расположены в порядке возрастания. Почти сразу же приходит в голову следующий простой способ ее решения: просматриваем *P* и находим в ней наименьшее число; вычеркиваем его из P и выписываем его в качестве первого числа R; снова обращаемся к P и находим в ней наименьшее число; приписываем его справа к полученной части *R* и так далее, до тех пор, пока в P не будут вычеркнуты все числа. Возникает естественный вопрос: что значит «и так далее». Для большей ясности перепишем описание способа решения в более четкой форме, разбив его на шаги и указав переходы между шагами.

Ш а г 1. Ищем в *P* наименьшее число.

Ш а г 2. Найденное число приписываем справа к *R* (в начальный момент *R* пуста) и вычеркиваем его из P.

Ш а г 3. Если в *P* нет чисел, то переходим к шагу 4. В противном случае переходим к шагу 1.

Ш а г 4. Конец. Результатом считать последовательность R, построенную к данному моменту.

Большинство читателей сочтет такое описание достаточно ясным (и даже излишне формальным), чтобы, пользуясь им, однозначно получить нужный результат. Однако это впечатление ясности опирается на некоторые неявные предположения, к правильности которых мы привыкли, но которые нетрудно нарушить. Например, что значит «дана последовательность чисел»? В нашем описании ничего не сказано, как найти наименьшее число среди таких чисел. В нем вообще не говорится о том, как искать наименьшие

числа, по-видимому, предполагается, что речь идет о числах, представленных в виде десятичных дробей, и что известно, как их сравнивать.

Необходимо уточнить формы представления данных. При этом нельзя просто заявить, что допустимо любое представление чисел. Ведь для каждого представления существует свой алфавит (который, помимо цифр, может включать запятые, скобки, знаки операций и функций и т. д.) и свой способ сравнения чисел (например, способ перевода в десятичную дробь), тогда как конечность алфавита требует фиксировать его заранее, а конечность описания алгоритма позволяет включить в него лишь заранее фиксированное число способов сравнения. Фиксация представления чисел в виде десятичных дробей так же не решает всех проблем. Сравнение 10—20-разрядных уже не может считаться элементарным действием: сразу нельзя сказать, какое из чисел больше: 90811557001,1 32899901467,0048. В машинных алгоритмах само представление числа еще требует дальнейшего уточнения: нужно во-первых, ограничить число разрядов (цифр) в числе, так как от этого зависит, сколько ячеек памяти будет занимать число, а во-вторых, договориться о способе» размещения десятичной запятой в числе, т, е. выбрать представление в виде числа с фиксированной или плавающей запятой, поскольку способы обработки этих двух представлений различны. Наконец, любой, кто имел дело с программированием, отметит, что на шаге 1 требуется узнать две вещи: само наименьшее число (чтобы записать его в *R)* и его место в Р, т. е. его адрес в той части *m* где хранится *P* (чтобы вычеркнуть его из Р), а следовательно нужно иметь средства указания этого адреса.

Таким образом, даже в этом простом примере несложный анализ показывает, что описанию, которое выглядит вполне ясным, еще далеко до алгоритма. Мы столкнулись здесь с необходимостью уточнить почти все основные характеристики алгоритма, которые отмечались ранее: алфавит данных и форму их представления, память и размещение в ней элементов *P* и *R,* элементарные шаги (поскольку шаг 1 явно не элементарен). Кроме того, становится ясным, что выбор механизма реализации (скажем, человека или ЭВМ) будет влиять и на сам характер уточнения: у человека требования к памяти, представлению данных и к элементарности шагов гораздо более слабые и «укрупненные», отдельные незначительные детали он может уточнить сам.

Пожалуй, только два требования к алгоритмам в приведенном описании выполнены в достаточной мере (они то и создают впечатление ясности). Довольно очевидна сходимость алгоритма: после выполнения шагов 1 и 2 либо работа заканчивается, либо из *P* вычеркивается одно число; поэтому ровно после *n* выполнений шагов 1 и 2 из Р будут вычеркнуты все числа и алгоритм остановится, а *R* будет результатом. Кроме того, не вызывает сомнения детерминированность: после каждого шага ясно, что делать дальше, если учесть, что здесь и в дальнейшем используется общепринятое соглашение –

если шаг не содержит указаний о дальнейшем переходе, то выполняем шаг, следующий за ним в описании. Поскольку использованные в примере средства обеспечения детерминированности носят довольно общий характер, остановимся на них несколько подробнее.

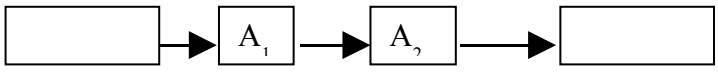
Блок-схемы алгоритмов. Связи между шагами можно изобразить в виде графа. Для примера 1 граф изображен на рис. 1.

Рис.1

Такой граф, в котором шаги, а ребрам — переходы между шагами, называется блок-схемой алгоритма. Его вершины могут быть двух видов: вершины, из которых выходит одно ребро (их называют операторами), и вершины, из которых выходят два ребра (их называют логическими условиями, или предикатами). Кроме того, имеется единственный оператор конца, из которого не выходит ни одного ребра, и единственный оператор начала. Важной особенностью блок-схемы является то, что связи, которые она описывает, не зависят от того, являются ли шаги элементарными или представляют собой самостоятельные алгоритмы,— как говорят в программировании, блоки (по существу шаг 1 таковым и является). Возможность «разблочивать» алгоритм хорошо известна в программировании и широко там используется: большой алгоритм разбивается на блоки, которые можно раздать для программирования разным лицам. Для данного блока неважно, как устроены другие блоки; для программирования блока достаточно знать, где лежит вся исходная информация, какова форма ее представления, что должен делать блок и куда записать результат.

С помощью блок-схем можно, наоборот, несколько алгоритмов, рассматриваемых как блоки, связать в один большой алгоритм. В частности, если алгоритм A_1 вычисляющий функцию $f(x)$, соединен с алгоритмом A_2, вычисляющим функцию $f(x)$ (рис. 2), и при этом исходными данными для A_2 служит результат A_1, то полученная блок-схема задает алгоритм, вычисляющий

Рис. 2



$f_2(f_1(x))$, т. е. *композицию* f_1 и f_2. Такое соединение алгоритм называется *композицией* алгоритмов.

На блок-схеме хорошо видна разница между описанием алгоритма и процессом его реализации. Описание – это граф; процесс реализации – это путь в графе. Различные пути в одном и том же графе возникают при различных данных, которые создают разные логические условия в точках

разветвления. Отсутствие сходимости означает, означает, что в процессе вычисления не появляется условий, ведущих к концу, и процесс идет по бесконечному пути (зацикливается).

При всей наглядности языка блок-схем не следует, однако, переоценивать его возможности. Он достаточно груб и отражает связи лишь по управлению (что делать в следующий момент, т. е. какому блоку передать управление), а не по информации (где этому блоку брать исходные данные). Например, рис. 2 при сделанной оговорке относительно данных изображает вычисление $f_2(f_1(x))$, однако он же мог изображать последовательное вычисление двух независимых функций $f_1(5)$ и $f_2(100)$, порядок которых несуществен. Блок-схемы не содержат сведений ни о данных, ни о памяти, ни об используемом наборе элементарных шагов. В частности, надо иметь в виду, что если в блок-схеме нет циклов, это еще не значит, что нет циклов в алгоритме (они могут быть в каком-нибудь неэлементарном блоке). По существу блок-схемы − это не язык; а средство, правда, очень удобное, для одной цели — описания детерминизма алгоритма. Оно будет неоднократно использоваться в дальнейшем с одним видоизменением: условия, т.е. точки разветвления, могут быть не только двоичными, но и многозначными; важно лишь, чтобы верным был ровно один из возможных ответов. Примеры таких условий: а) $x<0$, $x=0$, $x>0$; б) $x<5$, $5<=x<20$, $x=20$, $x<20$.

Лекция 11

О подходах к уточнению понятия «алгоритм».

Ранее были сформулированы основные требования к алгоритмам. Однако понятия, использованные в формулировках (ясность, четкость, элементарность), сами нуждаются в уточнении. Очевидно, что их словесные определения будут содержать новые понятия, которые потребуют уточнения, и т. д.

Поэтому в теории алгоритмов принимается другой подход: выбирается конечный набор исходных объектов, которые объявляются элементарными, и конечный набор способов построения из них новых объектов. Этот подход был использован при обсуждении вопроса о данных: уточнением понятия «данные» в дальнейшем будем считать множества слов в конечных алфавитах. Для уточнения детерминизма будут использоваться либо блок-схемы и эквивалентные им словесные описания, либо описание механизма реализации алгоритма. Кроме того, нужно зафиксировать набор элементарных шагов и договориться об организации памяти. После этого получится конкретная алгоритмическая модель.

Рассматриваемые алгоритмические модели, претендуют на право считаться формализацией понятия «алгоритм». Это значит, что они должны быть универсальными, т.е. допускать описание любых алгоритмов. Может возникнуть естественное возражение против предлагаемого подхода: не

приведет ли выбор конкретных средств к потере общности формализации? Если иметь ввиду основные цели, стоящие при создании теории алгоритмов, - универсальность и связанную с ней возможность говорить в рамках какой-либо модели о свойствах алгоритмов вообще, то это возражение снимается. Во-первых, доказывается сводимость одних моделей к другим, т.е. показывается, что всякий алгоритм, описанный средством одной модели, может быть описан средствами другой. Во-вторых, благодаря взаимной сводимости моделей в теории алгоритмов удалось выработать инвариантную по отношению к моделям систему понятий, позволяющую говорить о свойствах алгоритмов независимо от того, какая формализация алгоритма выбрана. Эта система понятий основана на понятии вычислимой функции, т. е. функции, для вычисления которой существует алгоритм. Общее понятие вычислимости и его свойства будут более подробно рассмотрены.

Тем не менее, хотя общность формализации в конкретной модели не теряется, различный выбор исходных средств приводит к моделям разного вида. Можно выделить три основных типа *универсальных алгоритмических моделей* различающихся исходными эвристическими соображениями относительно того, что такое алгоритм.

I. 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.

II. Второй тип основан на представлении об алгоритме как о некотором детерминированном устройстве, способном выполнять в каждый отдельный момент лишь весьма примитивные операции. Такое представление не оставляет сомнений в однозначности алгоритма и элементарности его шагов. Кроме того, эвристика этих моделей близка к ЭВМ и, следовательно, инженерной интуиции. Основной теоретической моделью этого типа (созданной в 30-х годах —раньше ЭВМ!) является машина Тьюринга.

III. Третий тип алгоритмических моделей — это преобразования слов в произвольных алфавитах, в которых элементарными операциями являются подстановки, т. е. замены куска слова (подслова) другим словом. Преимущества этого типа — в его максимальной абстрактности и возможности применить понятие алгоритма к объектам произвольной (не обязательно числовой) природы. Впрочем, как будет ясно из дальнейшего, модели второго и третьего типа довольно близки (их взаимная сводимость доказывается просто) и отличаются в основном эвристическими акцентами. Примеры моделей этого типа — канонические системы Поста и нормальные алгоритмы Маркова.

Лекция 12

МАШИНЫ ТЬЮРИНГА

Основные определения.

Определение. Машина Тьюринга состоит: 1) управляющего устройства, которое может находится в одном из состояний, образующих конечное множество $Q=\{q_1 ..., q_n\}$; 2) ленты, разбитой на ячейки, в каждой из которых может быть записан один из символов конечного алфавита $A=\{a_1, ..., a_m\}$; 3) устройства обращения к ленте, *считывающей* и *пишущей* головки, которая в каждый момент времени обозревает ячейку ленты, в зависимости от символа в этой ячейке и состояния управляющего устройства, записывает в ячейку символ (быть может, совпадающий с прежним или пустой, т. е. стирает символ), сдвигается на ячейку влево или вправо или остается на месте; при этом *управляющее устройство* переходит в новое состояние (или остается в старом.

Среди состояний управляющего устройства выделены начальное состояние q_1 и заключительное состояние, которое будем обозначать q_z (*z* здесь понимается не как числовая переменная, а как мнемонический знак конца). В начальном состоянии машина находится перед началом работы; попав в заключительное состояние, машина останавливается.

Таким образом, *память машины* Тьюринга – это конечное множество состояний (внутренняя память) и лента (внешняя память). Лента бесконечна в обе стороны, однако в начальный момент времени только конечное число ячеек ленты заполнено непустыми символами, остальные ячейки пусты, т. е. содержат пустой символ λ (пробел). Из характера работы машины следует, что и в любой последующий момент времени лишь конечный отрезок ленты будет заполнен символами. Поэтому важна не фактическая (как говорят в математике, актуальная) бесконечность ленты, а ее неограниченность, т. е. возможность писать на ней сколь угодно длинные, но конечные слова.

Данные машины Тьюринга – это слова в алфавите ленты; на ленте записываются и исходные данные, и окончательные результаты.

Элементарные шаги машины – это считывание и запись символов, сдвиг головки на ячейку влево и вправо, а также переход управляющего устройства в следующее состояние.

Детерминированность машины, т. е. последовательность ее шагов, определяется следующим образом: для любого внутреннего состояния q_i и символа a_j однозначно заданы: а) следующее состояние q'_i; *б)* символ a'_j который нужно записать вместо a_j в ту же ячейку (стирание символа будем понимать как запись пустого символа λ); в) направление сдвига головки d_k, обозначаемое одним из трех символов, *L* (влево), *R* (вправо), *E* (на месте). Это задание может описываться либо системой правил (команд), имеющих вид:

$$q_i\ a_j \rightarrow q'_i\ a'_j\ d_k, \qquad (1)$$

либо таблицей, строкам которой соответствуют состояния, столбцам — входные символы, а на пересечении строки q_i и столбца a_j записана тройка символов q'_i a'_j d_k и, наконец, блок-схемой, которую будем называть диаграммой переходов. В диаграмме переходов состояниям соответствуют вершины, а правилу вида (1)—ребро, ведущее из q_i в q'_i , на котором написано $a_j \to a'_j\ d_k$,. Условие однозначности требует, чтобы для любого j и любого i≠*z* в системе команд имелась одна команда, аналогичная (1), с левой частью $q_i\ \ a_j$; состояние q_z в левых частях команд не встречается. На диаграмме переходов это выражается условием, что из каждой вершины, кроме q_z, выходят ровно *m* ребер, причем на разных ребрах левые части различны; в вершине q_z нет выходящих ребер. В дальнейшем договоримся опускать символы $q'_i\ a'_j$, если $q'_i = q_i$, $a'_j = a_j$.

Полное состояние машины Тьюринга, по которому однозначно можно определить ее дальнейшее поведение, определяется ее внутренним состоянием, состоянием ленты (т. е. словом, записанным на ленте) и положением головки на ленте.

Полное состояние будем называть *конфигурацией,* или машинным словом, и обозначать тройкой $\alpha_1 q_i \alpha_2$, где q_i — текущее внутреннее состояние, α_1 — слово слева от головки, а α_2—слово, образованное символом, обозреваемым головкой, и символами справа от него, причем слева от α_1 и справа от α_2 нет непустых символов. Например, конфигурация с внутренним состоянием q_i, в которой на ленте записано *a b c d e,* а головка обозревает *d,* запишется как $abcq_ide$.

Стандартной начальной конфигурацией назовем конфигурацию вида $q_1\alpha$, т. е. конфигурацию, содержащую начальное состояние, в которой головка обозревает крайний левый символ слова, написанного на ленте. Аналогично *стандартной заключительной конфигурацией* назовем конфигурацию вида $q_z\alpha$. Ко незаключительной конфигурации *K* машины *T* применима ровно одна команда вида (1), которая *K* переводит в конфигурацию *K.'*. Это отношение между конфигурациями обозначим $K \to_T K$; если из контекста ясно, о какой машине *T* идет речь, индекс *T* будем опускать. Если для K_1 и $K._n$ существует последовательность конфигураций $K_1, K_2, ..., K_n$, такая, что $K_1 \to_T K_2 \to_T ... \to_T K_n$, обозначим это $K_1 =>_T K_n$.

Например, если в системе команд машины *T* имеются команды $q_2a_5 \to q_3a_4R$ и $q_3a_1\mathrm{L} \to q_4a_2$, то $q_2a_5a_1a_2 \to a_4q_3a_1a_2 \to q_4a_2q_4a_4a_2a_2$ и, следовательно $q_2a_5a_1a_2 =>_T q_4a_2q_4a_4a_2a_2$. Последовательность конфигураций $K_1 \to_T K_2 \to_T K_3 \to_T$…однозначно определяется исходной конфигурацией K_1 и полностью описывает работу машины *T,* начиная с K_1. Она конечна, если в ней встретится заключительная конфигурация, и бесконечна в противном случае.

Пример 1.

а. Машина с алфавитом A=={1, λ}, состояниями $\{q_1, q_z\}$ и системой команд $q_1 1 \to q_1 1R$, $q_1 \lambda \to q_1 1R$ из любой начальной конфигурации будет работать бесконечно, заполняя единицами всю ленту вправо от начальной точки.

б. Для любой машины T, *если* $K_1 =>_T K_l =>_T K_j$ и $K_i = K_j$, последовательность $K_1 => K_i => K_j => ...$ является бесконечной; ее отрезок $K_1 => K_j$ будет повторяться (зациклится).

Если $\alpha_1 q_1 \alpha_2 => \beta_1 q_z \beta_2$, то будем говорить, что машина *T* перерабатывает слово $\alpha_1\alpha_2$ в слово $\beta_1\beta_2$, и обозначать это $T(\alpha_1\alpha_2) == \beta_1\beta_2$. Запись $T(\alpha)$ иногда будем употреблять и в другом смысле — просто как обозначение машины *T* с исходными значениями α.

Действия машины Тьюринга.

Для того чтобы говорить о том, что могут делать машины Тьюринга, необходимо уточнить, как будет интерпретироваться их поведение, и как будут представляться данные. Исходными данными машины Тьюринга будем считать записанные на ленте слова в алфавите исходных данных $A_{исх}$ ($A_{исх} \subseteq A$) и векторы из таких слов (словарные векторы над $A_{исх}$). Это значит, что для каждой машины будут рассматриваться только те начальные конфигурации, в которых на ленте записаны словарные векторы над $A_{исх}$. Запись на ленте словарного вектора (α_1..., α_k) назовем правильной, если она имеет вид $\alpha_1\lambda\alpha_2\lambda... \alpha_{k-1}\lambda\alpha_k$. (при условии, что $\lambda \notin A_{исх}$) либо $\alpha_1 * \alpha_2 * ... \alpha_{k-1} * \alpha_k$, где * — специальный символ-разделитель, не входящий в $A_{исх}$. Для любого вектора $V_{исх}$ над $A_{исх}$ машина *T* либо работает бесконечно, либо перерабатывает его в совокупность слов (разделенных пробелами) в алфавите, который назовем алфавитом результатов и обозначим $A_{рез}$; $A_{исх}$ и $A_{реэ}$ могут пересекаться и даже совпадать. В ходе работы на ленте могут появляться символы, не входящие в $A_{исх}$ и $A_{реэ}$ и образующие промежуточный алфавит $A_{пр}$ (содержащий, в частности, разделитель). Таким образом, алфавит ленты $A == A_{исх} \cup A_{реэ} \cup A_{пр}$. В простейшем случае $A_{исх} == A_{рез}$ и $A == A_{исх} \cup \{\lambda\}$.

Пусть *f* — функция, отображающая множество векторов над $A_{исх}$ в множество векторов над $A_{рез}$. Машина *T правильно вычисляет функцию f,* если: 1) для любых *V* и *W,* таких, что $f(V) = W$, $q_1 V^* =>_z W^*$, где V* и W*, — правильные записи *V* и *W* соответственно; 2) для любого *V,* такого, что *f(V)* не определена, машина *T,* запущенная в стандартной начальной конфигурации $q_1 V^*$, работает бесконечно. Если для *f* существует машина *T,* которая ее правильно вычисляет, функция *f* называется ***правильно вычислимой по Тьюрингу***.

С другой стороны, всякой правильно вычисляющей машине Тьюринга, т. е. машине, которая, начав со стандартной начальной конфигурации $q_1\alpha$, может остановиться только в стандартной заключительной конфигурации $q_z \beta$, можно поставить в соответствие вычисляемую ей функцию. Две машины

Тьюринга с одинаковым алфавитом $A_{исх}$ будет называть эквивалентными, если они вычисляют одну и ту же функцию. В частности, машины из примеров 1 а) б) эквивалентны, так как они вычисляют одну и ту же нигде не определенную (пустую) функцию.

Пример 2.

Если машина *T* содержит команды $q_i a_j \to q'_i a'_j E$ и $q'_i a'_j \to q''_i a''_j d_k$, то, заменив эти две команды командой $q_i a_j \to q''_i a''_j d_k$, получим машину *T',* эквивалентную *T*. Путем таких преобразований можно в машине *T* убрать все команды, содержащие E, для случая, когда q'_i — незаключительное состояние, при этом может сократиться число состояний (некоторые q'_i не войдут в правые части новых команд и станут недостижимыми из q_1). Если q'_i — заключительное состояние, то, введя новое заключительное состояние q_{n+1} и заменив команду $q_i a_j \to q'_i a'_j E$ на m+1 команду $q_i a_j \to q'_i a'_j R$, $q'_i a_1 \to q_{n+1} a_1 L, \ldots$, $q'_i a_m \to q_{n+1} a_m L$ (*m*—число букв в А), получим машину, также эквивалентную *T*. Таким образом, для любой машины *T* существует эквивалентная ей машина, не содержащая в командах *E;* поэтому можно рассматривать машины, головки которых на каждом шаге движутся.

Определения, связанные с вычислением функций, заданных на словарных векторах, даны с явным «запасом общности» и имеют в виду переработку нечисловых объектов. В дальнейшем это понадобится; однако в ближайшее время будут рассматриваться числовые функции, точнее, функции, отображающие ***N*** в ***N***. Договоримся представлять натуральные числа в единичном (*унарном*) коде, т.е. для всех числовых функций $A_{исх} == \{1\}$ либо $A_{исх} == \{1, *\}$ и число *x* представляется словом $1\ldots1 == 1^x$, состоящим из *x* единиц. Таким образом, числовая функция $f(x_1, \ldots, x_n)$ правильно вычислима по Тьюрингу, если существует машина *T,* такая, что $q_1 1^{x1} * 1^{x2} * \ldots 1^{xn} \Rightarrow_T q_z 1^y$, когда $f(x_1, \ldots, x_n) = y$, и *T* работает бесконечно, начиная с $q_1 1^{x1} * 1^{x2} * \ldots 1^{xn}$, когда $f(x_1, \ldots, x_n)$ не определена.

Начав с общих определений абстрактных машин, мы затем ввели при определении вычислений на этих машинах ряд ограничений, связанных с правильной записью, правильным вычислением, представлением чисел в весьма неэкономном единичном коде и т.д. Не теряется ли при этом общность? Действительно, возможны другие определения вычисления: например, можно в заключительной конфигурации допускать символы из промежуточного алфавита $A_{пр}$, а результатом считать слово в $A_{рез}$, которое получится, если символы из $A_{пр}$ выкинуть и «сдвинуть» оставшиеся куски. В частности, если $A_{рез} = \{1\}$, то результатом будет число, равное числу единиц на ленте в заключительной конфигурации. Оказывается, что потери общности не происходит; в частности, если функция вычислима в последнем смысле, то она правильно вычислима. Не будем заниматься детальным доказательством этого утверждения в общем виде, однако проиллюстрируем его справедливость на примерах (см. далее пример).

Обычно прилагательное «правильный» будем опускать и говорить просто о функциях, *вычислимых по Тьюрингу*.

Пример 3.

а. *Сложение*. Во введенном ранее представлении чисел сложить числа a и b – это значит слово $1^a * 1^b$ переработать в слово 1^{a+b}, т.е. удалить разделитель* и сдвинуть одно из слагаемых, скажем первое, к другому. Это преобразование осуществляет машина T_+ *с* четырьмя состояниями и следующей системой команд (первая команда введена для случая, когда a=0 и исходное слово имеет вид $*1^b$):

$$q_1* \to q_2\lambda R;$$
$$q_1 1 \to q_2\lambda R;$$
$$q_2 1 \to q_2 1R;$$
$$q_2* \to q_3 1L;$$
$$q_3 1 \to q_3 1R;$$
$$q_3\lambda \to q_2\lambda R;$$

В этой системе команд перечислены не все сочетания состояний машины и символов ленты: опущены те из них, которые при стандартной начальной конфигурации никогда не встретятся. Опускать ненужные команды будем в дальнейшем; в таблицах это будет отмечено прочерками. Диаграмма переходов T_+ приведена на рис. 1; заключительное состояние отмечено двойным кружком.

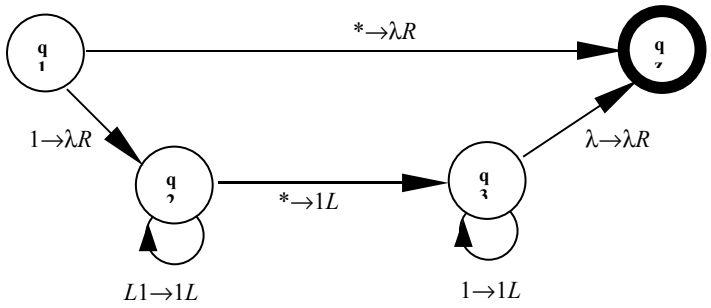


Рис. 1

б. *Копирование* (перезапись) слова, т. е. переработка слова α в α*α. Для чисел эту задачу решает машина $T_{коп}$, система команд которой приведена в табл. 1.

Таблица 1

	1	λ	*	0
q_1	$q_2 0R$	$q_z \lambda R$	$q_1 *L$	$q_1 1L$
q_2	$q_2 1R$	$q_3 *R$	$q_3 *R$	
q_3	$q_3 1R$	$q_4 1L$		
q_4	$q_4 1L$		$q_4 *L$	$q_1 0R$

Диаграмма переходов $T_{\text{коп}}$ дана на рис. 2. На этой диаграмме (а также последующих) приняты сокращения:

1) если из q_i в q_j ведут два ребра с одинаковой правой частью, то они объединяются в одно ребро, на котором левые части записаны через запятую;

2) если символ на ленте не изменяется, то он в правой части команды не пишется. На петле в q_4 использованы одновременно оба сокращения.

Машина $T_{\text{коп}}$ при каждом проходе исходного числа 1^a заменяет левую из его единиц нулем и пишет (в состоянии q_3) одну единицу справа от 1^a в ближайшую пустую клетку. При первом проходе, кроме того, в состоянии q_2 ставится маркер. Таким образом, копия 1^a строится за a проходов. После записи очередной единицы машина переходит в состояние q_4, которое передвигает головку влево от ближайшего нуля, после чего машина переходит в q_1 и цикл повторяется. Он прерывается, когда q_1 обнаруживает на ленте не единицу, а маркер. Это значит, что все единицы 1^a исчерпаны, т.е. сделано a проходов. Тогда головка возвращается влево в свое исходное положение, заменяя по дороге все нули единицами.

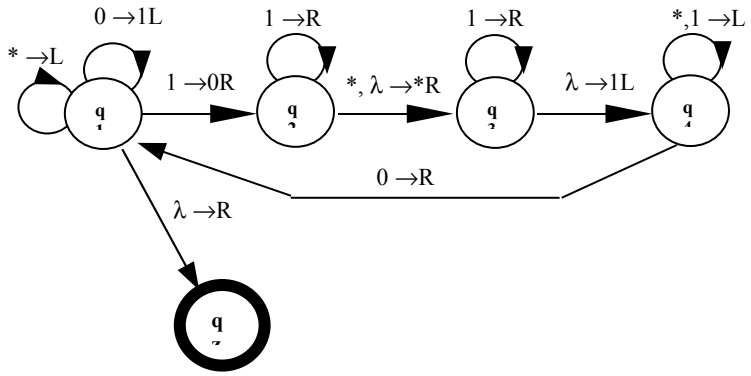


Рис. 2

Операции над машинами Тьюринга.

Работа машины Тьюринга полностью определяется исходными данными и системой команд. Однако для понимания того, как конкретная машина решает данную задачу, как правило, возникает потребность в содержательных пояснениях типа тех, которые приводились для машины $T_{\text{коп}}$. Эти пояснения часто можно сделать более формальными и точными,

если использовать блок-схемы и некоторые операции над машинами Тьюринга.

Композицией двух функций $f_1(x)$ и $f_2(x)$ называется функция $g(x)=f_2(f_1(x))$, которая получается при применении f_2 к результату вычисления f_1. Для того чтобы $g\{x)$ была определена при данном x, необходимо и достаточно, чтобы f_1 была определена на x и f_2 была определена на $f_1(x)$.

Теорема 1. Если $f_1(x)$ и $f_2(x)$ вычислимы по Тьюрингу, то их композиция $g(x)=f_2(f_1(x))$, также вычислима по Тьюрингу.

Пусть T_1 — машина, вычисляющая f_1, а T_2 — машина, вычисляющая f_2 и множества их состояний соответственно $Q_1==\{q_{11}, \ldots, q_{1n1}\}$ и $Q_2==\{q_{21}, \ldots, q_{2n1}\}$. Построим диаграмму переходов машины T из диаграмм T_1 и T_2 следующим образом: отождествим начальную вершину q_{21} диаграммы машины T_2 с конечной вершиной q_{1z} диаграммы машины T_1 (для систем команд это равносильно тому, что систему команд T_2 приписываем к системе команд T_1 и при этом q_{1z} в командах T_1 заменяем на q_{21}). Получим диаграмму с n_1+n_2-1 состояниями. Начальным состоянием T объявим q_{11}, а заключительным – q_{2z}. Для простоты обозначений будем считать f_1 и f_2 числовыми функциями одной переменной.

Пусть $f_2(f_1(x))$ определена. Тогда $T_1(1^x) = 1^{f1(x)}$ и $q_{11} \Rightarrow_{T1} q_{1z}\, 1^{f1(x)}$. Машина T пройдет ту же последовательность конфигураций с той разницей, что вместо $q_{1z}\, 1^{f1(x)}$ она перейдет в $q_{21}\, 1^{f1(x)}$. Эта конфигурация является стандартной начальной конфигурацией для машины T_2, поэтому $q_{21}\, 1^{f1(x)} \Rightarrow_{T2} q_{2z}1^{f2(f1(x))}$. Но так как все команды T_2, содержатся в T, то $q_{11}1^x \Rightarrow_T q_{21}\, 1^{f1(x)} \Rightarrow_T q_{2z}1^{f2(f1(x))}$ и, следовательно, $T(1^x) == 1^{f2(f1(x))}$. Если же $f_2(f_1(x))$ не определена, то T_1 или T_2 не остановится и, следовательно, машина T не остановится. Итак, машина T вычисляет $f_2(f_1(x))$.

Построенную таким образом машину T будем называть *композицией* машин T_1 и T_2 и обозначать $T_2(T_1)$, а также изображать блок-схемой (рис. 3).

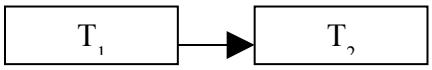


Рис.3

Эта блок-схема имеет более точный смысл, чем изображенная на рис.2 (введение в теорию алгоритмов), так как она всегда предполагает, что исходными данными машины T_2 являются результаты T_1. При этом они уже «готовы к употреблению», так как благодаря правильной вычислимости (которая существенна при композиции) заключительная конфигурация T_1 легко превращается в стандартную начальную конфигурацию T_2.

Поскольку машина Тьюринга вектор над $A_{исх}$ воспринимает как слово в алфавите $A_{исх} \cup \{*\}$, определение композиции и теорема 1 остаются в силе, если T_1 и T_2 вычисляют функции от нескольких переменных. Важно лишь,

чтобы данные для T_2 были в обусловленном виде подготовлены машиной T_1. Это видно на следующем примере.

Пример 4. Машина, диаграмма которой приведена на рис.4, —это машина $T_+(T_{коп})$. Она вычисляет функцию $f(x)=2x$ для $x\neq0$; при этом машина $T_{коп}$ строит двухкомпонентный вектор, а $T+$ вычисляет функцию от двух переменных.

Для удобства последующих построений установим следующий важный факт. Оказывается, что для вычисления на машине Тьюринга достаточно, чтобы лента была бесконечной только в одну сторону, например вправо. Такая лента *называется правой полулентой.*

Теорема 2. Любая функция, вычислимая по Тьюрингу, вычислима на машине Тьюринга с правой полулентой; иначе говоря, для любой машины Тьюринга T существует эквивалентная ей машина T_R с правой полулентой. Аналогичная теорема формулируется для левой полуленты.

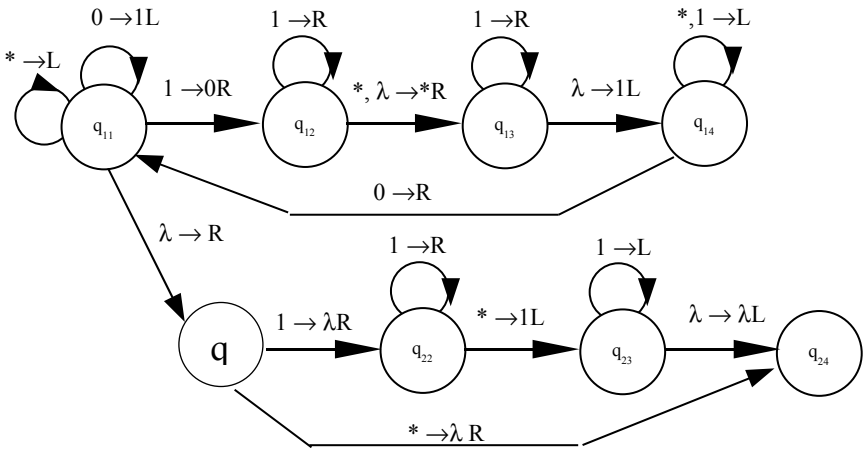


Рис.4

Можно предположить, по крайней мере, две идеи построения такой эквивалентной машины: 1) всякий раз, когда головке прежней машины надо зайти за левый край ленты, новая машина предварительно сдвинет все написанное (вместе с головкой) на клетку вправо; б) лента «перегибается пополам»; при этом ячейки правой половины прежней ленты размещаются, скажем, в нечетных ячейках полуленты, а ячейки левой половины − в четных.

Лекция 14

Вычисление предикатов на машинах Тьюринга

Рассмотрим теперь вычисление предикатов на машинах Тьюринга. Машина T вычисляет предикат Р(α) (α—слово в $A_{исх}$), если T(α)==ϖ, где ϖ ==И, когда Р(α) истинно, и ϖ=Л, когда Р(α) ложно. Если же Р(α) не определен, то машина T, как и при вычислении функций, не останавливается. При обычном вычислении предиката уничтожается α, что может оказаться неудобным, если после T должна работать другая машина. Поэтому введем понятие вычисления с восстановлением: машина T вычисляет Р(α) с восстановлением, если Т(α)==ϖα. Если существует машина T, вычисляющая Р(α), то существует и $T^{\sim}$, вычисляющая Р(α) с восстановлением. Действительно, вычисление с восстановлением можно представить следующей последовательностью конфигураций: $q_1\alpha \Rightarrow q_{n1}\alpha^*\alpha \Rightarrow \alpha^* q_{n2}\alpha \Rightarrow \alpha^* q_{n3}\varpi \Rightarrow q_{n4}\varpi\alpha$. Первая часть этой последовательности реализуется машиной $T_{коп}$, вторая—простым сдвигом головки до маркера *, третья—машиной T_R, вычисляющей Р(α) на правой полуленте (одного копирования мало для восстановления, так как копию может испортить основная машина T; нужна машина, не заходящая влево от α), четвертая —переносом ϖ в крайнее левое положение. Машина T является композицией указанных четырех машин. Однако в конкретных случаях возможны и более простые способы восстановления α.

Пример 5. Машина с диаграммой на рис. 5 вычисляет предикат «α—четное число»; головка достигает конца числа в состоянии q_2, если число единиц четно, и в состоянии q_3 если число единиц нечетно, после чего она перемещается в исходное положение в состоянии q_4 либо q_5 и печатает И либо Л соответственно. Для того чтобы этот предикат вычислялся с восстановлением, достаточно в петлях q_4 и q_5 не стирать, а сохранять единицы, т. е. заменить команды 1→λL на команды 1→L.

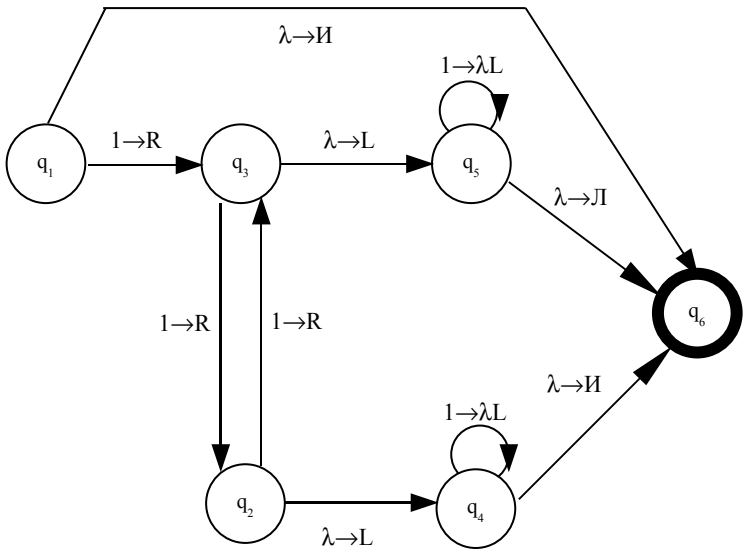


Рис.5

Так как машина Тьюринга с алфавитом $A_{исх}$= {И, Л} и командами q_1И $\rightarrow q_z$ЛЕ и q_1Л$\rightarrow$ q_zИЕ вычисляет отрицание логической переменной, то из вычислимости всюду определенного P(α) следует вычислимость ⌝P(α).

Пусть функция $f(\alpha)$ задана описанием: «если $P(\alpha)$ истинно, то $f(\alpha)=g_1(\alpha)$, иначе $f(\alpha)=g_2(\alpha)$» (под «иначе» имеется в виду «если $P(\alpha)$ ложно»; если же $P(\alpha)$ не определен, то $f(\alpha)$ также не определена). Функция $f(\alpha)$ называется *разветвлением* или *условным переходом* $g_1(\alpha)$ и $g_2(\alpha)$ по условию $P(\alpha)$.

Теорема 3. Если $g_1(\alpha), g_2(\alpha)$ и $P(\alpha)$ вычислимы по Тьюрингу, то разветвление g_1 и g_2 no P также вычислимо.

Пусть T_1 — машина с состояниями $q_{11}, q_{12}, \ldots, q_{1n1}$, и системой команд Σ_1, вычисляющая g_1; T_2 — машина с состояниями состояниями $q_{21}, q_{22}, \ldots, q_{2n2}$, и системой команд Σ_2, вычисляющая g_2; *Tp* вычисляет с восстановлением $P(\alpha)$. Тогда машина *T,* вычисляющая разветвление g_1 и g_2 по *P,* — это композиция *Tp* и машины *Tз*, система команд Σ_3 которой имеет следующий вид:

$\Sigma_3 = \Sigma_1 \cup \Sigma_2 \cup \{q_{31}И \rightarrow \lambda\, q_{11}R,\ q_{31}Л \rightarrow \lambda\, q_{21}R,\ q_{1z}И \rightarrow q_{2z}E\}$.

Первые две из новых команд передают управление системам команд Σ_1 или Σ_2 в зависимости от значения предиката $P(\alpha)$. Третья команда введена для того, чтобы *T*з имела одно заключительное состояние q_{2z}. Отсутствие символа ленты в этой команде означает, что она выполняется при любом символе.

Разветвление встречается в структуре диаграмм переходов многих машин Тьюринга.

Пример 6.

а. В примере 3, а было описано построение машины *T*+ для сложения двух чисел. На рис. 6 приведена диаграмма машины *T*++ для сложения *n* чисел *(n*=1, 2...). Цикл из состояний q_1, q_2, q_3 — это «зацикленная» машина *T*+, в которой заключительное состояние совмещено с начальным.

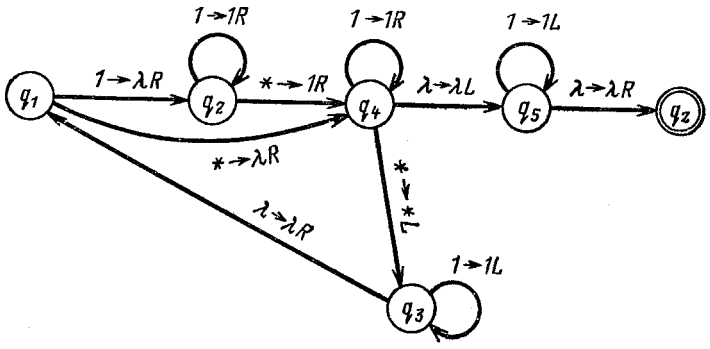


Рис.6

Сумма, полученная на очередном цикле, является первым слагаемым следующего цикла. Состояние q_4 реализует разветвление. В нем проверяется условие — есть ли второе слагаемое. Если да (о чем говорит наличие маркера *), то происходит переход к следующему циклу; если нет (о чем говорит λ после единиц), то машина выходит из цикла.

б. Пусть машина T с алфавитом A_T и с Аисх = {1} не является правильно вычисляющей и ее заключительная конфигурация стандартна, но может содержать любые символы из A_T (кроме пробелов), а результат интерпретируется как число, равное числу единиц на ленте. Построим для T машину T'++, которая работает как T++, а все символы, кроме 1 и λ, она воспринимает как маркеры, т. е. команды для них – те же, что и для *. Тогда T'++ соберет все единицы в один массив и, следовательно, T'++ (T'++(α)) правильно вычисляет функцию, вычисляемую машиной T.

Пример 6, а иллюстрирует важный частный случай разветвления — выход из цикла по условию, хорошо известный в программировании. В словесных описаниях (см. пример 1) и во многих языках программирования (например, в АЛГОЛе) этот случай формулируется так: повторять вычисление f_1 до тех пор, пока истинно условие Р; если P ложно, перейти к вычислению f_2. Благодаря вычислимости композиции и разветвления словесные описания и язык блок-схем можно сделать вполне точным языком для описания работы машин Тьюринга. Каждый блок — это множество состояний, в котором выделены начальное и заключительное состояния, и система команд. Правильное вычисление для промежуточных блоков не обязательно, поэтому при стыковке блоков важно согласовывать всю конфигурацию. Переход к блоку — это обязательно переход в его начальное состояние. Машина Тьюринга, описываемая блок-схемой, — это объединение состояний и команд, содержащихся во всех блоках. В частности, блоком может быть одно состояние. Блоки, вычисляющие предикаты, обозначим буквой P.

Пример 7.

На рис. 7 приведена блок-схема машины Тьюринга $T_\times$, осуществляющей умножение двух чисел:

$T_\times(1^a * 1^b) = 1^{ab}$. Ее заключительное состояние – q_{03}. Блоки реализуют следующие указания и вычисления:

P_1 — вычислить с восстановлением предикат «оба слагаемых больше нуля»;

T_1 — стереть все непустые символы справа;

T_2 — установить головку у ячейки, следующей за маркером *; маркер * заменить на ~;

*T*коп—см. пример 4,б, в котором команду $q_1\lambda \to q_z$R, надо заменить на команду $q_1 \sim\!\!\to q_z$R;

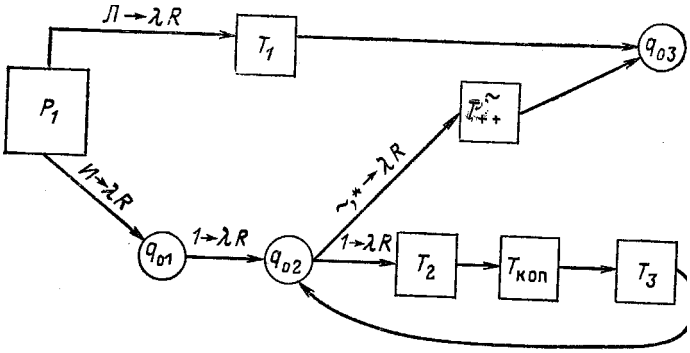


Рис. 7

*T*коп*(*а—1) раз копирует 1^b; после i-*го* цикла она останавливается в конфигурации $1^{(a-i-1)}(\sim 1^b)^{i-1} \sim q1^b * 1^b$;

*T*з — вернуть головку к крайнему слева непустому символу. После (а-1)-го цикла этим символом окажется ~(или *, если а=1), и происходит выход из цикла и переход к *T*~++.

T~++работает как *T*++ (см. пример 6, а) с той разницей, что числа, которые она суммирует, разделены двумя видами маркеров: ~ и *; для этой цели в нее к командам с маркером * в левой части добавлены такие же команды для маркера ~.

Аналогично тому, как из машины *T*+ была построена *T*++, можно из $T_\times$ построить машину $T_{\times\times}$, которая перемножает несколько чисел, и по схеме, аналогичной приведенной на рис. 7, с использованием $T\sim_{\times\times}$ вместо *T*~++ построить машину, осуществляющую возведение в степень. Другой важный пример, который читатель может либо построить, либо найти перевод унарного представления чисел в позиционное — двоичное или десятичное.

Универсальная машина Тьюринга

Построение универсальной машины *U*

Отметим, что при построении *U* (как, впрочем, и для многих других машин, описанных в этом параграфе) мы не преследовали целей оптимизации и не жалели ни символов ленты, ни состояний, стремясь к наглядности построения. Нетрудно показать, - а инженер-дискретчик, привыкший к двоичному кодированию, легко в это поверит - что можно построить машину *U* всего с двумя символами на ленте. **Шеннон установил менее очевидный факт — он построил универсальную машину с двумя состояниями. !В тоже время показано (Боброу и Минский), что универсальная машина с двумя состояниями и двумя символами невозможна.** Вообще в определенных пределах уменьшение числа символов *U* ведет к увеличению числа состояний, и наоборот, Подробная сводка результатов о минимальных универсальных машинах приведена в [Минский М. Вычисления и автоматы. М.: Мир, 1971].

Существование универсальной машины Тьюринга означает, что систему команд Σ_T любой машины *T* можно интерпретировать двояким образом: либо как описание работы конкретного устройства машины *T,* либо как программу для универсальной машины *U.* Для современного инженера, проектирующего систему управления, это обстоятельство вполне естественно. Он хорошо знает, что любой алгоритм управления может быть реализован либо аппаратурно – построением соответствующей схемы, либо программно – написанием программы для универсальной управляющей ЭВМ.

Однако важно сознавать, что сама идея универсального алгоритмического устройства совершенно не связана с развитием современных технических средств его реализации (электроники, физики твердого тела и т. д.) и является не техническим, а математическим фактом, описываемым в абстрактных математических терминах, не зависящих от технических средств, и к тому же опирающимся на крайне малое количество весьма элементарных исходных понятий.

Основополагающие работы по теории алгоритмов (и, в частности, работы Тьюринга) появились в 30-х годах, до создания современных ЭВМ.

Указанная двоякая интерпретация сохраняет на абстрактном уровне основные плюсы и минусы двух вариантов инженерной реализации. Конкретная машина *T* работает гораздо быстрее; кроме того, управляющее устройство машины *U* довольно громоздко (т.е. велико число состояний и команд). Однако его сложность постоянна, и, будучи раз построено, оно годится для реализации сколь угодно больших алгоритмов, понадобится лишь больший объем ленты, которую естественно считать более дешевой и более просто устроенной, чем управляющее устройство. Кроме того, при смене алгоритма не понадобится строить новых устройств; нужно лишь написать новую программу.

Тезис Тьюринга.

До сих пор нам удавалось для всех процедур, претендующих на алгоритмичность, т. е. конструктивных процедур, строить реализующие их машины Тьюринга. Будет ли это удаваться всегда? Утвердительный ответ на этот вопрос содержится в тезисе Тьюринга, который формулируется так: **«Всякий алгоритм может быть реализован машиной Тьюринга».**

Доказать тезис Тьюринга нельзя, поскольку само понятие алгоритма (или эффективной процедуры) является неточным. Это не теорема и не постулат математической теории, а утверждение, которое связывает теорию с теми объектами, для описания которых она создана. По своему характеру тезис Тьюринга напоминает гипотезы физики об адекватности математических моделей физическим явлениям и процессам. Подтверждением тезиса Тьюринга является, во-первых, математическая практика, а во-вторых, то обстоятельство (уже отмечавшееся в конце §1), что описание алгоритма в терминах любой другой известной алгоритмической модели может быть сведено к его описанию в виде машины Тьюринга.

Тезис Тьюринга позволяет, с одной стороны, заменить неточные утверждения о существовании эффективных процедур (алгоритмов) точными утверждениями о существовании машин Тьюринга, а с другой стороны, утверждения о несуществовании машин Тьюринга истолковывать как ут-

верждения о несуществовании алгоритмов вообще. Однако не следует понимать тезис Тьюринга в том смысле, что вся теория алгоритмов может быть сведена к теории машин Тьюринга. Например, в быстро развивающейся сейчас теории сложности алгоритмов (которая рассматривает сравнительную сложность алгоритмов по памяти, числу действий и т. д.) результаты, верные в рамках одной алгоритмической модели (скажем, о числе действий, необходимых для вычисления данной функции), могут оказаться неверными в другой модели.

Проблема остановки.

В числе общих требований, предъявляемых к алгоритмам (см. §1), упоминалось требование результативности. Наиболее радикальной формулировкой здесь было бы требование, чтобы по любому алгоритму A и данным α можно было определить, приведет ли работа A при исходных данных α к результату или нет. Иначе говоря, нужно построить алгоритм B, такой, что $B(A, \alpha)$ =И, если A (α) дает результат, и $B(A, \alpha)$=Л, если $A(\alpha)$ не дает результата. В силу тезиса Тьюринга эту задачу можно сформулировать как задачу о построении машины Тьюринга; построить машину T_0, такую, что для любой машины Тьюринга T и любых исходных данных α для машины T $T_0(\Sigma_T, \alpha)$==И, если машина $T(\alpha)$ останавливается, и $T_0(\Sigma_T, \alpha)$ ==Л, если машина $T(\alpha)$ не останавливается.

Эта задача *называется проблемой остановки*. Ее формулировка несколько напоминает задачу о построении универсальной машины и, в частности, также предполагает выбор подходящего кодирования Σ_T и α в алфавите машины T_0. Однако в данном случае никакое кодирование не приводит к успеху.

Теорема (проблема остановки) Не существует машины Тьюринга T_0, решающей проблему остановки для произвольной машины Тьюринга T.

Предположим, что машина T_0 существует. Для определенности будем считать, что маркером между Σ_T и α на ленте машины T_0 служит *. Построим машину $T_1(\Sigma_T)=T_0$ ($Tкоп(\Sigma т)$). Исходными данными машины T_1 являются системы команд (точнее, их коды) любой машины T. Запись Σ_T на ленте машина T_1 преобразует в $\Sigma_T*\Sigma_T$ (машина Ткоп для чисел описана в примере 4, б), а затем работает как машина T_0. Таким образом, T_1 также решает проблему остановки для любой машины T, но только в том случае, когда на ленте T в качестве данных α_T написана ее собственная система команд Σ_T. Иначе говоря, $T_1(\Sigma_T)$=И, если машина $T(\Sigma_T)$ останавливается, и $T_1(\Sigma_T)$=*Л*, если машина $T(\Sigma_T)$ не останавливается. Пусть q_{1n} — заключительное состояние T_1. Добавим к системе команд T_1 одно состояние $q_{1,n+1}$, объявив его

заключительным, и m команд *(m*—число символов T_1*)* $q_{1n}Л \to q_{1,n+1}E$, $q_{1n}a_j \to q_{1,n}R$ для любого a_j, (в том числе И), кроме Л. Получим машину $T_1'(\Sigma_T)$, которая останавливается, если T не останавливается, и не останавливается, если T останавливается. Запишем теперь на ленте машины T_1' ее собственную систему команд $\Sigma_{T'1}$. Тогда T_1' остановится, если она не останавливается, и не остановится, если она останавливается. Очевидно, такая машина T_1' невозможна. Но поскольку она получена из T_0 вполне конструктивными, не вызывающими сомнений средствами и при этом никак не связана с конкретной структурой машины T_0, остается заключить, что никакая машина T_0, решающая проблему остановки, невозможна.

В силу тезиса Тьюринга невозможность построения машины Тьюринга означает отсутствие алгоритма решения данной проблемы. Поэтому полученная теорема дает первый пример *алгоритмически неразрешимой проблемы,* а именно, алгоритмически неразрешимой оказывается проблема остановки для машин Тьюринга, т. е. проблема определения результативности алгоритмов.

При истолковании утверждений, связанных с алгоритмической неразрешимостью, следует иметь в виду следующее важное обстоятельство. В таких утверждениях речь идет об отсутствии единого алгоритма, решающего данную проблему; при этом вовсе не исключается возможность решения этой проблемы в частных случаях, но различными средствами для каждого случая. В частности, теорема (проблема остановки) не исключает того, что для отдельных классов машин Тьюринга проблема остановки может быть решена'. Например, она решается для всех машин, приведенных в примерах 2 - 8. Поэтому неразрешимость общей проблемы остановки вовсе не снимает необходимости доказывать сходимость предлагаемых алгоритмов, а лишь показывает, что поиск таких доказательств нельзя полностью автоматизировать.

Неразрешимость проблемы остановки можно интерпретировать как несуществование общего алгоритма для отладки программ, точнее, алгоритма, который по тексту любой программы и данным для нее определял бы, зациклится ли программа на этих данных или нет. Если учесть сделанное ранее замечание, такая интерпретация не противоречит тому эмпирическому факту, что большинство программ, в конце концов, удается отладить, т. е. установить наличие зацикливания, найти его причину и устранить ее. При этом решающую роль играют опыт и искусство программиста.

Лекция 15

РЕКУРСИВНЫЕ ФУНКЦИИ

Всякий алгоритм однозначно ставит в соответствие исходным данным (в случае, если он определен на них) результат. Поэтому с каждым алгоритмом однозначно связана функция, которую он вычисляет. Верно ли обратное: для всякой ли функции существует вычисляющий ее алгоритм? Исследование проблемы остановки для машин Тьюринга показывает, что нет: для предиката $P(T, \alpha)$, истинного, если и только если машина Тьюринга T останавливается при исходных данных α, алгоритма его вычисления не существует. Возникает вопрос: для каких функций алгоритмы существуют? Как описать такие алгоритмические, эффективно вычислимые функции?

Исследование этих вопросов привело к созданию в 30-х годах нашего века *теории рекурсивных функций*. В этой теории, как и вообще в теории алгоритмов, принят конструктивный, финитный подход, основной чертой которого является то, что все множество исследуемых объектов (в данном случае функций) строится из конечного числа исходных объектов базиса – с помощью простых операций, эффективная выполнимость которых достаточно очевидна. Операции над функциями в дальнейшем будем называть операторами.

Примитивно-рекурсивные функции. Определение и примеры. Займемся теперь конкретным выбором средств, с помощью которых будут строиться вычислимые функции. Очевидно, что к вычислимым функциям следует отнести все константы, т. е. 0 и все натуральные числа 1, 2... Однако в прямом включении бесконечного множества констант в базис нет необходимости.

Опр. 1. Достаточно одной константы 0 и *функции следования* $f(x)=x+1$, которую иногда обозначают x', чтобы получить весь натуральный ряд.

Опр. 2. Кроме того, в базис включим семейство $\{I^n_m\}$ *функций тождества* (или введения фиктивных переменных):

$$I^n_m(x_1,\dots x_n) = x_m \ (m \le n).$$

Весьма мощным средством получения новых функций из уже имеющихся является *суперпозиция*, знакомая нам. *Суперпозицией* называлась любая подстановка функций в функции. Здесь ей для большей обозримости удобно придать стандартный вид.

Опр. 3. *Оператором суперпозиции* S^n_m называется подстановка в функцию от m переменных m функций от n одних и тех же переменных. Она дает новую функцию от n переменных. Например, для $h(x_1,\dots x_m)$, $g_1(x_1,\dots x_n)$, …, $g_m(x_1,\dots x_n)$

$$S^n_m(h, g_1,\dots g_m) = h(g_1(x_1,\dots x_n), \dots, g_m(x_1,\dots x_n)) = f(x_1,\dots x_n).$$

Это определение порождает семейство операторов суперпозиции $\{S^n_m\}$. Благодаря функциям тождества стандартизация суперпозиции не уменьшает ее возможностей: любую подстановку функций в функции можно выразить через S^n_m и I^n_m.

Например, $f(x_1,x_2) = h(g_1(x_1,x_2), g_2(x_1))$ в стандартном виде запишется как $f(x_1,x_2) = S^2_2\ (h(x_1,x_2), g_1(x_1,x_2), S^1_2(I^2_1(x_1,x_2)), g_2(x_1), g_3(x_1))$, где g_3 — любая функция от x_1. В свою очередь, используя подстановку и функции тождества, можно переставлять и отождествлять аргументы в функции:

$$f(x_2, x_1, x_3, \ldots, x_n) = f(I^2_2(x_1,x_2), I^2_1(x_1,x_2), x_3, \ldots, x_n);$$

$$f(x_1, x_1, x_3, \ldots, x_n) = f(I^2_1(x_1,x_2), I^2_1(x_1,x_2), x_3, \ldots, x_n).$$

Таким образом, если заданы функции I^n_m и операторы S^n_m, то можно считать заданными всевозможные операторы подстановки функций в функции, а также переименования, перестановки и отождествления переменных.

Еще одно семейство операторов, которое здесь понадобится, – это операторы примитивной рекурсии.

Опр. 4. *Оператор примитивной рекурсии* R_n определяет (n+1)-местную функцию f через n-местную функцию g и (n+2)-местную функцию h следующим образом:

$$f(x_1, \ldots, x_n, 0) = g(x_1, \ldots, x_n);$$

$$f(x_1, \ldots, x_n, y+1) = h(x_1, \ldots, x_n, y, f(x_1, \ldots, x_n, y)). \qquad (1)$$

Пара равенств (1) называется *схемой примитивной рекурсии.* Тот факт, что функция f определена схемой (1), выражается равенством $f(x_1, \ldots, x_n, y) = R_n(g, h)$. В случае, когда n=0, т. е. определяемая функция f является одноместной, схема (1) принимает более простой вид:

$$f(0) = c;\ f(y+1) = h(y, f(y)). \qquad (2)$$

где с — константа.

Схемы (1) и (2) определяют f рекурсивно не только через другие функции g и h, но и через значения f в предшествующих точках: значение f в точке $y+1$ зависит от значения f в точке y. Для вычисления $f(x_1, \ldots, x_n, k)$ понадобится $k+1$ вычислений по схеме (1) - для $y=0,1,\ldots,k$.

Пример такого определения функции приводился для функции $n!$; здесь оно будет рассмотрено более подробно. Существенным в операторе примитивной рекурсии является то, что независимо от числа переменных в f рекурсия ведется только по одной переменной y; остальные n переменных $x_1, \ldots, x_n$ на момент применения схем (1), (2) зафиксированы и играют роль параметров.

Опр. 5. Функция называется *примитивно-рекурсивной,* если она может быть получена из константы 0, функции x' и функций I^n_m с помощью конечного числа применений операторов суперпозиции и примитивной рекурсии. Этому определению можно придать более формальный индуктивный вид.

1. Функции 0, x' и I^n_m для всех натуральных n,m, где $m \le n$, являются примитивно-рекурсивными.

2. Если $h(x_1,\dots x_m)$, $g_1(x_1,\dots x_n)$, …, $g_m(x_1,\dots x_n)$ - примитивно-рекурсивные функции, то $S^n_m(h, g_1,\dots g_m)$ - примитивно-рекурсивные функции для любых натуральных *n, m.*
3. Если $g_1(x_1,\dots x_n)$ и $h(x_1,\dots x_n, y, z)$ — примитивно-рекурсивные функции, то

   $R_n(g, h)$ - примитивно-рекурсивная функция.
4. Других примитивно-рекурсивных функции нет. Из такого индуктивного описания нетрудно извлечь процедуру, порождающую все примитивно-рекурсивные функции.

Пример 1. (сложение, умножение, возведение в степень).

1. Сложение $f_+(x, y)=x+y$ примитивно-рекурсивно:

$$f_+(x, 0) = x = I^1_1(x);$$

$$f_+(x, y+1) = f_+(x, y) +1 = (f_+(x, y))'.$$

Таким образом, $f_+(x, y) = R_1(I^1_1(x), h(x, y, z))$, где $h(x, y, z) = z' = z +1$.

2. Умножение $f_\times(x, y)=xy$ примитивно-рекурсивно:

$$f_\times(x, 0)=0;$$

$$f_\times(x, y+1)= f_\times(x, y) + x= f_+(x, f_\times(x, y)) .$$

3. Возведение в степень $f_{exp}(x, y) = x^y$ примитивно-рекурсивно:

$$f_{exp}(x, 0) = 1;$$

$$f_{exp}(x, y+1) = x^y\, x = f_\times(x, f_{exp}(x, y)).$$

Определим функцию x ÷ y (арифметическое или урезанное вычитание) следующим образом:

Пример 2. (вычитание). Примитивно-рекурсивными являются следующие функции:

1) $f(x)$=x÷1, определяемая схемой:

$f(0)=0\div1=0;$

$f(y+1)=y;$

2) f (x, y) = x ÷ y, *определяемая схемой:*

f(x,0)=x;

f(x,y+1)= x÷ (y+1)=(x÷y) ÷1=f(x, y) ÷1

(для определения функции из п.2 использована функция из п.1);

3) f(x, y)=|x-y|=(x÷y)+(y÷x);

4) функция знака (сигнум)

0, *если x*=0;

$sg(x) = 1$, *если* $x \neq 0$;

ее схема имеет вид:

$sg(0) = 0$;

$sg(x+1) = 1$;

5) $min(x, y) = x \div (x \div y)$;

6) $max(x, y) = y + (x \div y)$.

С помощью функции sg (сигнум) из примера 2 и ее отрицания $\neg sg(x) = 1 \div sg(x)$ построим примитивно-рекурсивное описание функций, связанных с делением.

Пример 3. (деление)

а. Функция $r(x, y)$ - остаток от деления y на x:

$$r(x, 0) = 0;$$

$$r(x, y+1) = (r(x, y) + 1)\, sg(|x - (r(x, y) + 1)|).$$

Смысл второй строки определения в следующем: если $y+1$ не делится на x, то $sg(|x \div (r(x, y) + 1)|) = 1$ и $r(x, y+1) = r(x, y)+1$; если же $y+1$ делится на x, то $r(x, y+1) = sg(|x - (r(x, y) + 1)|) = 0$.

б. Функция $q(x, y) = [y/x]$ — частное от деления y на ***x,*** т, е. целая часть дроби y/x:

$$q(x, 0) = 0;$$

$$q(x, y+1) = q(x, y) + \neg sg(|x - (r(x, y) + 1)|).$$

Второе слагаемое, как и в случае $r(x, y)$, зависит от делимости $y+1$ на x. Если $y+1$ делится на x, то $q(x, y+1)$ на единицу больше, чем $q(x, y)$; если нет, то $q(x,y+1) = q(x,y)$.

В рекурсивных описаниях функций примера 3 отчетливо видны логические действия: проверка условия (делимости $y+1$ на x) и выбор дальнейшего хода вычислений в зависимости от истинности условия. Займемся теперь более подробным выяснением того, какие возможности для логических операций имеются в рамках примитивно-рекурсивных функций.

Из функций примера 2 легко получается примитивная рекурсивность «арифметизованных» логических функций, т. е. числовых функций, которые на множестве $\{0,1\}$ ведут себя как логические функции. Действительно, если $x, y \in \{0, 1\}$, то

$$\neg x = 1 \div x;$$

$$x \vee y = \max(x,y);$$

$$x \,\&\, y = \min(x,y).$$

Из функциональной полноты (см. §) этого множества функций и того, что суперпозиция является примитивно-рекурсивным оператором (см. далее), следует примитивная рекурсивность всех логических функций.

Отношение $R(x_1, ..., x_n)$ называется примитивно-рекурсивным, если примитивно-рекурсивна его характеристическая функция χ_R:

$$\chi_R(x_1, ..., x_n) = \begin{cases} 1, \text{ если } R(x_1, ..., x_n) \text{ выполняется;} \\ 0 \text{ в противном случае.} \end{cases}$$

Ввиду взаимно однозначного соответствия между отношениями и предикатами χ_R будет характеристической функцией и для соответствующего предиката.

Напомним, что сам предикат принимает логические значения «И» и «Л», с которыми нельзя производить арифметических действий, даже когда эти значения изображаются числами 0 и 1 (см. §). Поэтому следует проводить различие между предикатами и их характеристическими функциями. В алгоритмических языках типа АЛГОЛ предикат будет принимать значения true и false, а его характеристическая функция – значения 0 и 1.

Опр.5. Предикат называется *примитивно-рекурсивным*, если его характеристическая функция примитивно-рекурсивна. В силу соотношений (3), если предикаты P_1 .., P_k примитивно-рекурсивны, то и любой предикат, полученный из них с помощью логических операций, примитивно-рекурсивен.

Пример 4. (предикаты и отношения)

а. Предикат $Pd_n(x)$ «x делится на n» примитивно-рекурсивен для любого n:

$$\chi_{Pdn}(x) = \overline{sg}(r(n,x)).$$

б. Предикат $Pd_{n,m}(x)$ «x делится на m и на n» примитивно-рекурсивен для любых m, n, так как $P_{n,m}(x) = P_m(x)$ & $P_n(x)$.

в. Отношение $x_1 > x_2$ примитивно-рекурсивно:

$$\chi_{>}(x_1, x_2) = sg(x_1 \dot{-} x_2).$$

г. Если $f(x)$ и $g(x)$ примитивно-рекурсивны, то предикат «$f(x)=g(x)$» примитивно-рекурсивен, так как его функция χ имеет вид:

$$\chi(x) = \overline{sg}(|f(x) - g(x)|).$$

Лекция 16

Примитивно-рекурсивные операторы.

Оператор называется примитивно-рекурсивным (сокращенно ПР-оператором), если он сохраняет примитивную рекурсивность функций, т. е., если результат его применения к примитивно-рекурсивным функциям дает снова примитивно-рекурсивную функцию. Операторы S^n_m и R_n являются ПР-операторами по определению. Рассмотрим другие ПР-операторы. Их использование позволяет существенно сократить примитивно-рекурсивные описания функций.

Мы встречались с оператором условного перехода (обозначим его здесь *B),* который по функциям $g_1(x_1, \ldots x_n)$, $g_2(x_1, \ldots x_n)$ и предикату $P(x_1,\ldots x_n)$ строит функцию $f(x_1, \ldots, x_n) = B(g_1, g_2,,P)$:

$$f(x_1,\ldots,x_n) = \begin{cases} g_1(x_1, \ldots, x_n), & \text{if } P(x_1, \ldots, x_n) \text{ истинно;} \\ g_2(x_1, \ldots, x_n), & \text{if } P(x_1, \ldots, x_n) \text{ ложно.} \end{cases} \quad (1)$$

Теорема 1. *B* вычислим по Тьюрингу.

Примитивная рекурсивность *B* видна из следующего соотношения, эквивалентного (1):

$$f(x_1, \ldots, x_n) = g_1(x_1, \ldots, x_n)\chi_P(x_1, \ldots, x_n) + g_2(x_1, \ldots, x_n)\chi_{|P}(x_1, \ldots, x_n)$$

Обобщение оператора *B* на случай многозначного перехода по предикатам $P_1,\ldots,P_k$, из которых истинен всегда один и только один предикат: $f(x_1, \ldots, x_n) = B(g_1, \ldots, g_k\ P_1,\ldots,P_k)$, также примитивно-рекурсивно, поскольку

$$f(x_1, \ldots, x_n) = g_1\chi_{P1} + \ldots + g_k\chi_{Pk}$$

(отметим, что это соотношение определяет *B* и в том случае, когда ни один из $P_1,\ldots,P_k$ не истинен: *B* при этом равно нулю).

С помощью оператора *B* удобно задавать функции, определенные на конечных множествах. Правда, *B*, как и любой другой ПР-оператор, всегда определяет функцию на всем натуральном ряде, т. е. производит доопределение функции вне области задания. **Например**, функцию *f,* определенную на множестве {1, 2, 6, 17} равенствами $f(1)=5$, $f(2)=1$, $f(6)==0$, $f(17)=8$, с помощью оператора *B* можно описать так:

$$f(x) = \begin{cases} 5, & \text{if } x=1; \\ 1, & \text{if } x=2; \\ 0, & \text{if } x=6; \\ 8, & \text{в остальных случаях.} \end{cases}$$

Такое описание полагает $f(x)$=8 вне исходной области задания.

Пусть f(x_1, ..., x_n, y)—функция от (n+1)-й переменной. Хорошо известные операции суммирования $\Sigma_{y<z}$ и перемножения $\Pi_{y<z}$ по переменной y с пределом z — это операторы, которые из функции f(x_1, ..., x_n, y) порождают новые функции $g(x_1, ..., x_n, z)= \Sigma_{y<z} f(x_1, ..., x_n, y)$ и $h(x_1, ..., x_n, z)= \Pi_{y<z} f(x_1, ..., x_n, y)$. Покажем их примитивную рекурсивность:

$g(x_1, ..., x_n, 0)=0$ (по определению);

$g(x_1, ..., x_n, z+1) = g(x_1, ..., x_n, z) + f(x_1, ..., x_n, z)$;

$h(x_1, ..., x_n, 0)=1$;

$h(x_1, ..., x_n, z+1) = h(x_1, ..., x_n, z)\, f(x_1, ..., x_n, z)$.

Таким образом, g и h могут быть определены по схеме "примитивной рекурсии с использованием сложения и умножения, примитивная рекурсивность которых доказана, а также функции f. Следовательно, g и h примитивно-рекурсивны, если f примитивно-рекурсивна.

С помощью $\Sigma_{y<z}$ и $\Pi_{y<z}$ нетрудно показать, что операторы $\Sigma^z_{y=k}$ и $\Pi^z_{y=k}$ и (т. е. суммирование и умножение от k до z) — так же ПР-операторы.

Рассмотрим теперь оператор, играющий важную роль в теории рекурсивных функций, — *ограниченный оператор наименьшего числа (μ-оператор), называемый также ограниченным оператором минимизации,* который применяется к предикатам и определяется как:

$$\mu\, y_{y<=z}\, P(x_1, ..., x_n\, y) = \begin{cases} \text{наименьшему } y<=z, \text{ такому, что } P(x_1, ..., x_n, y)=\text{И}, \\ \text{если такой } y \text{ существует;} \\ z \text{ в противном случае.} \end{cases}$$

Из предиката $P(x_1, ..., x_n\ y)$ с помощью оператора $\mu y_{y<=z}$ получается функция $f(x_1, ..., x_n, z)$. Второй случай в определении μ добавлен для того, чтобы f была всюду определена.

Пример 1. Пусть $P(x_1,x_2, y)=Pd_{x1,x2}(y)$ (пример). Тогда $\mu\, y_{y<=z}\, P(x_1,x_2, y)$ равен наименьшему общему кратному (НОК) x_1 и x_2, если z>=НОК, и равен z, если z<= НОК.

Ограниченный μ-оператор примитивно-рекурсивен:

$$\mu\, y_{y<=z}\, P(x_1, ..., x_n\, y) = \Sigma^z_{i=0} \Pi^i_{j=0}(1- \chi_P(x_1, ..., x_n, j)).$$

Ограничение z в ограниченном μ-операторе дает гарантию окончания вычислений, поскольку оно оценивает сверху число вычислений предиката P. Возможность оценить сверху количество вычислений является существенной особенностью примитивно-рекурсивных функций. Уже отмечалось, что если $f(x_1,..., x_n, k) = R_n(g, h)$, то для вычисления $f(x_1,..., x_n, k)$ понадобится k+ 1 вычислений по схеме (?5.2); одно вычисление g и k вычислений h. Правда, каждое из них может, в свою очередь, состоять из некоторого количества вычислений функций, входящих в определение g и *h;*

но в силу конечности общего числа операторов S^n_m и R_n, использованных для построения f из базисных функций 0, $x,'$ и 1^n_m, для любого k можно оценить количество элементарных действий (т. е. вычислений базисных функций), необходимых для вычисления $f(x_1,..., x_n, k)$. В дальнейшем будет показано, что неограниченный μ-оператор не является примитивно-рекурсивным.

μ-оператор (как ограниченный, так и неограниченный) является удобным средством для построения обратных функций. Действительно, функция $g(x)= \mu\, y(f\{y)=x)$ («наименьший y, такой, что $f(y)=x$») является обратной к функции $f(x)$. Поэтому в применении к одноместным функциям μ-оператор иногда называют оператором обращения.

Пример 2.

а. С помощью ограниченного μ-оператора определим деление, точнее функцию $[z/x]$ (частное от деления z на x;—см. пример), как функцию, обратную умножению:

$[z/x]= \mu\, y_{y<=z}\ (x(y+1)>z)$.

Целая часть от деления — функция, «не совсем обратная» умножению; точнее, обратная ему только в тех точках, где z делится нацело на x. Поэтому в ее описании используется не предикат равенства (как в определении обратной функции), а предикат >.

б. Целая часть $\sqrt{x}$ — функция $[\sqrt{x}]$— примитивно-рекурсивная, так как

$[\sqrt{x}]= \mu\, y_{y<=x}\ ((y+1)^2>x)$.

в. $[\log_k x] = \mu\, y_{y<=x}\ ((k^{y+1}>x)$, следовательно, целая часть логарифма по любому целому основанию k примитивно-рекурсивна.

Еще один ПР-оператор — это оператор совместной или одновременной рекурсии, точнее, целое семейство операторов $\{R_{nk}\}$. С помощью R_{nk} строится рекурсивное описание сразу нескольких функций f_1, ..., f_k от (n+1)-й переменной, причем значение каждой функции $y+1$ зависит от значения всех функций в точке y:

$$f_1(x_1, \ldots, x_n, 0) = g_1(x_1, \ldots, x_n);$$

$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

$$f_k(x_1, \ldots, x_n, 0) = g_k(x_1, \ldots, x_n);$$

$$f_1(x_1, \ldots, x_n, y+1) = h_1(x_1, \ldots, x_n, y, f_1(x_1, \ldots, x_n, y), \ldots, f_k(x_1, \ldots, x_n, y));$$

$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$

$$f_k(x_1, \ldots, x_n, y+1) = h_k(x_1, \ldots, x_n, y, f_1(x_1, \ldots, x_n, y), \ldots, f_k(x_1, \ldots, x_n, y)).$$

По существу совместная рекурсия дает рекурсивное описание функции-вектора $(f_1, ...,f_k)$. Для того чтобы доказать примитивную рекурсивность совместной рекурсии, нужно закодировать этот вектор числом, причем так, чтобы существовала однозначная и примитивно-

рекурсивная расшифровка этого кода (т. е. извлечение нужного разряда вектора). В качестве такого кода можно предложить число

$$\varphi(x_1, \ldots, x_n, y) = 2^{f1(x1, \ldots, xn, y)} \bullet 3^{f2} \bullet 5^{f3} \bullet \ldots \bullet p_{k-1}^{fk(x1, \ldots, xn, y)},$$

где p_i – i-е простое число (2 считается 0-м простым числом). Продемонстрируем эту идею на примере оператора R_{12}.

Пусть $f_1(x, y), f_2(x, y)$ заданы совместной рекурсией:

$f_1(x, 0) = g_1(x)$;

$f_2(x, 0) = g_2(x)$;

$f_1(x, y+1) = h_1(x, y, f_1(x, y), f_2(x,y))$;

$f_2(x, y+1) = h_2(x, y, f_1(x, y), f_2(x,y))$.

Определим кодирующую функцию F:

$F(x,0) = 2^{g1(x)} \bullet 3^{g2(x)}$;

$F(x,y+1) = 2^{h1(x, y, f1(x,y),f2(x,y))} \bullet 3^{h2(x, y, f1(x,y), f2(x,y))}$.

Тогда $f_1(x, y) = \mu z_{z<=F(x, y)}(\neg Pd._2{}^{z+1}(F(x, y))$ т. е. равна показателю при 2 в разложении $F(x, y)$ на простые множители; $f_2(x, y) = \mu z_{z<=F(x, y)}(Pd._3{}^{z+1}(F(x, y))$, т.е. равна показателю при 3 в разложении $F(x, y)$ на простые множители. (Определение предиката Pd см. в примере _).

Следует иметь в виду, что такая кодировка вовсе не предлагается в качестве метода рекурсивного вычисления функций-векторов. Наоборот, предлагается прямой и более простой метод совместной рекурсии, а кодировка является лишь доказательством того, что этот метод относится к числу примитивно-рекурсивных средств вычисления.

Можно предложить полезную схемную интерпретацию примитивной рекурсии вообще и совместной рекурсии в частности. На рис. 1, и изображена схема, состоящая из устройства, вычисляющего за один такт функцию h от двух переменных, и элемента задержки на один такт; по каналам схемы могут передаваться натуральные числа. Время t будем считать дискретным: t=0, I, 2... Схема имеет один вход x и выход f. Однако выход f зависит не только от значения x, но и от момента t, в который он рассматривается, т. е. представляет собой некоторую функцию $f(x, t)$. Рассмотрим эту зависимость подробнее. Будем считать значение x на все время рассмотрения, начиная с t=0, произвольным, но фиксированным. В начальный момент t=0 значение второго входа h является константой с, зависящей от начального состояния схемы: $f(x, 0)=h(x, c)=g(x)$. В момент t=1 $f(x, 1)=h(x, f(x, 0))$; в общем случае $f(x, t+1)=h(x, f(x, t))$. Таким образом, схема рис. 1,*а* реализует примитивную рекурсию по переменной t, т.е. по времени. Нетрудно убедиться, например, что если h выполняет умножение, а с=1, то $f(x, t)=x^{t+1}$, При аналогичной интерпретации (с начальными константами c_1, c_2;) схема на рис.1 реализует схему совместной рекурсии, причем рекурсия также ведется по времени, общему для устройств h_1 и h_2. Доказательство

того, что совместная рекурсия – это ПР-оператор, в терминах таких схем означает, что перекрестные обратные связи на рис.1 можно заменить простыми контурами

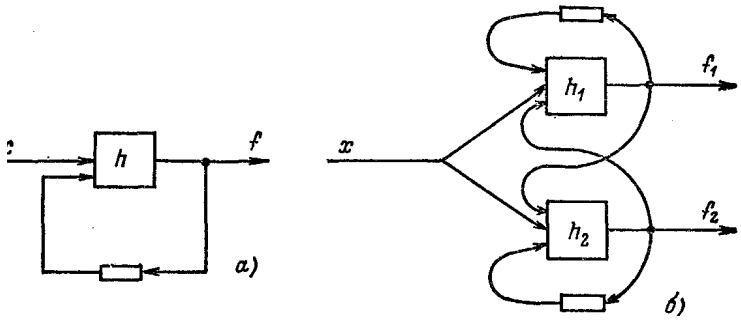


Рис. 1

обратной связи на рис. 1,*б* можно заменить простыми контурами обратной связи типа приведенных на рис. 1, *а*. Структура схемы при этом станет проще, однако понадобятся дополнительные вычислительные устройства, формирователь числового кода вектора (fi, f_2), передаваемого по одному каналу, и декодирующее устройство с двумя выходами fi и f_2.

Описанная схемная интерпретация примитивно-рекурсивных функций проходит при одном предположении, что за один такт любой канал схемы способен передать, а вычислительное устройство — воспринять и переработать любое, сколь угодно большое натуральное число. Такое предположение физически нереализуемо, поэтому теория таких схем не будет представлять самостоятельного интереса. Если же наложить ограничения на возможности каналов и элементов схем, то это приведет к теории конечных автоматов и схем из автоматов.

Подведем некоторые итоги.

Из простейших функций – константы 0, функции $x+1$ и функций тождества I^n_m – с помощью операторов суперпозиции и примитивной рекурсии было получено огромное разнообразие функций, включающих основные функции арифметики, алгебры и анализа (с поправкой на целочисленность). Тем самым выяснено, что эти функции имеют примитивно-рекурсивное описание, которое однозначно определяет процедуру их вычисления; следовательно, их естественно отнести к классу вычислимых функций.

Сделаем два замечания.

Во-первых, все примитивно-рекурсивные функции всюду определены. Это следует из того, что простейшие функции всюду определены, а операторы S^n_m и R_n это свойство сохраняют.

Во-вторых, строго говоря, мы имеем дело не с функциями, а с их примитивно-рекурсивными описаниями. Различие здесь имеет тот же смысл, что и отмечавшееся неоднократно в гл. 3 различие между функциями и их

представлениями в виде формул. Примитивно-рекурсивные описания также можно разбить на классы эквивалентности, отнеся в один класс все описания, задающие одну и ту же функцию.

Однако задача распознавания эквивалентности примитивно-рекурсивных описаний, как будет показано, алгоритмически неразрешима.

Лекция 17

Функции Аккермана. Пора уже задать вопрос: все ли функции являются примитивно-рекурсивными? Простые теоретико-множественные соображения показывают, что нет. Было показано, что множество всех функций типа $\boldsymbol{N} \to \boldsymbol{N}$ (т. е. одноместных целочисленных функций) несчетно, тем более это верно для функций типа $\boldsymbol{N}^{\mathbf{n}} \to \boldsymbol{N}$. Каждая примитивно-рекурсивная функция имеет конечное описание, т. е. задается конечным словом в некотором (фиксированном для всех функций) алфавите. Множество всех конечных слов счетно, поэтому примитивно-рекурсивные функции образуют не более чем счетное подмножество несчетного множества функций типа $\boldsymbol{N}^{\mathbf{n}} \to \boldsymbol{N}$. В действительности рассматривается более узкий вопрос: *все ли вычислимые функции можно описать как примитивно-рекурсивные?* Чтобы показать, что ответ на этот вопрос также является отрицательным, построим пример вычислимой функции, не являющейся примитивно-рекурсивной. Идея примера в том, чтобы, построив последовательность функций, каждая из которых растет существенно быстрее предыдущей, сконструировать с ее помощью функцию, которая растет быстрее любой примитивно-рекурсивной функции.

Начнем с построения последовательности функций φ_0, φ_1, φ_2. Положим $\varphi_0(a, y)=a+y$; $\varphi_1(a, y)=ay$, $\varphi_2(a, y)=a^y$. Эти функции связаны между собой следующими рекурсивными соотношениями:

$$\varphi_1(a, y+1) = a+ay = \varphi_0(a, \varphi_1(a, y)); \ \varphi_1(a, 1) = a;$$

$$\varphi_2(a, y+1) = aa^y = \varphi_1(a, \varphi_2(a, y)); \ \varphi_2(a, 1) = a.$$

Продолжим эту последовательность, положив по определению

$$\varphi_{n+1}(a, 0) = 1;$$

$$\varphi_{n+1}(a,1)=a; \qquad (1)$$

$$\varphi_{n+1}(a, y+1)\ \varphi_n = (a, \varphi_{n+1}(a, y)).$$

Эта схема имеет вид примитивной рекурсии, следовательно, все функции φ_n примитивно-рекурсивные. Растут они крайне быстро; например, $\varphi_3(a,1)=a$; $\varphi_3(a,2)=\varphi_2(a,a) =a^a$; $\varphi_3(a,3)=a^{(a)a}; \ldots; \varphi_3(a,k) =a^{(a)\ldots a\}k\ раз}$.

Зафиксируем теперь значение a: a=2.

Получим последовательность одноместных функций: $\varphi_0(2,y)$, $\varphi_1(2, y)$... Определим те-перь функцию $B(x, y)$, которая перечисляет эту последовательность: $B(x, y) = \varphi_x(2,y)$, т.е. й(0, $B(0, y) = \varphi_0(2,y) = 2+y$; $B(1, y) = \varphi_1(2,y) = 2y$ т.д., а также диагональную функцию $A(x)=B(x,x)$. Эти функции называются *функциями Аккермана*. Из соотношений (1) следует, что

$B(0, y) = 2+y;$

$B(x+1,y)=sgx;$ (2)

$B(x+1, y+1) = B(x, B(x+1, y))$.

Эти соотношения позволяют вычислять значения функций $B(x, y)$ и, следовательно, $A(x)$, причем для вычисления функции в данной точке нужно обратиться к значению функции в предшествующей точке — совсем как в схеме примитивной рекурсии. Однако здесь рекурсия ведется сразу по двум переменным (такая рекурсия называется двойной, двукратной, или рекурсией 2-й ступени), и это существенно, усложняет характер упорядочения точек, а следовательно, и понятие предшествования точек также усложняется. Например, $B(3, 3) = B(2, B(3, 2))$, а так как $B(3, 2) = \varphi_3(2, 2)= 2^2 = 4$, то вычислению B и в точке (3,3) по схеме (2) должно предшествовать вычисление B в точке (2,4); читатель может убедиться, что вычислению B в точке (3,4) должно предшествовать вычисление в точке (2,16). Важно отметить, что это упорядочение не предопределено заранее, как в схеме примитивной рекурсии, где n всегда предшествует $n+1$, а выясняется в ходе вычислений и для каждой схемы вида (2), вообще говоря, различно. Поэтому схему двойной рекурсии (в отличие от рассмотренной ранее одновременной рекурсии) не всегда удается свести к схеме примитивной рекурсии.

Теорема. Функция Аккермана A(x) растет быстрее, чем любая примитивно-рекурсивная функция, и, следовательно, не является примитивно-рекурсивной. Более точно, для любой одноместной примитивно-рекурсивной функции $f(x)$ найдется такое n, что для любого x>=n $A(x)>f(x)$.

Ограничимся наброском доказательства, опустив некоторые выкладки.

1. Вначале доказываются два свойства функции $B(x, y)$;

$B(x, y+1)>B(x, y)$ $(x, y=1, 2 ...)$; (3)

$B(x + 1, y) >= B(x, y+1)$ $(x >= 1, y >= 2)$. (4)

2. Назовем функцию $f(x_1,.., x_n)$ B-мажорируемой, если существует натуральное m от, такое, что для любых $x_1,.., x_n$ удовлетворяющих условию max $(x_1,.., x_n) >1$, $f(x_1,.., x_n) < B(m, \max (x_1,.., x_n))$. Покажем, что все примитивно-рекурсивные функции B-мажорируемы:

а) для простейших функций 0, x+1, I^n_m их B-мажорируемость очевидна.

б) рассмотрим оператор суперпозиции S^n_m, причем для простоты выкладок ограничимся случаем n=1, m=2, т.е. функцией $f(x) = h(g_1(x), g_2(x))$.

Тезис Черча. Связь рекурсивных функций с машиной Тьюринга

Определение 1 Функция называется *частично-рекурсивной*, если она может быть построена из простейших функций 0, x+1, I^n_m с помощью конечного числа применений оператора суперпозиции S^n_m, примитивной рекурсии R_n и *μ-оператора.*

Тезис Черча является аналогом тезиса Тьюринга для рекурсивных функций**: *всякая функция, вычислимая некоторым алгоритмом, частично-рекурсивна.***

Утверждение: функция вычислима машиной Тьюринга, тогда и только тогда, когда она частично-рекурсивна.

Это утверждение об эквивалентности двух алгоритмических моделей является (в отличие от тезисов) вполне точным математическим утверждением и должно быть доказано.

Разобьем утверждение на две теоремы.

Теорема 1.(достаточности) Всякая частично-рекурсивная функция вычислима на машине Тьюринга.

Теорема 2.(необходимости) Всякая функция вычислимая на машине Тьюринга, частично-рекурсивна.

Лекция 18

Трудоемкость вычислений

В классической теории алгоритмов задача считается разрешимой, если существует решающий ее алгоритм. Однако для реализации некоторых алгоритмов при любых разумных с точки зрения физики предположениях о скорости выполнения элементарных шагов может потребоваться больше времени, чем по современным воззрениям существует вселенная. Поэтому возникает потребность конкретизировать понятие разрешимости и придать ему оценочный, количественный характер, введя такие характеристики алгоритмов, которые позволяли бы судить о возможности и целесообразности их практического применения.

Среди различных характеристик алгоритмов наиболее важными с прикладной точки зрения являются: время и память, расходуемые при вычислении.

Физическое время вычисления алгоритма характеризуется произведением τ t, где t – число действий вычисления, τ – среднее физическое время реализации одного шага. Число шагов t определяется

описанием алгоритма в данной алгоритмической модели, это – величина математическая, не зависящая от особенностей физической реализации модели. Напротив, τ зависит от реализации и определяется скоростью обработки сигналов в элементах, записи и считывания информации и т.д. Поэтому число действий t считают математическим временем вычисления алгоритма, определяющим физическое время вычисления с точностью до константы τ, зависящей от реализации.

Память как количественная характеристика алгоритма определяется количеством *s* единиц памяти (ячеек ленты машины Тьюринга или машинных слов в современных ЭВМ). Ясно, что эта величина по порядку не может превосходить числа шагов вычисления: $s \le \mu t$, где μ - максимальное число единиц памяти, используемых в данной машине на одном шаге. Напротив, t может существенно превосходить *s*. С этой точки зрения время более тонко определяет сложность алгоритма, чем память; и это – одна из причин, по которой исследованию временных характеристик алгоритма уделяется большее внимание.

Трудоемкость алгоритма – это число элементарных действий, выполненных при его вычислении.

Полной характеристикой конкретного варианта задачи является его формальное описание. Характеристикой сложности описания можно считать его объем, который называют *размерностью задачи* (например, для изоморфизма графов размерностью задачи можно считать число символов в матрицах смежности графов); тогда исследование трудоемкости алгоритма рассматривается как исследование зависимости трудоемкости вычисления от размерности задачи, решаемой алгоритмом.

И в математике, и на практике в конечном счете нас интересуют не алгоритмы сами по себе, а задачи, которые они решают. Одна и та же задача может решаться различными алгоритмами и на разных машинах. Если машина *M* зафиксирована, то трудоемкостью данной задачи относительно машины *M* называется минимальная из трудоемкостей алгоритмов, решающих задачу на машине *M.* Задач, трудоемкость которых была бы определена точно, довольно мало; хорошим результатом считается определение ее по порядку, т. е. с точностью до множителя, ограниченного некоторой константой. Чаще удается оценить ее сверху или снизу. Оценку сверху получают, указав конкретный алгоритм решения задачи: по определению, трудоемкость задачи не превосходит трудоемкости любого из решающих ее алгоритмов. Оценки трудоемкости снизу – гораздо более трудное дело; их получают обычно из некоторых общих соображений (например, мощностных или информационных).

Можно ли говорить об инвариантности теории трудоемкости вычислений? Иначе говоря, возможны ли утверждения о трудоемкости вычислений, сохраняющиеся при переходе к любой алгоритмической модели? Что касается прямых количественных оценок, то инвариантами не являются не только константы, но и степени. Например, доказано, что

трудоемкость распознавания симметричности слова длины n относительно его середины на машине Тьюринга не меньше, чем cn^2, тогда как для любой ЭВМ, имеющей доступ к памяти по адресу, допускающей операции над адресами, легко написать программу, решающую эту задачу с линейной трудоемкостью.

Таким образом, скорости вычислений на разных моделях различны. Однако строить теорию трудоемкости вычислений, привязываясь к некоторым конкретным моделям, неудобно ни для теории, ни для практики. Для теории – потому, что такая привязка не дает достаточно объективных характеристик трудоемкости задачи, т.е. не позволяет отделить влияние особенностей выбранной модели от специфики самой задачи; для практики — потому, что разнообразие реальных машин растет, и нужны общие понятия и методы оценки трудоемкости решения задач, которые сохраняют свою силу при любых изменениях в мире компьютеров. Поэтому инвариантная теория трудоемкости нужна, и вопрос не в том, возможна ли она, а в том, как ее построить (т.е. какие инварианты найти). Для того чтобы обсуждать этот вопрос, прежде всего следует посмотреть, как меняется трудоемкость при переходе от одной машины к другой. Это рассмотрение мы начнем с некоторого краткого обзора парка абстрактных машин, о которых будет идти речь.

До сих пор в явном виде была описана только одна абстрактная машина - машина Тьюринга. Можно неявно использовать машину другого типа, гораздо более близкую к современным ЭВМ, в которой возможен доступ к памяти по адресам. Такая машина, называемая *машиной с произвольным доступом к памяти*, может на следующем шаге переходить к любой ячейке с указанным адресом (команды условного и безусловного переходов) и реализовать команды-операторы вида b: $=f(a_1, a_2, ..., a_p)$ (выполнить операцию f над содержимым ячеек a_1, a_2, ..., a_p и результат положить в ячейку b). Возможны различные варианты моделей машины с произвольным доступом к памяти, в более сложных вариантах допускаются операции над адресами. Здесь мы не будем рассматривать все эти варианты, ограничившись фиксацией лишь одной простой модели - машины элементарных логических операций, или кратко L-машины. Относительно других моделей абстрактных машин ограничимся констатацией их основных свойств, которых будет достаточно при последующих рассмотрениях. Будем считать, что каждая машина имеет конечное число устройств (головок, устройств управления головками, процессоров-устройств, выполняющих элементарные операции, и т.д.), каждое устройство и каждая ячейка памяти могут находиться в одном из конечного числа возможных состояний (состояние ячейки памяти - это записанный в ней символ), и выполнение любого элементарного действия (шага) зависит от информации из конечного числа ячеек памяти, ограниченного некоторой константой μ. Будем говорить, что все ячейки читаются на данном шаге. Полное состояние машины, т. е. набор

состояний устройств, состояний ячеек памяти и указание ячеек, читаемых в настоящий момент, называется, *конфигурацией машины*.

Еще одно вступительное замечание. Алгоритм, осуществляемый машиной, может быть реализован двояким образом: он может быть «встроен» в управляющее устройство или записан в памяти машины. В первом случае машина является специализированной и может выполнять только данный алгоритм; чтобы изменить алгоритм, надо поменять управляющее устройство. Таковы машины Тьюринга в примерах. Во втором случае запись алгоритма в памяти называется программой, а сама машина – *программируемой*; алгоритм, встроенный в управляющее устройство, решает задачу исполнения программ, записанных в памяти машины. Такова универсальная машина Тьюринга и все реальные универсальные ЭВМ. В обоих случаях начальная конфигурация машины – состояния всей памяти и всех устройств – полностью определяет процесс вычисления.

Машина элементарных логических операций (L-машина) – это машина с произвольным доступом к памяти, имеющая следующую систему команд:

x: = 0;
x: = 1;
x: = y;
x: = ¬ y;
x: =y& z;
x: = y∨ z;
“конец”,

где *x, y, z* — адреса ячеек памяти; := - знак присвоения.

Программа элементарных логических операций (L-программа) с однократной записью – это программа, состоящая из указанных команд, в которой запись в каждую ячейку памяти производится не более одного раза (читать ячейку можно неоднократно).

Можно было бы дополнить систему команд L-машины командами ввода информации в ячейки памяти для данных, ввода программ и вывода результатов решения задач, однако будем считать, что эти элементы памяти доступны для ввода информации и обозрения извне (трудоемкость этих действий следует считать пропорциональной количеству элементов памяти, куда надо ввести или откуда надо вывести информацию). Пусть программа и данные (значения двоичных переменных) находятся в разных секциях памяти, и каждая секция имеет свою адресацию. Тогда можно так перекодировать программу с однократной записью, что номера команд программы и адреса ячеек памяти, в которых эти команды производят запись станут одинаковыми, и будут иметь вид i:i:= U_i (i=1,2, …, t-1), где выражения U_i имеют вид либо 0, либо 1, либо j, либо ¬ j, либо j& k, либо j∨ k (в конце программы стоит команда “конец”).

Интерпретация машинных действий
Интерпретация элементарных действий

Теоремы о трудоемкости

Теорема 1. Пусть Манина M_2 полиномиальным образом со степенью C интерпретируема на машине M_1, и задача *p* решается на машине M_2 из частичной конфигурации C_{M2}^{π} не более чем за t шагов, после чего машина останавливается. Тогда задача *p* может быть решена на машине M_1 не более чем за Bt^C шагов, где B – положительная константа, т.е. ее сложность относительно машины M_1 не больше Bt^C.

Теорема 2. Если трудоемкость задачи *p* относительно машины Тьюринга *T* не больше *t,* то ее можно решить при помощи не более чем $Bt^2+Ct=0$ (t^2) элементарных логических операций.

Полиномиальная интерпретируемость машин друг на друга. Аналогичным образом доказывается, что машина с произвольным доступом к памяти и «полным» набором команд − конечным набором операторов вычисления функций от конечного же количества двоичных переменных, условными и безусловными переходами и индексированными командами – полиномиальным образом интерпретируема на машине элементарных логических операций. Однако в этом случае степень больше трех (сложнее всего интерпретация индексированных команд, особенно адресов, которые могут появиться в результате индексации). Обратная полиномиальная интерпретируемость машины элементарных логических операций *L* на машине с произвольным доступом к памяти тривиальна, так как в системе команд последней машины есть все команды первой.

Так как машина с произвольным доступом к памяти полиномиальным образом интерпретируема на машине элементарных логических операций L, для доказательства ее полиномиальной интерпретируемости на некоторой машине Тьюринга *PU* достаточно доказать полиномиальную интерпретируемость машины L на машине *PU.* Кроме того, этим будет доказана полиномиальная интерпретируемость произвольных машин Тьюринга *T* на машине *PU.*

Теорема 3. L-машина полиномиальным образом интерпретируется на машине Тьюринга. Для доказательства нужна определенная техника программирования машин Тьюринга, и мы его опускаем.

7. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ПРАКТИЧЕСКИХ РАБОТ

План практических работ

№	Содержание практического занятия	Часы
	Математическая логика	
1	Логика высказываний. Логическое следование, принцип дедукции. Метод резолюций.	2
2	Исчисление высказываний. Аксиомы и правила вывода.	2
3	Основные метатеоремы исчисления высказываний. Теорема дедукции.	2
4	Исчисление предикатов и теории первого порядка. Аксиомы и правила вывода. Правила переименования свободных и связанных переменных.	2
5	Доказательства в исчислении предикатов. Префиксная нормальная форма	2
	Теория алгоритмов	
6	Примеры машин Тьюринга. Описание машин T_{+},$T_{коп}$,T_{2x},T_{++}.	2
7	Композиция машин Тьюринга. Пример машин T_{++}, $T_{\times}$. Вычисление предикатов на примере предиката P(x) – “x четное”. Проблема остановки.	2
8	Рекурсивные функции. Примитивно-рекурсивные функции (сложение, умножение, возведение в степень, урезанное вычитание, min, max).	2
9	Примитивно-рекурсивные операторы (оператор условного перехода, ограниченный оператор наименьшего числа)	2
Итого		18ч.

Практическое занятие №1

Тема: Логика высказываний. Логическое следование, принцип дедукции. Метод резолюций.

Для практического занятия использовать материалы лекций 1 и 2.

Контрольные вопросы и задания

Практическое занятие №2

Тема: Исчисление высказываний. Аксиомы и правила вывода.

Для практического занятия использовать материалы лекции 2 и 3. Из логики высказываний использовать понятия: тождественно-истинные формулы, логически верные умозаключения логики высказываний, логические схемы рассуждений, правило утверждения. Из построения исчисления высказываний использовать понятия: вывод и выводимость в исчислении высказываний, доказательство в теории Т.

Контрольные вопросы и задания

1. Как построено исчисление высказываний (основные принципы)?
2. Какие формулы являются аксиомами исчисления высказываний?
3. Какие правила вывода используются в исчислении высказываний?
4. Выводимость в исчислении высказываний, показать на примерах.
5. Выучить систему аксиом II.

Практическое занятие №3

Тема: Основные метатеоремы исчисления высказываний. Теорема дедукции.

Для практического занятия использовать материалы лекции 4. Из исчисления высказываний использовать понятия: метатеорема исчисления, теорема дедукции, применение теоремы дедукции на примерах.

Контрольные вопросы и задания

1. Теорема дедукции.
2. Применение теоремы дедукции. Примеры
3. Теорема о выводимости в исчислении высказываний.
4. Применение теоремы о выводимости. Проверка формул исчисления высказываний на выводимость.

Практическое занятие №4

Тема: Исчисление предикатов и теории первого порядка. Аксиомы и правила вывода. Правила переименования свободных и связанных переменных.

Для практического занятия использовать материалы лекций 5 и 6. Из исчисления предикатов использовать понятия: теория первого порядка, принципы построения исчисления предикатов, на примере правил переименования показать выводимость в исчислении предикатов.

Контрольные вопросы и задания

1. Как построено исчисление предикатов (основные принципы)?
2. Аксиомами исчисления предикатов?
3. Какие правила вывода используются в исчислении предикатов?
4. Выводимость в исчислении предикатов, показать на примерах.
5. Выучить систему аксиом исчисления предикатов.

Практическое занятие №5

Тема: Доказательства в исчислении предикатов. Префиксная нормальная форма.

Для практического занятия использовать материалы лекции 7. Из исчисления предикатов использовать понятия: доказательства в исчислении предикатов, понятие префиксной нормальной формы, этапы приведения формулы исчисления предикатов к префиксной нормальной форме, эквивалентности, используемые в исчислении предикатов.

Контрольные вопросы и задания

1) Метод доказательства от противного. Показать на примере предикатной формулы.
2) Понятие префиксной нормальной формы.
3) Процедура приведения к ПНФ формулы исчисления предикатов.
4) Эквивалентности, используемые в исчислении предикатов.
5) Заданные формулы привести к ПНФ.
6) Доказательство в исчислении предикатов.

Практическое занятие №6

Тема: Примеры машин Тьюринга. Описание машин T_{+},$T_{коп}$,T_{2x},T_{++}.

Для практического занятия использовать материалы лекций 11 и 12. Использовать понятия: машина Тьюринга, конфигурация машины Тьюринга, правильное вычисление функции на машине Тьюринга, протокол машины Тьюринга.

Контрольные вопросы и задания

1. Для машины Тьюринга T_+ использовать разные представления.
2. Для заданных машин T_+, $T_{коп}$,T_{++} построить протоколы, описать работу машин.

Практическое занятие №7

Тема: Композиция машин Тьюринга. Примеры машин T_{2x}, $T_{\times}$. Вычисление предикатов на примере предиката P(*x*) –"*x* четное". Проблема остановки.

Для практического занятия использовать материалы лекций 13 и 14. Использовать понятия: композиция машин Тьюринга, построение композиции машин Тьюринга, вычисление предикатов на машине Тьюринга, вычисление предикатов с восстановлением и без восстановления, тезис Тьюринга, проблема остановки, тезис Черча для машин Тьюринга.

Контрольные вопросы и задания

1. Ввести понятие композиции машин Тьюринга.
2. Для машины $T_+(T_{коп})$ построить протокол, определить какую функцию вычисляет эта машина.
3. Пример машины Тьюринга, вычисляющей предикат, построение протокола этой машины.
4. В чем заключается проблема остановки?
5. Тезис Тьюринга и Черча.

Практическое занятие №8

Тема: Рекурсивные функции. Примитивно-рекурсивные функции (сложение, умножение, возведение в степень, урезанное вычитание, min, max).

Для практического занятия использовать материалы лекции 15. Использовать понятия: оператора суперпозиции, примитивно-рекурсивной функции, схема рекурсии.

Контрольные вопросы и задания

1. Понятие примитивно-рекурсивной функции.
2. Схема рекурсии, привести примеры.
3. Тестирование примитивно-рекурсивных функций.

Практическое занятие №9

Тема: Примитивно-рекурсивные операторы.

Для практического занятия использовать материалы лекций 16 и 17. Использовать понятия: характеристическая функция примитивно-рекурсивного оператора, примитивно-рекурсивного оператора, оператор условного перехода, ограниченный оператор наименьшего числа μ-оператор.

Контрольные вопросы и задания

1. Понятие примитивно-рекурсивного оператора.
2. На примере применения ограниченного оператора наименьшего числа показать, что этот оператор примитивно рекурсивный.

8. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ КОНТРОЛЬНЫХ РАБОТ

Контрольная работа №1

«Исчисление высказываний, исчисление предикатов»

Вариант – I.

1. Определить истинность, ложность либо выполнимость в области N_0 натуральных чисел формулы: $\exists x\, S(x,y,y) \rightarrow \forall x\, S(x,y,y)$, где $S(x,y,y)$=’$x+y=y$’.

2. Получить ПНФ: $\exists x \forall y P_1(x,y) \rightarrow \forall x \exists y P_2(x,y)$.

3. Являются ли аксиомами следующие формулы:

   1.$((A \rightarrow B)\&A) \rightarrow B$
   2.$(A \rightarrow B) \rightarrow B$
   3.$A \rightarrow (A \rightarrow B)$

Вариант – II.

1. Определить истинность, ложность либо выполнимость в области N_0 натуральных чисел формулы: $\exists x\, \prod(y, x, y) \rightarrow \forall y\, \prod(y, x, y)$, где $\prod(y,x,y)$=’$yx=y$’.

2. Получить ПНФ: $\exists y P_1(x,y) \rightarrow \forall x P_2(x,y)$.

3. Являются ли аксиомами следующие формулы:

   1.$(A \rightarrow A)$
   2.$A \rightarrow (A \rightarrow B)$
   3.$((A \rightarrow B)\&A) \rightarrow B$

Вариант – III.

1. Определить истинность, ложность либо выполнимость в области N_0 натуральных чисел формулы: $\exists y\, \prod(x, x, y) \rightarrow S(x, x, y)$, где $\prod(x, x, y)$=’$xx=y$’; $S(x, x, y)$= ’$x+x=y$’.

2. Получить ПНФ: $\forall x \forall y P_1(x,y) \rightarrow \exists x \exists y P_2(x,y)$.

3. Являются ли аксиомами следующие формулы:

1. $(A \rightarrow B)$
2. $A \rightarrow (B \rightarrow A)$.
3. $((A \rightarrow B)\&B) \rightarrow A$

Для выполнения контрольной работы №2 использовать материалы лекций 6, 7, 8 и 9.

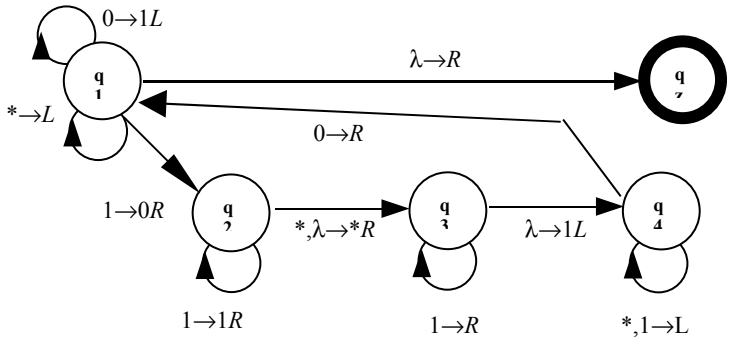
Контрольная работа №2

«Машины Тьюринга. Протокол машины Тьюринга»

Вариант 1

Построить диаграмму переходов для машины Тьюринга

4. сложения и написать протокол для x = 4 y =5;
5. Какую функцию натурального аргумента вычисляет машина Тьюринга, заданная диаграммой переходов:

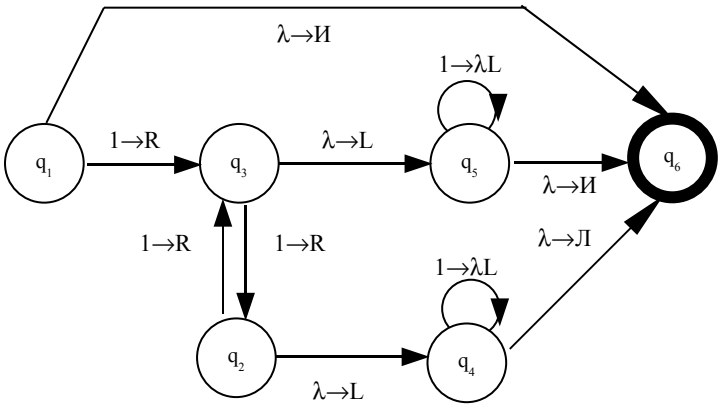


Вариант 2

Построить систему команд для машины Тьюринга

1) копирования и написать протокол для x = 5;

6. Какую функцию натурального аргумента вычисляет машина Тьюринга, заданная диаграммой переходов:



Вариант 3

Построить таблицу переходов для машины Тьюринга

7. вычисления предиката P(x)–“x нечетное число” с восстановлением и написать протокол для x = 4;
8. Какую функцию натурального аргумента вычисляет машина Тьюринга, заданная таблицей переходов:

	q_1	q_2	q_3
1	$\lambda\, q_2+1$	$1\, q_2+1$	$1q_3-1$
*	$\lambda\, q_z+1$	$1q_3-1$	
λ			$\lambda\, q_z+1$

Для выполнения контрольной работы №2 использовать материалы лекций 12 и 13.

Контрольная работа №3

«Примитивно-рекурсивные функции»

Используя схему рекурсии, предложенных в вариантах примитивно-рекурсивных функций определить значения функции для заданных значений переменных, составив протокол.

Вариант 1

1) сложение при x = 4 y =5;
2) урезанное вычитание при x = 14 y =3.

Вариант 2

1) умножение x = 4 y =5;
2) min(x = 25 y =13), max(x = 11 y = 45).

Вариант 3

1) возведение в степень x = 2 y =6;
2) сигнум (sg) при x = 13.

Для выполнения контрольной работы №3 использовать материалы лекции 15.

9. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ОРГАНИЗАЦИИ МЕЖСЕССИОННОГО КОНТРОЛЯ ЗНАНИЙ СТУДЕНТОВ

1. Межсессионная аттестация студентов проводится дважды в семестр на 7 и 13 неделях семестра.

2. Аттестационная оценка выставляется по результатам работы в семестре: выполнения домашних заданий, выполнения контрольных работ, сдачи промежуточного коллоквиума по 1-ой части дисциплины, успешного тестирования по 1-ой и 2-ой частям дисциплины, а также посещения практических занятий и посещений лекционных занятий.

3. Организация аттестации студентов, проводится в соответствии с положением АмГУ о курсовых, экзаменах и зачетах.

10. КОМПЛЕКТЫ ЗАДАНИЙ К КОЛЛОКВИУМУ

Практические задания к коллоквиуму по части «Математическая логика»

1.Какие формулы являются аксиомами исчисления предикатов?
а) $\forall x F(x) \to F(y)$;
б) $\exists y F(y) \to \forall x F(x)$;
в) нет таких формул.

а) $\forall x F(x) \to F(y)$;
б) $F(y) \to \exists x F(x)$;
в) нет таких формул.

Какие формулы являются аксиомами исчисления предикатов?
а) $\forall x F(x) \to \exists y F(y)$;
б) $\exists y F(y) \to \forall x F(x)$;
в) $F(y) \to \exists x F(x)$.

Являются ли соотношения эквивалентными?
а) $\neg(\exists x F(x)) \sim \exists x \neg F(x)$;
б) $\neg(\forall x F(x)) \sim \exists x \neg F(x)$.

Убедиться, что формула является тавтологией (тождественно ложной)

$A\&(B\to A)\ \&\ (A\to A\&(B\to A))$

2.“Всякая тождественно-истинная формула является теоремой исчисления высказываний.” Указать варианты, содержащие теоремы:
а) $\Gamma, U \vdash B$, то $\Gamma \vdash U\to B$;
б) $\vdash U\to B$;
в) $\vdash A\to(A\to A)$;
г) $\vdash (U\to B)\to B$.

а) $\Gamma, U \vdash B$, то $\vdash U\to B$;
б) $\vdash \neg(A\&B) \sim \neg A\vee\neg B$;
в) $\vdash A\to(A\to A)$;
г) $\vdash (U\to B)\to B$.

Если $\Gamma,A \vdash B$ и $\Gamma,A \vdash \neg B$, то $\Gamma\vdash \neg A$.
а) это теорема дедукции;
б) это правило введения отрицания.

Если Г,А |— В, то Г|— А.→В.
а) это теорема дедукции;
б) это правило введения отрицания.

Если $\mathfrak{R}$ – выводимая формула, содержащая букву A ($\mathfrak{R}$ (A)), то выводима формула $\mathfrak{R}(B)$, где B тождественно-истинная формула: $\frac{\mathfrak{R}\,(A)}{\mathfrak{R}\,(B)}$
а) это правило заключения;
б) это правило подстановки;
в) это не есть правило.

3.Проверить истинность, ложность, выполнимость формул:

$\forall x((\mathrm{P}(x) \to \mathrm{Q}(x)) \,\&\, (\mathrm{Q}(x) \to \mathrm{P}(x)))$
а) тождественно-истинная формула
б) тождественно-ложная формула
с) формула принимает разные значения на наборах предикатов P,Q.

$\exists x S(x,y,y) \to \neg\forall y S(x,y,y)$, где $\mathrm{S}(x,y,z)$ = "$x+y=z$ ", $x,y,z \in \boldsymbol{N}$;
а) тождественно-истинная формула
б) тождественно-ложная формула
с) формула принимает разные значения на $\boldsymbol{N}$

$\exists x S(y,x,y) \to \forall y S(y,x,y)$, где $\mathrm{S}(x,y,z)$ = "$x+y=z$ ", $x,y,z \in \boldsymbol{N}_0$;
а) тождественно-истинная формула
б) тождественно-ложная формула
с) формула принимает разные значения на $\boldsymbol{N_0}$

$\exists x П(x,y,y) \to \forall y П(x,y,y)$, где $П(x,y,z)$ = "$x*y=z$ ", $x,y,z \in \boldsymbol{N}$;
а) тождественно-истинная формула
б) тождественно-ложная формула
с) формула принимает разные значения на $\boldsymbol{N}$

$\exists x \forall y\, П(x,y,y) \to \exists y \forall x П(x,y,y)$, где $\mathrm{П}(x,y,z)$ = "$x*y=z$ ", $x,y,z \in \boldsymbol{N}_0$;
а) тождественно-истинная формула
б) тождественно-ложная формула
с) формула принимает разные значения на $\boldsymbol{N_0}$

Доказать истинность формулы

$\forall x(P(x) \to Q(\mathrm{x})) \vee (Q(\mathrm{x}) \to P(\mathrm{x}))$,

$\forall x(P(x) \vee \neg P(x))$
$\forall x(P(x) \& \neg P(x))$
$\forall x(P(x) \to \neg P(x))$.

4.Привести формулу логики предикатов к ПНФ

$\forall x F(x) \to F(y)$
$\forall y(F_1(x,y) \vee \neg \exists x\, F_2(x,y))$
$\exists x \forall y P_1(x,y) \to \neg \exists x \forall y P_2(x,y)$
$\exists x \forall y P_1(x,y) \to \forall x \exists y P_2(x,y)$
$(\exists x \forall y P_1(x,y) \vee \exists x P_2(x)) \to \exists y \forall z\, P_3(y,z)$

5.Выводимость в исчислении высказываний

$\vdash A \to A$
$A \vdash B \to A$
$\vdash (\neg A \to \neg B) \to ((\neg A \to B) \to A)$
выводимость в правиле введения отрицания $((\Gamma, A \vdash B, \Gamma, A \vdash \neg B) \to \Gamma \vdash \neg A)$
закон исключенного третьего $(\vdash A \vee \neg A)$
Правило переименования связанных переменных с доказательством выводимости (если $\vdash \forall x\, F(x)$, то $\vdash \forall y\, F(y)$)
Правило переименования свободных переменных с доказательством выводимости (если $\vdash F(x)$, то $\vdash F(y)$)

11. КОМПЛЕКТЫ ЗАДАНИЙ К ЗАЧЕТУ

Зачет по дисциплине «Математическая логика и теория алгоритмов» состоит из двух частей: «Математическая логика» и «Теория алгоритмов».

Часть I.
Математическая логика

1. Принципы построения формальных теорий.

2. Вывод формулы В. Доказательство формулы В.

3. Теоремы теории Т. Доказательство теорем в теории.

4. Исчисление высказываний. Принципы построения теории.

5. Формулы схемы формул исчисления высказываний.

6. Аксиомы исчисления высказываний. Аксиоматический метод.

7. Правила вывода в исчислении высказываний.

8. Теорема дедукции. Применение теоремы дедукции.

9. Применение теоремы дедукции. Метод доказательства от противного.

10. Теорема о выводимости формулы в исчислении высказываний. Пример.

11. Теоремы о тождественной истинности в исчислении высказываний. Примеры.

12. Полнота и непротиворечивость теории. Теоремы Геделя.

13. Исчисление предикатов и теории первого порядка. Принципы построения теории.

14. Аксиомы исчисления предикатов.

15. Правила вывода в исчислении предикатов.

16. Выводимость и истинность в исчислении предикатов. Эквивалентности, выводимые в исчислении предикатов.

17. Предваренная форма или префиксная нормальная форма (ПНФ). Эквивалентности необходимые для приведения формулы к ПНФ.

18. Пример теории первого порядка исчисления с равенством.

19. Теория первого порядка исчисления частичного нестрогого порядка.

20. Формальная арифметика (принцип индукции).

Часть II.
Теория алгоритмов

1. Понятие алгоритма. Основные требования к алгоритмам.

По заданной машине T с внешним алфавитом A ={|, λ, *} и слову u = | |*| λ| найти слово T(u):

	q_1	q_2	q_3
|	λ q_2 +1	| q_2 +1	| q_1 -1
*	λ q_z+1	| q_3 -1	
λ			λ q_z +1

Построить диаграмму переходов.

2. Определение машины Тьюринга. Пример.

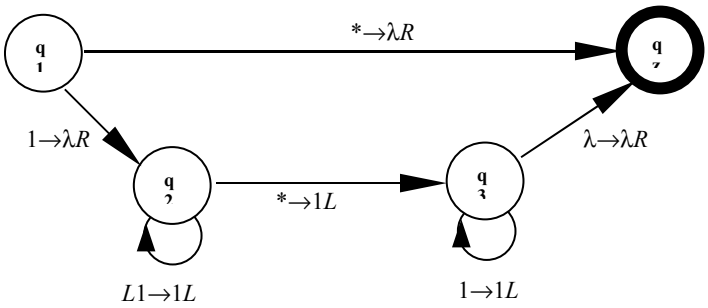
По заданной машине T с внешним алфавитом A ={|, λ} и слову u = | | λ | λ | | найти слово T(u):

	q_1	q_2
|	λ q_2 +1	| q_2 -1
λ	| q_z 0	λ q_1 +1

Построить диаграмму переходов.

3. Конфигурация или полное состояние машины Тьюринга. Стандартная начальная конфигурация, стандартная заключительная конфигурация. Правильная запись слов над алфавитом $A_{исх}$ на ленте.

Какую функцию натурального аргумента вычисляет машина Тьюринга, заданная диаграммой переходов:



По заданной машине T с внешним алфавитом A и слову u = | * | | найти слово T(u).

4. Память машины Тьюринга, данные Машины Тьюринга, детерминированность машины Тьюринга.

Какую функцию натурального аргумента вычисляет машина Тьюринга, заданная таблицей:

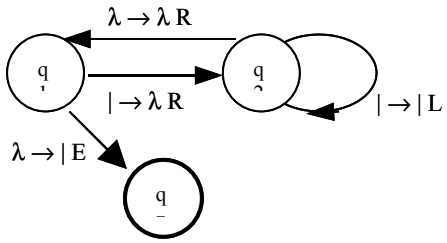
T

	q_1	q_2	q_3	q_4
|	0 q_2+1	|q_2+1	|q_3+1	|q_4-1
*	*q_1-1	*q_3+1	-	*q_4-1
λ	λ q_z+1	*q_3+1	|q_4-1	-
0	|q_1-1	-	-	0 q_1+1

Построить диаграмму переходов.

5. Представления машин Тьюринга. Система команд, построение таблицы переходов, построение диаграммы переходов машин Тьюринга. Примеры.

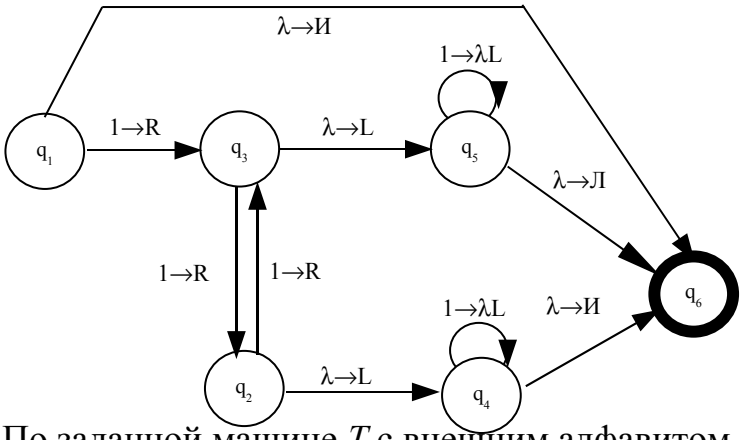
Какую функцию натурального аргумента вычисляет машина Тьюринга, заданная диаграммой переходов:



По заданной машине T с внешним алфавитом А и словам $u_1 = | | \lambda | |$ и $u_2 = | | |$ найти слова на ленте $T(u_1)$ и $T(u_2)$.

6. Понятие функции правильно вычислимой по Тьюрингу. Пример.

Какую функцию натурального аргумента вычисляет машина Тьюринга, заданная диаграммой переходов:



По заданной машине T с внешним алфавитом А и слову u = | | | | найти слово T(u).

7. Машина Тьюринга вычисляющая сложение (T_+).

Какую функцию натурального аргумента вычисляет машина Тьюринга, заданная таблицей:

T

	q_1	q_2
|	| q_2 +1	| q_2 +1
λ	| q_z 0	| q_z 0

По заданной машине T с внешним алфавитом А ={|, λ} и слову u = | | λ | λ | | найти слово T(u).

8. Машина Тьюринга вычисляющая копирование ($T_{коп}$).

9. Машина Тьюринга - композиция машин, вычисляющая функцию $f(x)=2x$ ($T_{+}(T_{\text{коп}})$).

10. Машина Тьюринга с правой полулентой. Пример машины T_{++}.

11. Вычисление предикатов на машинах Тьюринга. Пример машины с восстановлением.

12. Тезис Тьюринга. Проблема остановки.

По заданной машине T с внешним алфавитом A ={|, λ} и слову u = | λλ | найти слово T(u):

	q_1	q_2
|	| q_2 +1	| q_2 +1
λ	| q_z 0	| q_z 0

Построить диаграмму переходов.

13. Понятие рекурсивной функции. Схема рекурсии.

Какую функцию натурального аргумента вычисляет машина Тьюринга, заданная таблицей:

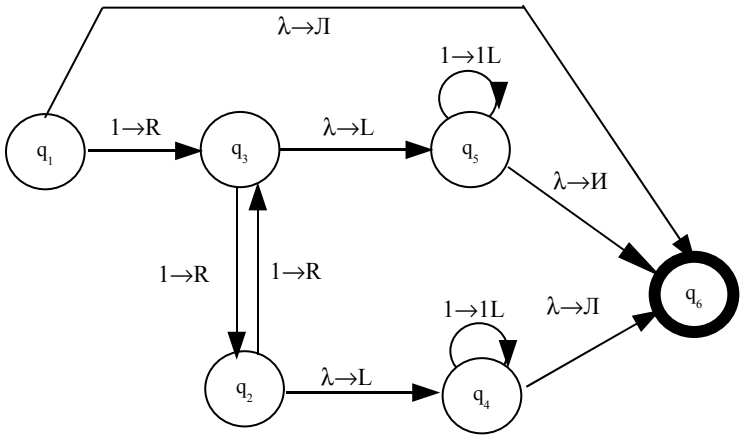
T

	q_1	q_2	q_3	q_4	q_5
|	λ q_2 +1	|q_2 +1	|q_3 -1	|q_4 +1	|q_5 -1
*	λq_4 +1	|q_4 +1	-	*q_3 -1	-
λ	-	-	λq_1 +1	λq_5 -1	λq_z +1

Построить диаграмму переходов.

14. Схемы рекурсии функций сложения $f_{+}(x, y)=x+y$, умножения $f_{\times}(x, y)=xy$.

Какую функцию натурального аргумента вычисляет машина Тьюринга, заданная диаграммой переходов:



По заданной машине T с внешним алфавитом А и слову u = | | И | найти слово T(u).

15. Схема рекурсии функции возведения в степень $f_{exp}(x, y) = x^y$.

16. Схемы рекурсии функций арифметическое или урезанное вычитание $f(x, y)$ = x ÷ y и $f(x, y)$ =|x - y|.

17. Схемы рекурсии функций min(x, y) и max(x, y).

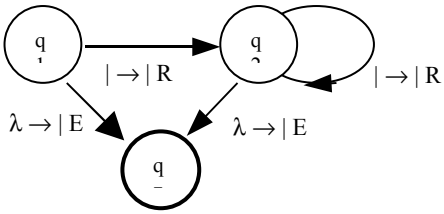
Какую функцию натурального аргумента вычисляет машина Тьюринга, заданная таблицей:

	q_1	q_2	q_3	q_4
				T
|	λ q_2+1	|q_2+1	|q_3-1	|q_4+1
*	λq_4+1	|q_4+1	-	*q_3-1
λ	-	-	λq_1+1	λq_z-1

По заданной машине T с внешним алфавитом А и слову u = | |*| *| *| | найти слово T(u).

18. Примитивно-рекурсивные предикаты. Характеристическая функция предиката. Пример характеристической функции отношения $R_{>}$ = “>”.

Какую функцию натурального аргумента вычисляет машина Тьюринга, заданная диаграммой переходов:



По заданной машине T с внешним алфавитом A и слову u = | | | найти слово T(u).

12. ТЕСТОВЫЕ ЗАДАНИЯ

«Математическая логика»

Вариант1

I. “Всякая тождественно-истинная формула является теоремой исчисления высказываний.” Указать варианты, содержащие теоремы:

а) Г, U |— B, то Г |— U→B;
б) |— U→B;
в) |— A→(A→A);
г) |— (U→B) →B.

II. Какие формулы являются аксиомами исчисления предикатов?

а) $\forall x\mathrm{F}(x) \to \mathrm{F}(y)$;
б) $\exists y\mathrm{F}(y) \to \forall x\mathrm{F}(x)$;
в) нет таких формул.

III. $\forall x((\mathrm{P}(x) \to \mathrm{Q}(x)) \,\&\, (\mathrm{Q}(x) \to \mathrm{P}(x)))$

а) тождественно-истинная формула
б) тождественно-ложная формула
с) формула принимает разные значения на наборах предикатов P,Q.

«Математическая логика»
Вариант2

I. Если $\mathfrak{R}$ – выводимая формула, содержащая букву A ($\mathfrak{R}(A)$), то выводима формула $\mathfrak{R}(B)$, где B произвольная формула: $\dfrac{\mathfrak{R}(A)}{\mathfrak{R}(B)}$

а) это правило заключения;
б) это правило подстановки;
в) это не есть правило.

II. Какие формулы являются аксиомами исчисления предикатов?

а) $\forall x\mathrm{F}(x) \to \mathrm{F}(y)$;
б) $\mathrm{F}(y) \to \exists x\mathrm{F}(x)$;
в) нет таких формул.

III. $\forall x((\mathrm{P}(x) \to \mathrm{Q}(x)) \vee \neg\, (\mathrm{P}(x) \to \mathrm{Q}(x)))$

а) тождественно-ложная формула
б) тождественно-истинная формула
с) формула принимает разные значения на наборах предикатов P,Q.

«Математическая логика»
Вариант3

I. “Всякая тождественно-истинная формула является теоремой исчисления высказываний.” Указать варианты, содержащие теоремы:

а) Г, U |— B, то |— U→B;
б) |— ¬(A&B) ~ ¬A∨¬B;
в) |— A→(A→A);
г) |— (U→B) →B.

II. Какие формулы являются аксиомами исчисления предикатов?

а) $\forall x F(x) \to \exists y F(y)$;
б) $\exists y F(y) \to \forall x F(x)$;
в) $F(y) \to \exists x F(x)$.

III. Если Г,A |— B и Г,A |— ¬B, то Г|— ¬A.

а) это теорема дедукции;
б) это правило введения отрицания.

«Математическая логика»
Вариант4

I. Если $\mathfrak{R}$ – выводимая формула, содержащая букву A ($\mathfrak{R}(A)$), то выводима формула $\mathfrak{R}(B)$, где B тождественно-истинная формула: $\frac{\mathfrak{R}(A)}{\mathfrak{R}(B)}$

а) это правило заключения;
б) это правило подстановки;
в) это не есть правило.

II. Являются ли соотношения эквивалентными?

а) $\neg(\exists x F(x)) \sim \forall x \neg F(x)$;
б) $\neg(\forall x F(x)) \sim \exists x \neg F(x)$.

III. $\forall x((P(x) \to Q(x)) \vee (Q(x) \to P(x)))$

а) тождественно-истинная формула
б) тождественно-ложная формула
с) формула принимает разные значения на наборах предикатов P,Q.

«Математическая логика»
Вариант5

I. “Всякая тождественно-истинная формула является теоремой исчисления высказываний.” Указать варианты, содержащие теоремы:

а) $\vdash (U \to B) \to B$;
б) $\vdash (A \& B) \to B$;
в) $\vdash \neg\, \neg A \to A$;
г) $\Gamma, U \vdash B$, то $\Gamma \vdash U \to B$.

II. Являются ли соотношения эквивалентными?

а) $\neg(\exists x F(x)) \sim \exists x\, \neg F(x)$;
б) $\neg(\forall x F(x)) \sim \exists x\, \neg F(x)$.

III. Если $\Gamma, A \vdash B$, то $\Gamma \vdash A. \to B$.

а) это теорема дедукции;
б) это правило введения отрицания.

«Математическая логика»
Вариант6

I. Если $\mathfrak{R}$ и $\mathfrak{R} \to \mathbf{א}$ – выводимые формулы, то $\mathbf{א}$ выводима: $\dfrac{\mathfrak{R},\ \mathfrak{R} \to \mathbf{א}}{\mathbf{א}}$

а) это правило заключения;
б) это правило подстановки;
с) это не есть правило.

II. Какие формулы являются аксиомами исчисления предикатов?

а) $\forall x F(x) \to \exists y F(y)$;
б) $\exists y F(y) \to \forall x F(x)$;
в) нет таких формул.

III. $\forall x(\neg\, (P(x) \to Q(x)) \to P(x)))$

а) тождественно-ложная формула
б) тождественно-истинная формула
с) формула принимает разные значения на наборах предикатов P,Q.

«Теория алгоритмов»
Вариант1

I. Какую функцию натурального аргумента вычисляет машина Тьюринга, заданная таблицей переходов:

а) суммирование двух чисел, представленных в унарной форме;

б) копирование чисел, представленных в унарной форме;

в) функцию $f(x)=2x$, для целого положительного x, представленного в унарной форме;

г) предикат P(x)={“ x-четное число”}, без востановления;

д) нет такой функции.

	q_1	q_2	q_3	q_4	q_5
1	1 q_3 +1	1 q_3 +1	1 q_2 +1	λ q_4-1	λ q_5-1
λ	И q_z 0	λ q_4-1	λ q_5-1	И q_z0	Л q_z0

II. Машина T правильно вычисляет функцию f, если

1) для любых Vи W, таких, что f(V)=W, $q_1V^* \Rightarrow_T q_zW^*$, где V* и W*- правильные записи V и W соответственно;
2) для любого V, такого, что f(V) не определена, машина T, запущенная в стандартной начальной конфигурации q_1V^*, работает бесконечно;
3) пункт 1) и 2) вместе.

«Теория алгоритмов»
Вариант2

I. Какую функцию натурального аргумента вычисляет машина Тьюринга, заданная таблицей переходов:

а) суммирование двух чисел, представленных в унарной форме;

б) копирование чисел, представленных в унарной форме;

в) функцию $f(x)=4x$, для целого положительного x, представленного в унарной форме;

г) предикат P(x)={“ x-нечетное число”}, с востановлением;

д) нет такой функции.

	q_1	q_2	q_3
1	λ q_2+1	1 q_2 +1	1q_3 -1
*	λ q_z+1	1q_3 -1	
λ			λ q_z +1

II. “Всякий алгоритм может быть реализован машиной Тьюринга”

1) для алгоритмичных конструктивных процедур невозможно строить реализующие их машины Тьюринга;
2) это есть тезис Тьюринга;
3) это есть проблема остановки.

«Теория алгоритмов»
Вариант3

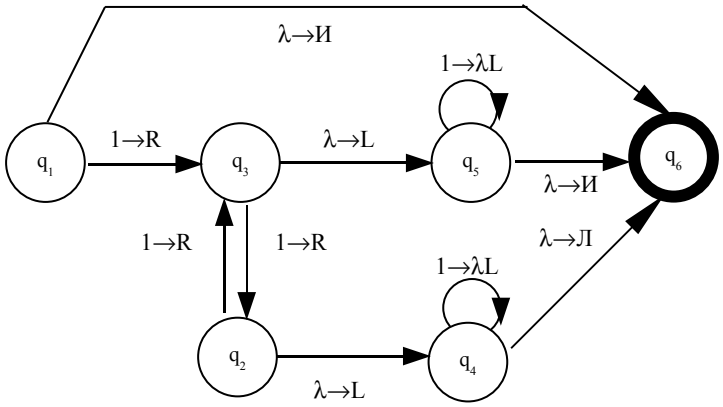
I. Какую функцию натурального аргумента вычисляет машина Тьюринга, заданная диаграммой переходов:

а) суммирование двух чисел, представленных в унарной форме;

б) копирование чисел, представленных в унарной форме;

в) функцию $f(x)=2x$, для целого положительного x, представленного в унарной форме;

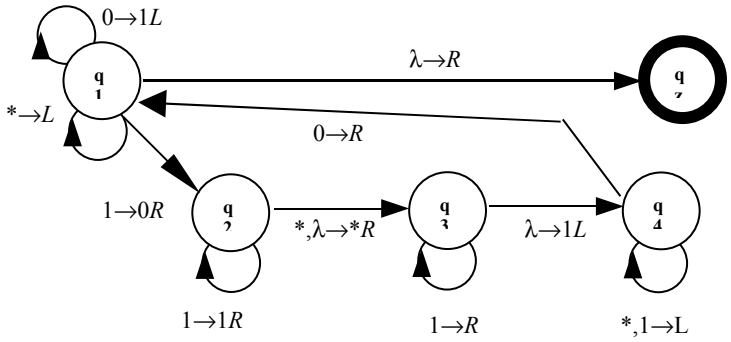
г) предикат P(x)={“ x-четное число”}, без востановления;
д) нет такой функции.



II. Построение машины T_0, такой, что для любой машины Тьюринга T и любых исходных данных α для машины T $T_0(\Sigma_T, \alpha)$ = И, если T(α) останавливается, и $T_0(\Sigma_T, \alpha)$ = Л, если T(α) не останавливается:
1) это есть тезис Тьюринга;
2) это есть алгоритмически неразрешимая проблема остановки;
3) для подобных алгоритмичных процедур можно строить реализующие их машины Тьюринга.

«Теория алгоритмов»
Вариант 4

I. Какую функцию натурального аргумента вычисляет машина Тьюринга, заданная диаграммой переходов:
а) умножение двух чисел, представленных в унарной форме;
б) копирование чисел, представленных в унарной форме;
в) функцию $f(x)=x$, для целого положительного x, представленного в унарной форме;
г) предикат P(x)={“ x-четное число”}, без востановления;
д) нет такой функции.



II. Не существует машины Тьюринга T, реализующей проблему остановки для произвольной машины Тьюринга:
1) это есть тезис Тьюринга;
2) это есть теорема об алгоритмически неразрешимой проблеме остановки.

«Теория алгоритмов»
Вариант5

I. Какую функцию натурального аргумента вычисляет машина Тьюринга, заданная диаграммой переходов:

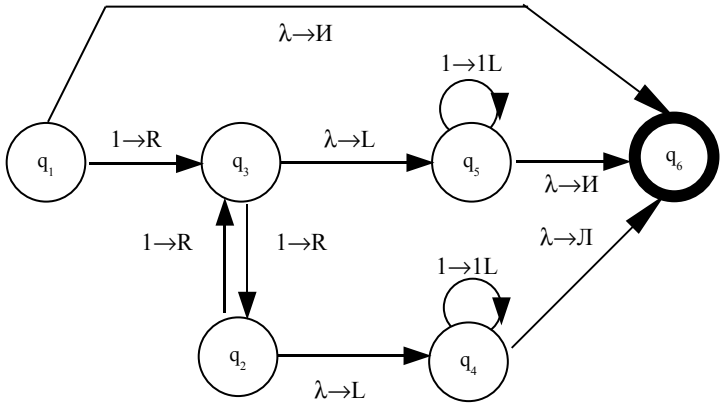
а) суммирование двух чисел, представленных в унарной форме;

б) копирование чисел, представленных в унарной форме;

в) предикат P(x)={“ x- нечетное число”}, с востановлением;

г) функцию $f(x)=2x$, для целого положительного x, представленного в унарной форме;

д) нет такой функции.



II. Машина Тьюринга с алфавитом $A_{исх}$ = {И, Л} и командами q_1 И$\rightarrow_T q_z$Л0, q_1 Л$\rightarrow_T q_z$И0 вычисляет:

1) отрицание логической переменной (¬P(α));

2) значение логической переменной (P(α));

3) предикат P(α) = {α- целое число}.

«Теория алгоритмов»
Вариант 6

I. Какую функцию натурального аргумента вычисляет машина Тьюринга, заданная таблицей переходов:

а) суммирование двух чисел, представленных в унарной форме;

б) копирование чисел, представленных в унарной форме;

в) функцию $f(x)=2x$, для целого положительного x, представленного в унарной форме;

г) предикат P(x)={“ x - нечетное число”}, с востановлением;

д) нет такой функции.

	q_1	q_2	q_3	q_4	q_5
1	1 q_3+1	1 q_3+1	1 q_2+1	λ q_4-1	λ q_5-1
λ	И q_z0	λ q_4-1	λ q_5-1	И q_z0	Л q_z0

II. “Всякий алгоритм может быть реализован машиной Тьюринга”

1) для алгоритмичных конструктивных процедур невозможно строить реализующие их машины Тьюринга;
2) это есть проблема остановки;
3) неверны оба утверждения.

13. КАРТА КАДРОВОЙ ОБЕСПЕЧЕННОСТИ ДИСЦИПЛИНЫ

Лектор	к.т.н., доцент	Семичевская Н.П.
Руководитель лабораторных работ	к.т.н., доцент	Семичевская Н.П.

Наталья Петровна СЕМИЧЕВСКАЯ
доцент кафедры информационных управляющих систем АмГУ

Редактор О.К. Мамонтова

МАТЕМАТИЧЕСКАЯ ЛОГИКА И ТЕОРИЯ АЛГОРИТМОВ
Учебно-методический комплекс дисциплины

Издательство АмГУ. Подписано к печати ___.___.07. Формат _____. Усл. печ.л. _____, уч.-изд. л. _____. Тираж _____. Заказ ___.