

МИНИСТЕРСТВО СЕЛЬСКОГО ХОЗЯЙСТВА РОССИЙСКОЙ  
ФЕДЕРАЦИИ

Федеральное государственное образовательное учреждение  
высшего профессионального образования  
«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ АГРАРНЫЙ УНИВЕРСИТЕТ»

Г. А. Аршинов, В. Г. Аршинов, С. В. Лаптев

МАТЕМАТИЧЕСКАЯ ЛОГИКА И ТЕОРИЯ АЛГОРИТМОВ  
Курс лекций

Под редакцией профессора В.И. Лойко

Краснодар  
2008

УДК 519(075.8)

ББК 22

A55

Рецензенты:

Доктор технических наук, профессор В.А. Атрощенко – декан факультета компьютерных технологий и автоматизированных систем Кубанского государственного технологического университета.

Доктор экономических наук, профессор М.И. Семенов – профессор кафедры информационных систем Кубанского государственного аграрного университета.

A55 Аршинов Г.А., Аршинов В.Г., Лаптев С.В. Математическая логика и теория алгоритмов: Курс лекций / Под ред. В.И.Лойко/. – Краснодар: КубГАУ, 2008. – 119 с.

В учебном пособии изложены элементы теории множеств, алгебры логики, исчисления высказываний, теории алгоритмов. Учебное пособие подготовлено в соответствии с требованиями Государственного образовательного стандарта высшего профессионального образования для специальности «Информационные системы и технологии».

ISBN 978-5-94672-307-7

© Аршинов Г.А., Аршинов В.Г., Лаптев С.В., 2008.

© Федеральное государственное образовательное учреждение высшего профессионального образования «Кубанский государственный аграрный университет», 2008.

## Содержание

|   |           |
|---|-----------|
| <b>Введение.....</b>  | <b>6</b>  |
| <b>Лекция 1. Элементы теории множеств.....</b>  | <b>7</b>  |
| 1. Основные понятия и определения.....  | 7         |
| 2. Операции над множествами и свойства операций.....  | 7         |
| <b>Лекция 2. Зависимости между элементами множеств.....</b>   | <b>11</b> |
| 1. Соответствия.....  | 11        |
| 2. Отображения, функции и отношения.....  | 13        |
| <b>Лекция 3. Высказывания и логические операции над<br/>    высказываниями.....</b>                                     | <b>16</b> |
| 1. Основные понятия логики высказываний .....   | 16        |
| 2. Логические операции над высказываниями .....   | 17        |
| 3. Свойства логических операций.....  | 22        |
| <b>Лекция 4. Формулы логики высказываний.....</b>   | <b>24</b> |
| 1. Понятие формулы логики высказываний и ее<br>логический смысл. Приоритет логических операций.....                     | 24        |
| 2. Вычисление значений истинности формул логики<br>высказываний.....  | 25        |
| 3. Тавтологично-истинные и тавтологично-ложные<br>формулы логики высказываний. Логическая<br>равносильность формул..... | 28        |
| <b>Лекция 5. Применение логики высказываний<br/>    в математических доказательствах.....</b>                           | <b>31</b> |
| 1. Доказательства прямые и от противного .....  | 31        |
| 2. Доказательство приведением к абсурду.<br>Необходимые и достаточные условия.....                                      | 32        |
| <b>Лекция 6. Применение логики высказываний к анализу<br/>    и синтезу переключательных (контактных) схем.....</b>     | <b>35</b> |
| 1. Представление законов логики<br>высказываний переключательными схемами .....   | 35        |
| 2. Примеры переключательных схем.....   | 38        |
| <b>Лекция 7. Булевы функции и булевы алгебры.....</b>   | <b>42</b> |
| 1. Булевы функции.....  | 42        |
| 2. Булевы алгебры.....  | 46        |
| <b>Лекция 8. Применение аппарата булевой алгебры<br/>    к анализу и синтезу комбинационных схем.....</b>               | <b>49</b> |
| 1. Комбинационные схемы (схемы без памяти).....   | 49        |

|   |           |
|---|-----------|
| <b>Лекция 9. Примеры комбинационных схем.....</b>                                 | <b>53</b> |
| 1. Функциональные схемы двоичных сумматоров.....                                  | 53        |
| 2. Логические операции, выполняемые<br>микропроцессором.....                      | 56        |
| <b>Лекция 10. Исчисление высказываний.....</b>                                    | <b>58</b> |
| 1. Понятие формулы исчисления высказываний.....                                   | 58        |
| 2. Аксиомы исчисления высказываний.....   | 60        |
| 3. Правила вывода. Доказуемые формулы.....  | 60        |
| <b>Лекция 11. Производные правила вывода.</b>                                     |           |
| <b>Выводимость формул.....</b>  | <b>64</b> |
| 1. Производные правила вывода.....  | 64        |
| 2. Понятие выводимости формулы из совокупности<br>формул.....                     | 67        |
| <b>Лекция 12. Основные понятия теории алгоритмов.....</b>                         | <b>70</b> |
| 1. Неформальное понятие алгоритма.<br>Свойства алгоритма.....                     | 71        |
| 2. Способы описания алгоритмов.....   | 72        |
| <b>Лекция 13. Виды алгоритмических процессов.....</b>                             | <b>75</b> |
| 1. Линейные и разветвляющиеся алгоритмические<br>процессы.....                    | 75        |
| 2. Циклические алгоритмические процессы.....                                      | 76        |
| <b>Лекция 14. Уточнение понятия алгоритма.....</b>                                | <b>79</b> |
| 1. Различные подходы к определению алгоритма.....                                 | 79        |
| 2. Машина Тьюринга.....   | 81        |
| <b>Лекция 15. Вычисления на машине Тьюринга.....</b>                              | <b>84</b> |
| 1. Вычисления на машинах Тьюринга.<br>Примеры машин Тьюринга.....                 | 84        |
| <b>Лекция 16. Примеры машин Тьюринга и способы<br/>        их задания.....</b>    | <b>90</b> |
| 1. Примеры машин Тьюринга.....  | 90        |
| 2. Задание машины Тьюринга в виде таблицы.....                                    | 92        |
| 3. Машины Тьюринга как словарные функции.....                                     | 93        |
| <b>Лекция 17. Методы построения программ для<br/>        машины Тьюринга.....</b> | <b>95</b> |
| 1. Суперпозиция машин (программ).....   | 95        |
| 2. Композиция машин (программ).....   | 97        |

|  |            |
|--|------------|
| <b>Лекция 18. Ветвление программ и циклы для</b>           |            |
| <b>    машины Тьюринга.....</b>                            | <b>99</b>  |
| 1. Ветвление машин (программ).....                         | 99         |
| 2. Циклические вычисления на машинах Тьюринга.....         | 100        |
| <b>Лекция 19. Рекурсивные функции.....</b>                 | <b>107</b> |
| 1. Вычислимые функции. Частично рекурсивные                |            |
| и общерекурсивные функции.....                             | 107        |
| <b>Лекция 20. Уточнение понятия алгоритма Маркова.....</b> | <b>113</b> |
| 1. Нормальные алгоритмы Маркова.....                       | 113        |
| 2. Некоторые неразрешимые алгоритмические                  |            |
| проблемы .....   | 115        |

## **Введение**

Математика является наукой, в которой все утверждения доказываются с помощью умозаключений, то есть путем использования законов человеческого мышления. Изучение законов человеческого мышления является предметом логики. Как самостоятельная наука логика оформилась в трудах греческого философа Аристотеля (384-322 г. до н.э.). Он систематизировал известные до него сведения, и эта система стала впоследствии называться формальной или Аристотелевой логикой. Формальная логика просуществовала без серьезных изменений более двадцати столетий. Естественно, что развитие математики выявило недостаточность Аристотелевой логики и потребовало дальнейшего ее развития.

Впервые в истории идеи о построении логики на математической основе были высказаны немецким математиком Г. Лейбницем (1646-1716) в конце XVII века. Он считал, что основные понятия логики должны быть обозначены символами, которые соединяются по особым правилам. Это позволит всякое рассуждение заменить вычислением. «Мы употребляем знаки не только для того, чтобы передать наши мысли другим лицам, но и для того, чтобы облегчить сам процесс нашего мышления» (Лейбниц).

Первая реализация идеи Лейбница принадлежит английскому ученому Д. Булю (1815-1864). Он создал алгебру, в которой буквами обозначены высказывания, и это привело к алгебре высказываний.

Применение математики к логике позволило представить логические теории в новой удобной форме и применить вычислительный аппарат к решению задач, малодоступных человеческому мышлению, и это, конечно, расширило область логических исследований.

Современную математическую логику определяют как раздел математики, посвященный изучению математических доказательств и вопросов оснований математики.

Математическая логика является составной частью теоретических основ информатики и математических моделей элементов компьютеров.

## Лекция 1. Элементы теории множеств

### Учебные вопросы:

1. Основные понятия и определения.
2. Операции над множествами и свойства операций.

### 1. Основные понятия и определения

**Определение.** Множеством называют совокупность элементов, объединенных по некоторому общему признаку.

**Примеры:** множество студентов университета, множество натуральных чисел.

Множества обозначаются большими буквами латинского алфавита, а элементы множеств - соответствующими малыми буквами с нижними индексами, нумерующими сами элементы.

**Пример.** Множество  $A = \{a_1, a_2, \dots, a_n, \dots\}$  содержит элементы  $a_1, a_2, \dots, a_n, \dots$ . Если элемент  $a_i$  принадлежит множеству  $A$ , то пишется  $a_i \in A$ .

**Определение.** Множество  $B$  называют подмножеством множества  $A$ , если каждый элемент  $B$  является также и элементом множества  $A$ . Этот факт обозначается  $A \subset B$ . (Читается: «Множество  $A$  включает множество  $B$ », или « $B$  является подмножеством множества  $A$ ».)

**Определение.** Множество, которое не содержит ни одного элемента, называют пустым множеством.

Пустое множество является подмножеством любого множества и обозначается символом  $\emptyset$ .

Множества могут содержать конечное или бесконечное число элементов. В первом случае их называют конечными, во втором – бесконечными. Например, множество студентов в группе – конечное, а множество целых положительных чисел, кратных 3 – бесконечное.

### 2. Операции над множествами и свойства операций

Пусть имеются множества  $A = \{a_1, a_2, \dots, a_n, \dots\}$  и  $B = \{b_1, b_2, \dots, b_n, \dots\}$ .

**Определение.** Объединением или суммой двух множеств  $A$  и  $B$  называют третье множество  $C$ , которое состоит из элементов, принадлежащих хотя бы одному из множеств  $A, B$ .

Объединение двух множеств  $A$  и  $B$  обозначается  $A \cup B$  или  $A+B$ . Пишется  $C = A \cup B$  или  $C = A+B$ . На рис. 1 геометрически показано объединение двух множеств  $A$  и  $B$ , элементами которых являются соответственно точки, лежащие внутри левого и правого контуров.



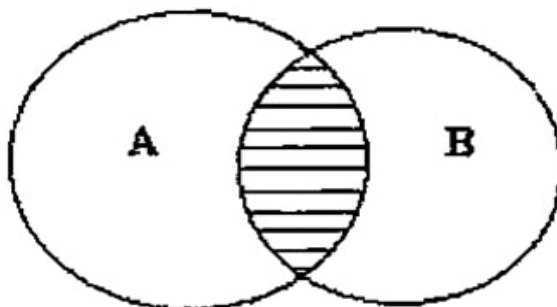
$A+B$ .

**Рис. 1. Объединение множеств  $A, B$**

**Пример.**  $A = \{1, 3, 5\}$ ,  $B = \{4, 7, 1, 3\}$ , тогда  $A+B = \{1, 3, 5, 4, 7\}$ .

**Определение.** Пересечением (или произведением) двух множеств  $A$  и  $B$  называют третье множество  $C$ , которое состоит из элементов, принадлежащих одновременно и множествам  $A$  и  $B$  (рис. 2).

Пересечение двух множеств  $A$  и  $B$  обозначается  $A \cap B$  или  $A \cdot B$ . Пишется  $C = A \cap B$  или  $C = A \cdot B$ . На рис. 2 геометрически показано пересечение двух множеств  $A$  и  $B$ , элементами которых являются соответственно точки, лежащие внутри левого и правого контуров.



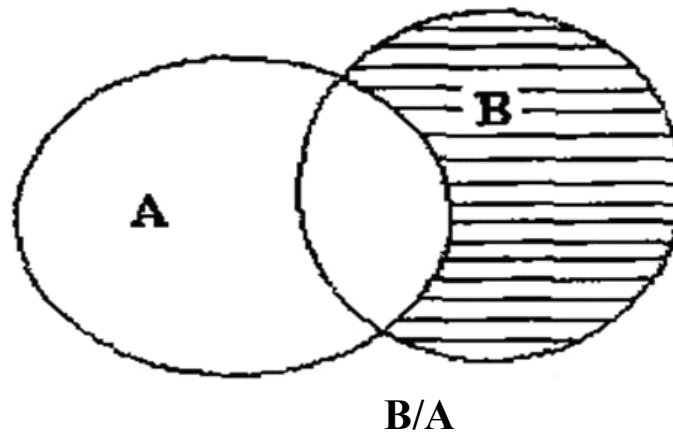
$A \cdot B$



## Рис.2. Пересечение множеств А, В

**Пример.**  $A=\{1,3,5\}$ ,  $B=\{4,7,1,3\}$ , тогда  $A \cdot B=\{1,3\}$

**Определение.** Разностью множеств В и А называют множество, состоящие из тех элементов множества В, которые не принадлежат А. Разность множеств В и А обозначается  $B \setminus A$  (рис. 3).



## Рис. 3. Разность множеств В, А

**Пример.**  $A=\{1,3,5\}$ ,  $B=\{4,7,1,3\}$ , тогда  $B/A=\{4,7\}$ .

**Некоторые свойства операций над множествами:**

1.  $A+B = B+A$  – коммутативность объединения множеств.
2.  $A \cdot B = B \cdot A$  – коммутативность пересечения множеств.
3.  $A+(B+C)=(A+B)+C$  – ассоциативность объединения множеств.
4.  $A \cdot (B \cdot C)=(A \cdot B) \cdot C$  – ассоциативность пересечения множеств.
5.  $A \cdot (B+C)=A \cdot B+A \cdot C$  – дистрибутивность пересечения относительно объединения.
6.  $A+(B \cdot C)=(A+B) \cdot (A+C)$  – дистрибутивность объединения относительно пересечения.
7.  $A+A \cdot C=A$  – закон поглощения.
8.  $A \cdot (A+C) = A$  – закон поглощения.

Пусть имеются множества:  $A = \{a_1, a_2, \dots, a_n, \dots\}$  и  $B = \{b_1, b_2, \dots, b_n, \dots\}$ .

**Определение.** Декартовым или прямым произведением множеств А и В называют множество С, состоящее из всех упорядоченных пар, в которых первый элемент выбран из множества А, а

второй – из множества  $B$ :  $C = \{(a_1, b_1), (a_1, b_2), (a_1, b_n), \dots, (a_2, b_1), (a_2, b_2), \dots, (a_2, b_n), \dots\}$ .

Декартово произведение множеств  $X$  и  $Y$  обозначается  $A \times B$ , то есть  $C = A \times B$ . Пара является упорядоченной, то есть порядок элементов в ней имеет существенное значение, иными словами, пара  $(a, b)$  не равна паре  $(b, a)$ .

**Примеры** декартового произведения.

1. Пусть имеются множества:  $A = \{1, 3, 6\}$  и  $B = \{3, 7\}$ , тогда  $A \times B = \{(1, 3), (1, 7), (3, 3), (3, 7), (6, 3), (6, 7)\}$

2. Пусть  $A = \{\text{Иванов, Петров}\}$ ,  $B = \{\text{высокий, сероглазый, сильный}\}$ . Тогда  $A \times B = \{(\text{Иванов, высокий}), (\text{Иванов, сероглазый}), (\text{Иванов, сильный}), (\text{Петров, высокий}), (\text{Петров, сероглазый}), (\text{Петров, сильный})\}$ .

## Лекция 2. Зависимости между элементами множеств

### Учебные вопросы:

1. Соответствия, отображения.
2. Функции и отношения.

### 1. Соответствия, отображения.

**Определение.** Будем говорить, что между множествами  $X$ ,  $Y$  установлено соответствие, если по определенному правилу каждому элементу подмножества  $A \subset X$  сопоставляется элемент подмножества  $B \subset Y$ . При этом совершенно необязательно, чтобы в сопоставлении участвовали все элементы множеств  $X$  и  $Y$ . Соответствие между двумя элементами множеств  $A, B$  называется бинарным отношением.

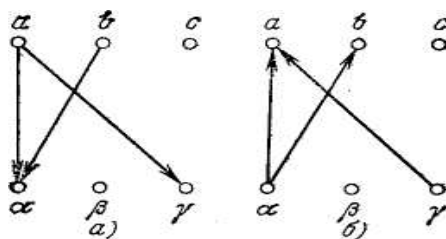
Обозначив соответствие  $q$ , его можно представить тройкой множеств  $q=(X, Y, Q)$ .  $X$  называется областью отправления соответствия, а  $Y$ —областью прибытия. Множество  $A$  называется областью определения соответствия,  $B$ —областью значения соответствия, а множество  $Q$  является подмножеством прямого произведения  $X \times Y$ .

**Пример 1.** Пусть  $X=\{1, 2\}$ ,  $Y=\{3, 5\}$ , тогда  $X \times Y=\{(1, 3), (1, 5), (2, 3), (2, 5)\}$ . Это множество дает возможность получить 16 различных соответствий. Некоторые из них:  $Q_1= \{(1, 3)\}$ ;  $Q_2=\{(1, 3), (1, 5)\}$ .

**Пример 2.** На предприятии есть две грузовые автомашины  $\alpha$ ,  $\beta$ , работающие в две смены, и автобус  $\gamma$ , используемый редко. Машина  $\beta$  находится в ремонте. В штате имеются три шофера  $a, b, c$ , из которых  $c$  находится в отпуске. Любое распределение шоферов по машинам представляет собой соответствие. Одним из возможных соответствий будет следующее:  $q=(\{a, b, c\}, \{\alpha, \beta, \gamma\}, \{(a, \alpha), (a, \gamma), (b, \alpha)\})$ , в котором область определения соответствия  $A= \{a, b\}$ , область значений  $B=\{\alpha, \gamma\}$ ,  $Q=\{(a, \alpha), (a, \gamma), (b, \alpha)\}$  (рис.1).

Элемент  $a$  соответствует элементам  $\alpha$  и  $\gamma$ , а элемент  $b$  — элементу  $\alpha$ .

**Определение.** Композицией соответствий называют последовательное применение двух соответствий.



**Рис. 1.Соответствие между водителями и машинами:  
а – прямое соответствие, б – обратное соответствие**

Композиция соответствий есть операция с тремя множествами  $X$ ,  $Y$  и  $Z$ , на которых определены два соответствия  $q = (X, Y, Q)$ , и  $r = (Y, Z, R)$ , причем область прибытия первого соответствия совпадает с областью отправления второго.

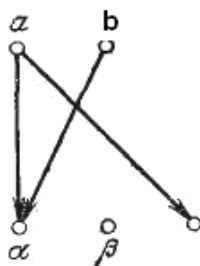
Композицию соответствий  $q$  и  $r$  будем обозначать  $q(r)$ .

**Пример.** Если  $q$  — соответствие, определяющее распределение шоферов по автомашинам,  $r$  — соответствие, определяющее распределение автомашин по маршрутам, то соответствие  $q(r)$  — соответствие, определяющее распределение водителей по маршрутам.

Операцию композиции можно распространить и на большее число соответствий.

**Определение.** Пусть  $X$  и  $Y$  — некоторые множества. Соответствие  $q=(X, Y, Q)$  называется отображением множества  $X$  во множество  $Y$ , если область определения соответствия совпадает с областью отправления  $X$ . Таким образом, каждому элементу отображение  $Q$  ставит в соответствие некоторый элемент подмножества множества  $Y$ , называемый образом элемента  $x$ .

**Пример.** Если в примере 2 исключить из рассмотрения шофера  $c$ , то получим отображение  $Q: X \rightarrow Y$ , в котором  $X=\{a,b\}$  — множество шоферов;  $Y=\{\alpha, \beta, \gamma\}$  — множество автомашин;  $Q = \{(a, \alpha), (a, \gamma), (b, \alpha)\}$  — распределение шоферов по автомашинам (рис. 2).



**Рис. 2. Геометрическое представление отображения**

## 2. Функции и отношения

**Определение.** Функцией называется отображение  $F: X \rightarrow Y$ , если оно однозначно, т. е. если для любого  $x$  из  $X$  существует единственный элемент  $y$  из  $Y$ .

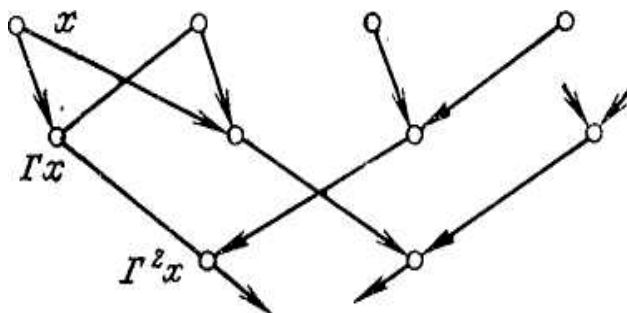
**Пример.** Из одного города в другой можно проехать по железной дороге (ж.д.), автобусом (авт.) или самолетом (сам.). Стоимость билета будет соответственно руб. Стоимость билета можно представить как функцию от вида транспорта. Для этого рассмотрим множества  $X = \{\text{ж.д.}, \text{авт.}, \text{сам.}\}$ ;  $Y = \{300, 290, 1000\}$ . Функция  $F: X \rightarrow Y$  представляется прямым произведением  $X \times Y$ , которое записывается в виде  $f = \{(\text{ж.д.}, 7), (\text{авт.}, 9), (\text{сам.}, 12)\}$ .

Важным частным случаем отображения является  $\Gamma: X \rightarrow Y$  случай, когда множества  $X$  и  $Y$  совпадают.

Частным случаем бинарного отношения называется отображение  $\Gamma: X \rightarrow X$ . Будем обозначать его  $(X, \Gamma)$ . Элемент, определяемый отношением  $\Gamma$  и соответствующий  $x$ , принадлежит тоже множеству  $X$  и обозначается  $\Gamma x$ .

**Пример.** Пусть  $X$  – множество людей. Обозначим через  $\Gamma$  отношение быть ребенком человека  $x$ , то есть  $y = \Gamma x \subset X$  является ребенком человека  $x$ . Тогда композиция отношений  $\Gamma(\Gamma x) = \Gamma^2 x$  — множество внуков человека  $x$ ;  $\Gamma^3 x$  — множество правнуков человека  $x$ ; обратное отображение  $\Gamma^{-1}x$  — множество родителей человека  $x$  и т.д.

Изображая людей точками и рисуя стрелки, идущие из  $x$  в  $\Gamma x$ , получаем родословное или генеалогическое дерево (рис. 3).



**Рис. 3. Генеалогическое дерево**

Пусть задано отношение  $\Gamma: X \rightarrow X$ . Если элемент  $x$  находится

в отношении  $\Gamma$  к элементу  $y$ , то это удобно записывать в виде  $y\Gamma x$ .  
Рассмотрим некоторые свойства отношений.

**Определение.** Отношение  $(X, \Gamma)$  называется *рефлексивным*, если  $x\Gamma x$  истинно и *антирефлексивным*, если  $x\Gamma x$  ложно.

**Определение.** Отношение  $(X, \Gamma)$  называется *симметричным*, если из  $x\Gamma y$  следует  $y\Gamma x$ , в противном случае - *несимметричным*.

**Определение.** Отношение  $(X, \Gamma)$  называется *антисимметричным*, если из  $x\Gamma y$  и  $y\Gamma x$  следует  $x=y$ .

**Определение.** Отношение  $(X, \Gamma)$  называется *транзитивным*, если из  $x\Gamma y$  и  $y\Gamma z$  следует  $x\Gamma z$ .

**Определение.** Отношение  $(X, \Gamma)$  называется *отношением эквивалентности*, если оно *рефлексивно, симметрично и транзитивно*, т.е. для любых элементов  $x, y, z$ , принадлежащих множеству  $X$ , имеет место:

1.  $x\Gamma x$ ;
2. Если  $x\Gamma y$ , то  $y\Gamma x$ ;
3. Если  $x\Gamma y$  и  $y\Gamma z$ , то  $x\Gamma z$ .

**Примеры.**

1. Отношение «быть на одном курсе», определенное на множестве студентов  $X$  какого-либо факультета есть отношение эквивалентности;

2. Отношение параллельности на множестве прямых плоскости – отношение эквивалентности;

3. Отношение подобия на множестве треугольников – отношение эквивалентности.

**Определение.** Подмножество элементов множества  $X$ , на котором определено отношение эквивалентности, эквивалентных некоторому элементу, будем называть *классом эквивалентности*.

Отношение эквивалентности разбивает множество  $X$  на непересекающиеся подмножества, называемые классами эквивалентности. Объединение всех классов эквивалентности дает множество  $X$ .

**Пример.** Пусть  $X$  — множество студентов второго курса, а отношением эквивалентности является отношение «быть в одной группе», тогда группа, в которой учится студент Иванов, будет классом эквивалентности, эквивалентным студенту Иванову.

Отношение эквивалентности разбило исходное множество студентов  $X$  на непересекающиеся классы эквивалентности (груп-

пы), объединение которых дает полное множество  $X$  студентов второго курса.

Рассмотрим отношения порядка, задающие некоторый порядок расположения элементов множества на основе понятий «раньше» и «позже», «больше» и «меньше». Различают отношение нестрогого порядка, для обозначения которого используется символ  $\leq$ , и отношение строгого порядка, для которого используется символ  $<$ .

**Определение.** Отношением нестрогого порядка на множестве  $X$  называют отношение, обладающее следующими свойствами:

1.  $x \leq x$  истинно (рефлексивность);
2. Если  $x \leq y$  и  $y \leq x$  истинно, то  $x=y$  (антисимметричность);
3. Если  $x \leq y$  и  $y \leq z$  истинно, то  $x \leq z$  (транзитивность).

**Определение.** Отношением строгого порядка на множестве  $X$  называют отношение, обладающее следующими свойствами:

1.  $x < x$  ложно (антирефлексивность);
2. Если  $x < y$  истинно, то и  $y < x$  ложно (несимметричность);
3. Если  $x < y$  и  $y < z$  истинно, то  $x < z$  (транзитивность).

**Определение.** Множества  $X$  называется упорядоченным, если любые два элемента  $x, y$  являются сравнимыми, т.е. находятся в отношении  $x < y$  или  $x=y$  или  $y < x$ .

**Определение.** Пусть  $X$  – множество людей. Если  $x$  в чем-то превосходит  $y$ , то на множестве  $X$  определено *отношение доминирования*.

В этом случае говорят, что  $x$  доминирует над  $y$ , и пишут  $x \gg y$ .

Отношение доминирования является:

- антирефлексивным** (нельзя доминировать над самим собой);
- несимметричным** (в каждой паре только один доминирует над другим);
- не является транзитивным**.

В отношении доминирования свойство транзитивности не имеет места. Например, если в соревнованиях команда  $x$  победила команду  $y$ , а команда  $y$  победила команду  $z$ , то отсюда еще не следует, что команда  $x$  обязательно победит команду  $z$ .

### Лекция 3. Высказывания и логические операции над высказываниями

#### Учебные вопросы:

1. Основные понятия логики высказываний.
2. Логические операции над высказываниями.
3. Свойства логических операций.

#### 1. Основные понятия логики высказываний

Все свои мысли человек облакает в высказывания. Из одних высказываний строятся другие высказывания, делаются логические выводы, обобщения. Все теоремы в математике, законы в физике, химии, биологии, медицине, юриспруденции и т.д. – это высказывания. Высказывание является основой любого человеческого языка.

**Определение.** *Высказыванием называется повествовательное предложение (утверждение), относительно которого объективно можно сказать, что оно истинно или ложно.*

Это весьма сильное требование. В реальной жизни используются не только такие высказывания. Например, высказывания «*Возможно*, что мы все-таки поедим этим летом на Кавказ», «*Вероятно*, что этот участок трассы давно не ремонтировали» и т.п. Такие высказывания называют *модальными*.

Есть так называемые *нормативные* высказывания, широко используемые в юридической практике: «Проезд перекрестка на красный сигнал светофора *запрещен*», «По улице Зеленой *разрешено* двухстороннее движение», «Для закона *безразличен* цвет глаз истца» и т.п. Раздел логики, оперирующий с такими высказываниями, называют *нормативной логикой*.

Например, высказывания «Москва – столица Франции», «Корень квадратный из 36 равен 6», «Все лошади имеют по четыре ноги» и т. д. объединяет лишь то, что они либо истинны, либо ложны.

В логике высказываний не принимается во внимание смысл высказываний и рассматривается не содержательная, а формальная сторона высказываний, т.е. истинно высказывание или ложно, и как это проверить.



**Определение. Высказывания, содержащие одно утверждение, называют простыми высказываниями.**

Они не могут быть «разложены» на более элементарные высказывания, относительно которых сохранилась бы объективная возможность оценить их истинность.

Из простых высказываний могут составляться (строиться) другие, более сложные высказывания. Такие высказывания мы будем называть составными, или сложными высказываниями.

В русском языке (и не только в русском) составные высказывания строятся из простых с помощью союзов **и**, **или**, частицы **не** и словосочетаний **если...,то...; ...тогда и только тогда, когда...; ...если, и только если...;...необходимо и достаточно для...** и т.д.

Например, «Деньги хранят в банке **или** в коробке из под конфет», «**Если** все три стороны треугольника равны, **то** равны и его углы», «Для того, чтобы четырехугольник был квадратом, **необходимо и достаточно**, чтобы все его углы были прямыми», «**Не** является верным, что трижды четыре – девять» и так далее.

Логическое значение простого высказывания (то есть, истинно оно или ложно) можно оценить по содержанию. Что же касается составных высказываний, то здесь дело обстоит не так просто. Например, трудно оценить истинность такого высказывания: «**Если** огород зарос бузиной **или** дядя уехал в Киев, **то** дважды два – пять». Это высказывание может оказаться как истинным, так и ложным в зависимости от истинности двух высказываний, составляющих его посылку (т.е. «Огород зарос бузиной» и «Дядя уехал в Киев»). Поэтому задача состоит в том, чтобы научиться «вычислять» значение истинности составного высказывания в зависимости от логических значений составляющих его простых высказываний.

## **2. Логические операции над высказываниями**

Чтобы сосредоточиться на формальной (структурной) стороне составных высказываний, условимся обозначать простые высказывания буквами начала латинского алфавита: **А, В, С** (возможно с индексами: **А<sub>1</sub>, А<sub>2</sub>, А<sub>3</sub>** и так далее), а значения истинности высказываний - цифрой **1** или **И** (истина), цифрой **0** и или буквой **Л** (ложь), которые называют логическими константами.

Определим логические операции над высказываниями, которые будут соответствовать грамматическим связкам *и, или*, частице *не*, словосочетаниям *если ..., то ...; ...тогда и только тогда, когда ....; ...если, и только если ...; ...необходимо и достаточно для...* и т.д. русского языка.

Логические операции будут определены на множестве  $\{0, 1\}$ , а результаты этих операций также будут значениями из этого множества. Задаются логические операции с помощью таблиц.

**Определение.** *Отрицанием заданного высказывания  $A$  называется другое высказывание, обозначаемое символом  $\neg A$  и являющееся истинным, если исходное высказывание ложно, и ложным в противном случае.*

Отрицание соответствует частице «не». Отрицание  $\neg A$  читается **не  $A$** . Операции отрицания задается таблицей истинности

| $A$      | $\neg A$ |
|----------|----------|
| <b>1</b> | <b>0</b> |
| <b>0</b> | <b>1</b> |

**Пример.**  $A = \{\text{Карась – рыба.}\}$ ,  $\neg A = \{\text{Карась – не рыба.}\}$ .

**Определение.** *Дизъюнкцией высказываний  $A, B$  называется новое высказывание, являющееся ложным лишь в случае ложности обоих высказываний  $A, B$  и истинным во всех остальных случаях.*

Символически дизъюнкция записывается в виде  $A \vee B$  или  $A + B$ . Дизъюнкция соответствует союзу «или». Дизъюнкция  $A \vee B$  читается  **$A$  или  $B$** .

Операции дизъюнкция задается таблицей

| $A$      | $B$      | $A \vee B$ |
|----------|----------|------------|
| <b>1</b> | <b>0</b> | <b>1</b>   |
| <b>0</b> | <b>1</b> | <b>1</b>   |
| <b>1</b> | <b>1</b> | <b>1</b>   |
| <b>0</b> | <b>0</b> | <b>0</b>   |

**Пример.** Пусть высказывание  $A = \{\text{Число } 10 \text{ кратно } 2\}$ , а высказывание  $B = \{\text{Число } 10 \text{ кратно } 5\}$ . Тогда дизъюнкция заданных высказываний  $A+B = \{\text{Число } 10 \text{ кратно } 2 \text{ или } 5\}$ .

Обоснованием такого способа определения (задания) операции **дизъюнкции** является то, что согласно интуитивному пониманию союза **или**, составное высказывание типа «**A или B**» ложно тогда и только тогда, когда ложны оба составляющие его высказывания, на что и указывает последняя строка таблицы. В остальных случаях дизъюнкция двух высказываний истинна.

Приведенное определение операции **дизъюнкции** соответствует употреблению союза **или** в русском языке в так называемом **соединительном смысле**. Но часто этот союз употребляется в **разделительном смысле**, то есть понимается как «**либо A, либо B**, но не то и другое вместе». Такому пониманию союза **или** отвечает следующая операция **строгой дизъюнкции**.

**Определение.** *Строгой дизъюнкцией высказываний A, B называется новое высказывание, являющееся истинным лишь в случае, когда A, B имеют противоположный смысл, и ложным во всех остальных случаях.*

Символически строгая дизъюнкция записывается в виде  $A \oplus B$ .

Строгая дизъюнкция  $A \oplus B$  читается «**либо A, либо B**» и задается таблицей

| A | B | $A \oplus B$ |
|---|---|--------------|
| 1 | 0 | 1            |
| 0 | 1 | 1            |
| 1 | 1 | 0            |
| 0 | 0 | 0            |

**Определение.** *Конъюнкцией высказываний A, B называется новое высказывание, являющееся истинным лишь в случае истинности обоих высказываний A, B и ложным во всех остальных случаях.*

Символически конъюнкция записывается в виде  $A \wedge B$ ,  $A \& B$ ,  $A \cdot B$ . Конъюнкция соответствует союзу «и». Конъюнкция  $A \wedge B$  читается A и B.

Конъюнкция задается таблицей

| A | B | $A \wedge B$ |
|---|---|--------------|
| 1 | 0 | 0            |
| 0 | 1 | 0            |
| 1 | 1 | 1            |
| 0 | 0 | 0            |

**Пример.** Пусть высказывание  $A = \{\text{Число } 10 \text{ кратно } 2\}$ , а высказывание  $B = \{\text{Число } 10 \text{ кратно } 5\}$ . Тогда конъюнкция заданных высказываний  $A \wedge B = \{\text{Число } 10 \text{ кратно } 2 \text{ и } 5\}$ .

Обоснованием такого способа определения (задания) операции **конъюнкции** является то, что согласно интуитивному пониманию союза **и**, составное высказывание типа «**A и B**» истинно тогда и только тогда, когда истинны оба составляющих его высказывания, на что и указывает предпоследняя строка таблицы. В остальных случаях конъюнкция двух высказываний ложна. Иногда знак конъюнкции между высказывания опускают, подобно тому, как в обычной алгебре часто опускают знак операции умножения.

**Определение.** *Импликацией высказываний A, B называется новое высказывание, являющееся ложным лишь в случае, когда A – истинно, а B – ложно истинным во всех остальных случаях.*

Символически импликация записывается в виде  $A \rightarrow B$  или  $A \supset B$ . Импликация соответствует словосочетанию союзу «*если ..., то ...*». Импликация  $A \rightarrow B$  читается **если A, то B**.

Импликация задается таблицей

| A | B | $A \rightarrow B$ |
|---|---|-------------------|
| 1 | 0 | 0                 |
| 0 | 1 | 1                 |
| 1 | 1 | 1                 |
| 0 | 0 | 1                 |

Определение **импликации** весьма условно можно считать формализацией словосочетания «*если ..., то ...*». Дело в том, что словосочетание «*если ..., то ...*» выражает в языке не только логическую, но и причинно-следственную связь, которую материальная

импликация выразить не может. И, тем не менее, это определение в значительной степени соответствует интуитивному пониманию словосочетания «*если ..., то ...*» в смысле логического следования. По крайней мере, высказывание, являющееся импликацией двух высказываний, ложно в том и только том случае, если мы из истины пытаемся сделать (или, как говорят, имплицировать, вывести) ложное заключение (первая строка таблицы).

Словосочетанию «*...тогда и только тогда, когда ...*» (синонимы: «*... если и только если ...*», «*... эквивалентно...*», «*... необходимо и достаточно для ...*») соответствует логическая операция, называемая эквиваленцией и обозначаемая символом  $\sim$ .

*Эквиваленция* задается следующей таблицей:

| <b>A</b> | <b>B</b> | <b>A~B</b> |
|----------|----------|------------|
| <b>Л</b> | <b>Л</b> | <b>И</b>   |
| <b>Л</b> | <b>И</b> | <b>Л</b>   |
| <b>И</b> | <b>Л</b> | <b>Л</b>   |
| <b>И</b> | <b>И</b> | <b>И</b>   |

То есть, *эквиваленция* двух высказываний истинна тогда и только тогда, когда высказывания либо оба ложны, либо оба истинны.

Примером эквиваленции двух высказываний является высказывание «Четырехугольник является квадратом тогда и только тогда, когда все его стороны и углы равны между собой».

Как мы увидим далее, операции строгой дизъюнкции, импликации, эквиваленции могут быть выражены через операции конъюнкции, дизъюнкции и отрицания, поэтому они являются как бы избыточными. Более того, дизъюнкцию можно выразить через конъюнкцию и отрицание, а конъюнкцию – через дизъюнкцию и отрицание / Немецкий логик Шеффер предложил вообще одну-единственную логическую операцию, через которую можно выразить все остальные. Она получила название «штрих Шеффера». То же самое сделал английский логик Пирс, предложив еще одну универсальную операцию – «стрелку Пирса». Их определение и применение мы рассмотрим позднее/. Отмеченная избыточность не является, вообще говоря, недостатком логики высказываний, она позволяет в более естественном виде осуществлять формализацию высказываний и рассуждений.

### 3. Свойства логических операций

Рассматриваемые нами логические операции над высказываниями обладают рядом свойств, которые удивительным образом напоминают нам соответствующие свойства операций над множествами в интуитивной теории множеств.

Свойства мы зададим в виде так называемых логических равенств. Мы будем понимать логическое равенство как утверждение логической равносильности, (логической эквивалентности) двух высказываний. Логическая равносильность высказываний означает, что значения истинности этих высказываний совпадают /Ниже мы дадим более формальное (более точное) определение логической равносильности двух высказываний./. Для обозначения логической равносильности двух высказываний будем использовать символ  $\equiv$ . Приведем здесь лишь свойства основных логических операций: конъюнкции, дизъюнкции и отрицания.

#### *Свойства коммутативности*

$A \& B \equiv B \& A$  – коммутативность конъюнкции;

$A \vee B \equiv B \vee A$  – коммутативность дизъюнкции;

#### *Свойства ассоциативности*

$A \& (B \& C) \equiv (A \& B) \& C$  – ассоциативность конъюнкции;

$A \vee (B \vee C) \equiv (A \vee B) \vee C$  – ассоциативность дизъюнкции;

#### *Свойства дистрибутивности*

$A \& (B \vee C) \equiv (A \& B) \vee (A \& C)$  – дистрибутивность конъюнкции относительно дизъюнкции:

$A \vee (B \& C) \equiv (A \vee B) \& (A \vee C)$  – дистрибутивность дизъюнкции относительно конъюнкции:

#### *Свойства логических констант*

свойства константы И:

$$A \& I \equiv A, \quad A \vee I \equiv A;$$

свойства константы Л:

$$A \& L \equiv L, \quad A \vee L \equiv A.$$

**Законы Де Моргана:**

$$\neg(A \& B) \equiv \neg A \vee \neg B,$$

$$\neg(A \vee B) \equiv \neg A \& \neg B.$$

**Закон исключенного третьего** (*tertium non datur* - третьего не дано):

$$\neg A \vee A \equiv \text{И}.$$

**Закон противоречия:**

$$\neg A \& A \equiv \text{Л}.$$

**Закон снятия двойного отрицания:**

$$\neg \neg A \equiv A.$$

**Законы идемпотентности**

$A \& A \equiv A$  – идемпотентность конъюнкции:

$A \vee A \equiv A$  – идемпотентность дизъюнкции:

**Законы поглощения**

$$A \& (A \vee B) \equiv A,$$

$$A \vee (A \& B) \equiv A.$$

## Лекция 4. Формулы логики высказываний

Учебные вопросы:

1. Понятие формулы логики высказываний и ее логический смысл. Приоритет логических операций.
2. Вычисление значений истинности формул логики высказываний.
3. Тавтологично-истинные и тавтологично-ложные формулы логики высказываний. Логическая равносильность формул.

### 1. Понятие формулы логики высказываний и ее логический смысл. Приоритет логических операций

*Переменную, которая может принимать значения конкретных высказываний, будем называть пропозициональной переменной. Логические константы 1, 0 будем называть пропозициональными константами.*

Истинностными значениями пропозициональных переменных являются пропозициональные константы 1, 0.

Пропозициональные переменные будем обозначать буквами конца латинского алфавита  $X, Y, Z$  (возможно с индексами:  $X_1, X_2, X_3$  и так далее).

Индуктивное определение формулы логики высказываний:

1. *Всякая пропозициональная константа и переменная есть формула логики высказываний.*
2. *Если  $F, \Phi$  – формулы логики высказываний, то следующие последовательности символов также будут формулами логики высказываний:  
 $(\neg F), (F \& \Phi), (F \vee \Phi), (F \oplus \Phi), (F \rightarrow \Phi), (F \sim \Phi).$*
3. *Те и только те последовательности символов будут формулами логики высказываний, для которых это следует из пп.1 и 2 данного определения /  $F$  и  $\Phi$  называют в этом случае подформулами формулы логики высказываний /.*
4. *Истинностным значением (или просто значением) формулы логики высказываний является значение истинности, получаемое при вычислении результатов всех логических*



*операций, с помощью которых строится формула, при той или иной комбинации значений пропозициональных переменных и констант, входящих в формулу.*

При вычислении значения формулы мы будем руководствоваться (как и в школьной алгебре) круглыми скобками (,) и следующим приоритетом (старшинством) операций:

- отрицание ( $\neg$ ),
- конъюнкция ( $\&$ ),
- дизъюнкция ( $\vee$ ), строгая дизъюнкция ( $\oplus$ ),
- импликация ( $\rightarrow$ ),
- эквиваленция ( $\sim$ ).

Операции перечислены в порядке убывания приоритета: отрицание ( $\neg$ ) имеет самый высокий приоритет, а эквиваленция ( $\sim$ ) – самый низкий. Старшинство операций учитывается, если скобки не определяют однозначно порядок вычисления.

## **2. Вычисление значений истинности формул логики высказываний**

Покажем на примерах, как вычисляется значение истинности формул логики высказываний при заданных значениях истинности входящих в формулу пропозициональных переменных и констант. Для этого воспользуемся универсальным для логики высказываний методом – **методом истинностных таблиц**.

**Пример 1.** Вычислить значение истинности формулы  $\neg((X\&Y\rightarrow\neg Y)\rightarrow\neg X)$  при следующих значениях пропозициональных переменных:  $X = 1$ ,  $Y = 0$ .

Прежде всего, заметим, что порядок вычисления значения истинности этой формулы определяется частично скобками, а частично старшинством операций. Этот порядок и вычисления в соответствии с ним представлены в таблице 1.

Результирующее значение 1 представлено в последней строке в столбце, соответствующем последней выполняемой операции отрицания  $\neg$  (в выделенной жирной линией клетке).

Предпоследнюю и последнюю строки в дальнейшем будем объединять в одну для того, чтобы сократить размеры таблицы 1.

Таблица 1

|                   |        |   |   |   |   |   |               |        |   |   |               |        |   |   |
|-------------------|--------|---|---|---|---|---|---------------|--------|---|---|---------------|--------|---|---|
| Порядок операций  | 6      |   |   |   | 3 |   | 4             | 1      | ) |   | 5             | 2      |   |   |
| Формула           | $\neg$ | ( | ( | X | & | Y | $\rightarrow$ | $\neg$ | Y | ) | $\rightarrow$ | $\neg$ | X | ) |
| Заданные значения |        |   |   | 1 |   | 0 |               |        | 0 |   |               |        | 1 |   |
| Результат         | 1      |   |   |   | 0 |   | 1             | 1      |   |   | 0             | 0      |   |   |

**Пример 2.** Вычислить значение истинности высказывания «*Если* огород зарос бузиной *или* дядя уехал в Киев, *то* дважды два – пять» при различных значениях истинности входящих в него простых высказываний.

Представим приведенное высказывание в виде формулы логики высказываний. Легко сообразить, что это будет формула  $A \vee B \rightarrow 1$ , где **A** – высказывание «Огород зарос бузиной», **B** – высказывание «Дядя уехал в Киев», 0 – ложное высказывание «Дважды два – пять».

Всевозможные значений для высказываний **A** и **B** и результаты вычислений для каждой комбинации значений представлены в следующей таблице 2.

Таблица 2

|   |   |   |   |        |   |               |   |
|---|---|---|---|--------|---|---------------|---|
| Порядок операций  |   |   |   | 1      |   | 2             |   |
| Простые высказывания и формула  | A | B | A | $\vee$ | B | $\rightarrow$ | 0 |
| Комбинации значений истинности простых высказываний и результаты вычислений | 0 | 0 | 0 | 0      | 0 | 1             | 0 |
|   | 0 | 1 | 0 | 1      | 1 | 0             | 0 |
|   | 1 | 0 | 1 | 1      | 0 | 0             | 0 |
|   | 1 | 1 | 1 | 1      | 1 | 0             | 0 |

В выделенном столбце представлен результат вычислений для каждой комбинации значений истинности простых высказываний **A** и **B**. Он показывает, что анализируемое высказывание ложно тогда и только тогда, когда оба простых высказывания **A** и

**В** ложны.

**Пример 3.** Построить таблицу значений истинности высказывания

$A \rightarrow B \sim (\neg A \vee B)$  при всех возможных комбинациях истинности составляющих его простых высказываний

На этот раз мы упростим таблицу истинности, убрав из нее все пояснения (за исключением, пока еще, порядка выполнения операций) и выделив результирующий столбец. Надеемся, что читатель уже понял структуру таблиц истинности. Итак, имеем таблицу истинности (таблица 3).

Таблица 3

|   |   |               |   |        |   |        |   |        |   |   |
|---|---|---------------|---|--------|---|--------|---|--------|---|---|
| Порядок операций  |   | 3             |   | 4      |   | 1      |   | 2      |   |   |
| Простые высказывания и формула  | A | $\rightarrow$ | B | $\sim$ | ( | $\neg$ | A | $\vee$ | B | ) |
| Комбинации значений истинности простых высказываний и результаты вычислений | 0 | 1             | 0 | 1      |   | 1      | 0 | 1      | 0 |   |
|   | 0 | 1             | 1 | 1      |   | 1      | 0 | 1      | 1 |   |
|   | 1 | 0             | 0 | 1      |   | 0      | 1 | 0      | 0 |   |
|   | 1 | 1             | 1 | 1      |   | 0      | 1 | 1      | 1 |   |

Из таблицы ясно, что формула принимает значение 1, независимо от комбинации значений истинности, входящий в нее высказываний.

**Пример 4.** Построить таблицу истинности формулы логики высказываний  $\neg(((A \rightarrow B) \& (B \rightarrow C)) \rightarrow (A \rightarrow C))$  при любых комбинациях значений истинности высказываний A, B, C. Результат представлен в таблице 4.

Необходимо обратить внимание на порядок перечисления наборов значений истинности для всех пропозициональных переменных, входящих в формулу, чтобы избежать либо пропуска, либо повторения тех или иных наборов.

Таблица 4

| $\neg$ | ( | ( | ( | A | $\rightarrow$ | B | ) | & | ( | B | $\rightarrow$ | C | ) | ) | $\rightarrow$ | ( | A | $\rightarrow$ | C | ) | ) |
|--------|---|---|---|---|---------------|---|---|---|---|---|---------------|---|---|---|---------------|---|---|---------------|---|---|---|
| 0      |   |   |   | 0 | 1             | 0 |   | 1 |   | 0 | 1             | 0 |   |   | 1             |   | 0 | 1             | 0 |   |   |
| 0      |   |   |   | 0 | 1             | 0 |   | 1 |   | 0 | 1             | 1 |   |   | 1             |   | 0 | 1             | 1 |   |   |
| 0      |   |   |   | 0 | 1             | 1 |   | 0 |   | 1 | 0             | 0 |   |   | 1             |   | 0 | 1             | 0 |   |   |
| 0      |   |   |   | 0 | 1             | 1 |   | 1 |   | 1 | 1             | 1 |   |   | 1             |   | 0 | 1             | 1 |   |   |
| 0      |   |   |   | 1 | 0             | 0 |   | 0 |   | 0 | 1             | 0 |   |   | 1             |   | 1 | 0             | 0 |   |   |
| 0      |   |   |   | 1 | 0             | 0 |   | 0 |   | 0 | 1             | 1 |   |   | 1             |   | 1 | 1             | 1 |   |   |
| 0      |   |   |   | 1 | 1             | 1 |   | 0 |   | 1 | 0             | 0 |   |   | 1             |   | 1 | 0             | 0 |   |   |
| 0      |   |   |   | 1 | 1             | 1 |   | 1 |   | 1 | 1             | 1 |   |   | 1             |   | 1 | 1             | 1 |   |   |

### 3. Тавтологично-истинные и тавтологично-ложные формулы логики высказываний. Логическая равносильность формул

Формулу логики высказываний, принимающую значение истинности 1 (истина) на любом наборе значений для пропозициональных переменных, входящих в формулу, называют тавтологично-истинной формулой, или тавтологией.

Формулу логики высказываний, принимающую значение истинности 0 (ложь) на любом наборе значений для пропозициональных переменных, входящих в формулу, называют тавтологично-ложной формулой, или противоречием.

Формулу логики высказываний, не являющуюся ни тавтологично-истинной, ни тавтологично ложной, называют выполнимой.

Пусть формулы  $F$  и  $\Phi$  логики высказываний содержат пропозициональные переменные  $X_1, X_2, \dots, X_n$ . Будем считать эти формулы логически равносильными, если они принимают одинаковые значения истинности на соответствующих наборах значений для пропозициональных переменных  $X_1, X_2, \dots, X_n$ , входящих в эти формулы.

Если множества пропозициональных переменных, входящих в формулы  $F$  и  $\Phi$  не совпадают, то можно добиться этого совпадения, введя в ту или другую формулу недостающую переменную в качестве "фиктивной". Пусть, например, формула  $F$  не содержит пропозициональной переменной  $X_i$ . Тогда эту переменную можно ввести в формулу  $F$  "фиктивно", заменив формулу  $F$  на формулу  $F \vee (X_i \& \neg X_i)$  или на формулу  $F \& (X_i \vee \neg X_i)$ , которые на основании закона противоречия, закона исключенного третьего и свойств логических констант 1 и 0, равносильны  $F$ . Аналогично можно "фиктивно" ввести в формулы  $F$  и  $\Phi$  все другие недостающие переменные. Это соображение легко распространить на любое число формул.

Как мы условились выше, тот факт, что формулы  $F$  и  $\Phi$  логически равносильны будем обозначать  $F \equiv \Phi$ .

Отношение равносильности формул, очевидно, обладает свойством *транзитивности*: если  $F \equiv \Phi$  и  $\Phi \equiv \Psi$ , то  $F \equiv \Psi$ .

Приведенные выше свойства операций и законы логики высказываний, как легко проверить с помощью таблиц истинности, выражают логическую равносильность (эквивалентность) тех или иных формул.

Кроме приведенных выше равносильностей в логике высказываний большое значение имеют и другие, среди которых отметим следующие:

1.  $X \oplus Y \equiv Y \oplus X$  – коммутативность строгой дизъюнкции.
2.  $X \oplus (Y \oplus Z) \equiv (X \oplus Y) \oplus Z$  – ассоциативность строгой дизъюнкции.
3.  $X \rightarrow Y \equiv \neg Y \rightarrow \neg X$  – закон контрапозиции.
4.  $(X \rightarrow Y) \& (Y \rightarrow Z) \rightarrow (X \rightarrow Z)$  – транзитивность импликации.
5.  $X \rightarrow Y \equiv \neg X \vee Y = X \& \neg Y$  – выражение импликации через дизъюнкцию и отрицание ( $\neg$ ).
6.  $(X \sim Y) \equiv (X \rightarrow Y) \& (Y \rightarrow X) = X \& Y \vee \neg X \& \neg Y$  – выражение эквиваленции через импликацию и конъюнкцию.
7.  $X \vee Y \equiv \neg(\neg X \& \neg Y)$  – выражение дизъюнкции через конъюнкцию.

*юнкцию и отрицание.*

**8.  $X \& Y \equiv \neg(\neg X \vee \neg Y)$  – выражение конъюнкции через дизъюнкцию и отрицание.**

Логические равносильности играют важную роль в логике высказываний. Они фактически являются правилами и законами логических рассуждений, **законами правильного мышления** /Естественно, логикой высказываний не исчерпывается все многообразие логических рассуждений. Кроме логики высказываний, важное значение имеют логика предикатов, модальная логика, нормативная логика, временная логика, многозначная логика, нечеткая логика и т.д./.

## Лекция 5. Применение логики высказываний в математических доказательствах

### Учебные вопросы:

1. Доказательства прямые и от противного.
2. Доказательство приведением к абсурду. Необходимые и достаточные условия.

### 1. Доказательства прямые и от противного

Рассмотрим с точки зрения логики высказываний наиболее типичные методы доказательств в математике.

#### 1.1. Доказательство с помощью построения цепочки импликаций.

Этим методом пользуются при доказательстве теорем, выраженных в форме импликации: "Если высказывание **A** истинно, то и высказывание **B** истинно", то есть  $A \rightarrow B$ . Доказательство строится как последовательность тождественно-истинных импликаций вида:  $A \rightarrow A_1, A_1 \rightarrow A_2, \dots, A_{n-1} \rightarrow A_n, A_n \rightarrow B$ , где  $A_1, A_2, A_3, \dots, A_n$  - некоторые вспомогательные высказывания. Отсюда делается вывод (в силу транзитивности импликации) о справедливости теоремы  $A \rightarrow B$ .

Такое доказательство называется прямым доказательством. Напомним классификацию теорем из средней школы (рис.1)

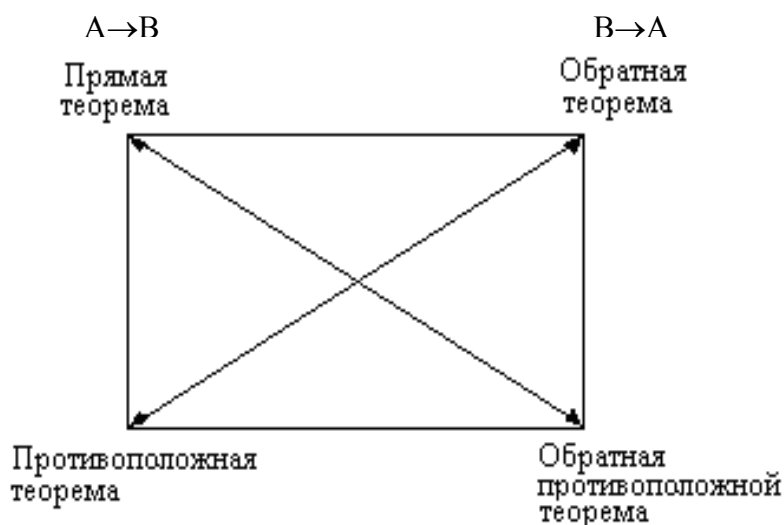


Рис.1. Взаимосвязь между прямой и обратной, противоположной и обратной противоположной теоремами

Как легко проверить, используя метод истинностных таблиц, прямая теорема оказывается равносильной обратной противоположной, а обратная теорема - противоположной. И в то же время таких равносильностей в общем случае не существует между прямой и обратной теоремами, между прямой и противоположной, между обратной и обратной противоположной, между противоположной и обратной противоположной.

### **1.2. Доказательство от противного**

Этот метод используется при доказательстве теорем вида  $A \rightarrow B$  и основывается на законе контрапозиции  $X \rightarrow Y \equiv \neg Y \rightarrow \neg X$ , который фактически гласит, что доказательство теоремы  $A \rightarrow B$  может быть заменено доказательством эквивалентной ей теоремы, которая формулируется как  $\neg B \rightarrow \neg A$ . Последняя теорема называется *обратная противоположной (или противоположная обратной)*.

Из указанных равносильностей вытекает следующий метод доказательства.

Доказательство теоремы  $\neg B \rightarrow \neg A$  осуществляется прямым путем, то есть как цепочка импликаций:  $\neg B \rightarrow B_1, B_1 \rightarrow B_2, \dots, B_{n-1} \rightarrow B_n, B_n \rightarrow \neg A$ , из которой делается вывод (в силу транзитивности импликации) о справедливости теоремы  $\neg B \rightarrow \neg A$ . А в силу закона контрапозиции заключается о справедливости теоремы  $A \rightarrow B$ .

## **2. Доказательство приведением к абсурду. Необходимые и достаточные условия**

### **2.1. Доказательство приведением к абсурду**

Пусть требуется доказать истинность некоторого утверждения  $A$ . Предположим, что  $A$  ложно, тогда  $\neg A$  - истинно, поскольку закон противоречия ( $X \& \neg X \equiv 0$ ), имеющий место в логике высказываний, означает, что одновременно не могут быть истинными утверждение и его отрицание.

После этого показывается, что тогда имеется некоторое утверждение  $B$  такое, что истинными являются одновременно два утверждения:  $\neg A \rightarrow B$  и  $\neg A \rightarrow \neg B$  /Как такое утверждение  $B$  найти или построить - это и есть часть доказательства, зачастую носящая эвристический, то есть поисковый, характер и относится к содер-



жанию математической дисциплины, теорема из которой доказывается. Для нас важна **форма** доказательства, то есть, его **структура**./ Это и есть то, что называют **абсурдом**.

В логике высказываний тождественно-истинной является формула:  $(\neg A \rightarrow B) \& (\neg A \rightarrow \neg B) \rightarrow A$  (проверку чего мы предоставляем читателю).

Из этой формулы и  $(\neg A \rightarrow B) \& (\neg A \rightarrow \neg B)$  по правилу вывода *modus ponens* следует, что имеет место утверждение **A**.

## 2.2. Доказательство необходимых и достаточных условий

В математике часто встречаются теоремы вида: "Условие **A** равносильно условию **B**", что также выражается словами: "Для того, чтобы имело место условие **A**, необходимо и достаточно, чтобы выполнялось условие **B**". В виде формулы логики высказываний такая теорема может быть записана в виде:  $A \sim B$ . Доказательство ее обычно сводится к доказательству двух утверждений:

1.  $A \rightarrow B$  (Если имеет место условие **A**, то выполняется и условие **B**)

2.  $B \rightarrow A$  (Если имеет место условие **B**, то выполняется и условие **A**). /Как известно, формула  $A \sim B$  эквивалентна конъюнкции  $(A \rightarrow B) \& (B \rightarrow A)$ ./

Первое условие называют **необходимым** (то есть, **B** необходимо для **A**), а второе условие - **достаточным** (то есть, **A** достаточно для **B**). По-другому, первое называют **прямой** теоремой, а второе - **обратной**.

Доказательство и прямой, и обратной теорем может быть осуществлено любым из трех приведенных выше способов. После чего, можно утверждать и справедливость теоремы "Условие **A** равносильно условию **B**".

Существует и другой способ доказательства теорем вида: "Условие **A** равносильно условию **B**", когда одновременно доказываются необходимость и достаточность условия **B** для **A**. Для этого находится последовательность тождественно-истинных эквиваленций вида:  $A \sim A_1, A_1 \sim A_2, \dots, A_{n-1} \sim A_n, A_n \sim B$ , где  $A_1, A_2, A_3, \dots, A_n$  - некоторые вспомогательные высказывания.

Отсюда делается вывод (в силу транзитивности эквиваленции) о справедливости теоремы  $A \sim B$ .

Наконец, доказательство теоремы вида  $A \sim B$  можно заменять доказательством равносильной ей противоположной теоремы  $\neg A \sim \neg B$ . (В равносильности этих теорем легко убедиться с помощью таблиц истинности.)

Изучая доказательство любой теоремы, прежде всего, необходимо выяснить *структуру* доказательства, постараться отнести его к одному из рассмотренных четырех видов, а уже затем изучать само доказательство, четко представляя ту идею, которую применил автор теоремы для того, чтобы ее доказать.

## Лекция 6. Применение логики высказываний к анализу и синтезу переключательных (контактных) схем

### Учебные вопросы:

1. Представление законов логики высказываний переключательными схемами.
2. Примеры переключательных схем.

1. Представление законов логики высказываний переключательными схемами

Переключательной (или контактной) схемой мы называется участок электрической цепи, включающий ряд переключателей (контактных выключателей), подобный приведенному на рис.1.

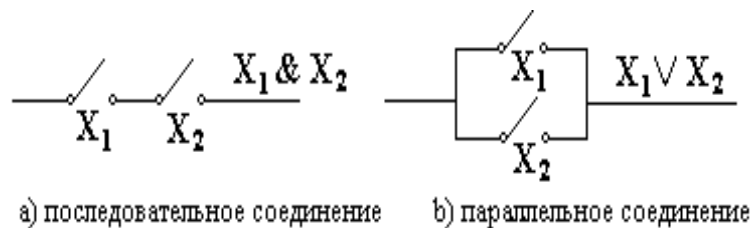


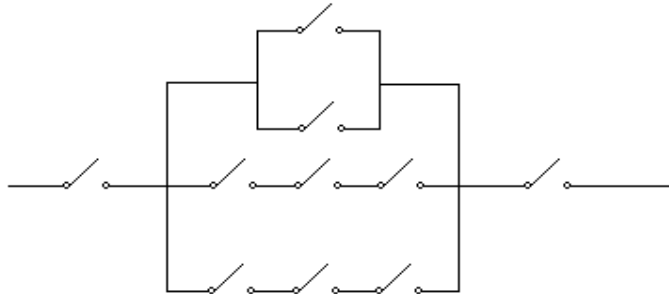
Рис.1. Вид переключательной схемы

Каждому переключателю схемы сопоставим пропозициональную переменную  $X_i$ , которая будет принимать значение 1 (истина) или 0 (ложь), если соответствующий переключатель замкнут (то есть, проводит электрический ток) или разомкнут (то есть, не проводит электрический ток).

Поскольку функция участка электрической цепи состоит в том, чтобы проводить электрический ток, то два участка, содержащие одни и те же переключатели и проводящие или не проводящие ток при одном и том же состоянии всех выключателей ("замкнут" или "разомкнут"), мы будем считать "равными" и не различать между собой.

Последовательное соединение двух переключателей  $X_1$  и  $X_2$  в цепи будет представлять собой **конъюнкция** пропозициональных переменных  $X_1$  и  $X_2$  логики высказываний, т.е.  $X_1 \& X_2$  (это означа-

ет, что такой участок проводит ток тогда и тогда, когда оба переключателя  $X_1$  и  $X_2$  замкнуты). Параллельное соединение двух переключателей  $X_1$  и  $X_2$  в цепи соответствует **дизъюнкции** пропозициональных переменных  $X_1$  и  $X_2$ , то есть  $X_1 \vee X_2$  (это означает, что такой участок проводит ток тогда и тогда, когда хотя бы один из переключателей  $X_1$ ,  $X_2$  замкнут). Сказанное представлено на рис.2.



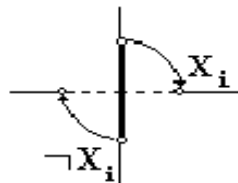
**Рис. 2. Соответствие формул логики высказываний видам соединения переключателей**

Условимся обозначать через 1 всегда замкнутый контакт, а через 0 - всегда разомкнутый. На схемах это будет выглядеть так, как представлено на рис.3.



**Рис.3. Соответствие логических констант всегда замкнутому и всегда разомкнутому контактам**

Условимся, наконец, обозначать через  $X_i$  и  $\neg X_i$  такую пару контактов, что когда контакт  $X_i$  замкнут, контакт  $\neg X_i$  обязательно разомкнут, и наоборот. Техническое осуществление такой пары контактов показано на рис.4.



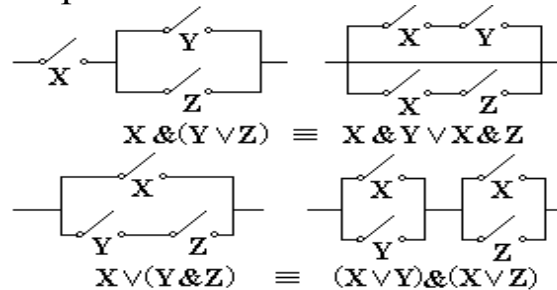
**Рис.4. Реализация контактов  $X_i$  и  $\neg X_i$**

Ясно, что параллельное и последовательное соединение переключательных схем обладает свойствами коммутативности, ассоциативности, идемпотентности.

Несколько сложнее проверяется выполнимость двух законов дистрибутивности:

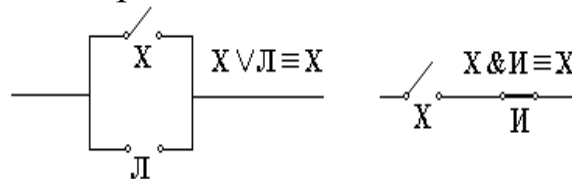
$$X \& (Y \vee Z) \equiv (X \& Y) \vee (X \& Z) \text{ и } X \vee (Y \& Z) \equiv (X \vee Y) \& (X \vee Z).$$

На рис.5 приведены попарно эквивалентные переключательные схемы, подтверждающие справедливость указанных законов дистрибутивности для переключательных схем.



**Рис.5. Техническая иллюстрация законов дистрибутивности  $\vee$  и  $\&$**

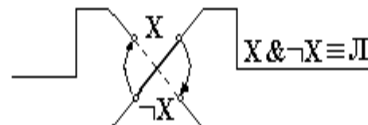
Рис.6–10 иллюстрируют другие законы логики высказываний, выраженные на "языке" переключательных схем.



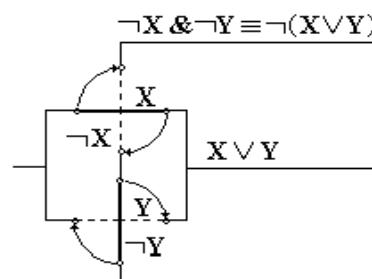
**Рис.6. Свойства логических констант**



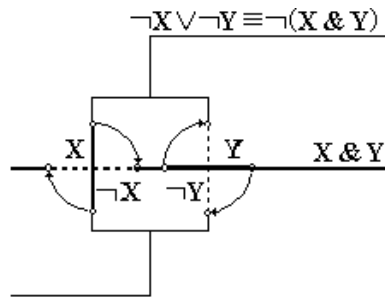
**Рис.7. Закон исключенного третьего**



**Рис. 8. Закон противоречия**



**Рис. 9. Первый закон Де Моргана**



**Рис.10. Второй закон Де Моргана**

Таким образом, все законы логики высказываний имеют аналогии в логике переключательных схем. Это позволяет *моделировать* сложные высказывания с помощью электрических цепей, *конструировать* (*синтезировать*) переключательные схемы, удовлетворяющие наперед заданным условиям (которые могут быть и достаточно сложными).

## 2. Примеры переключательных схем

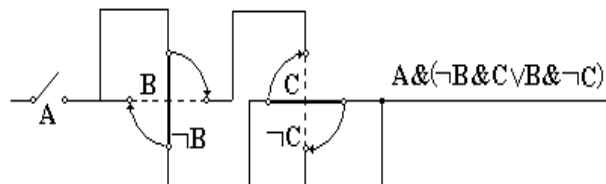
Приведем пример моделирования сложного высказывания переключательной схемой. Пусть дано высказывание

$$A \& \neg B \& C \vee A \& B \& \neg C.$$

Перед тем, как построить соответствующую этому высказыванию переключательную схему, преобразуем данную формулу логики высказываний в равносильную ей более простую формулу. Используя закон дистрибутивности  $\&$  относительно  $\vee$ , получим формулу, эквивалентную исходной

$$A \& (\neg B \& C \vee B \& \neg C).$$

Последней формуле отвечает переключательная схема, представленная на рис.11.



**Рис.11. Переключательная схема, отвечающая формуле**

$$A \& (\neg B \& C \vee B \& \neg C)$$

Покажем теперь на примерах, как по словесному описанию можно синтезировать переключательную схему, используя аппарат логики высказываний.

**Пример 1.** Пусть требуется спроектировать электрическую цепь для спальни с одной электрической лампочкой, где желательно иметь два выключателя: один у двери, а второй - над постелью; при этом поворот каждого выключателя независимо от состояния второго выключателя должен размыкать цепь, если до этого она была замкнута, и замыкать, если ранее она была разомкнута.

**Решение.** Обозначим выключатели пропозициональными переменными  $X$  и  $Y$ . Значениями переменных могут быть 1 (выключатель замыкает цепь) или 0 (выключатель размыкает цепь). Обозначим искомую формулу логики высказываний буквой  $F$ . Составим таблицу истинности формулы  $F$ , исходя из условия задачи.

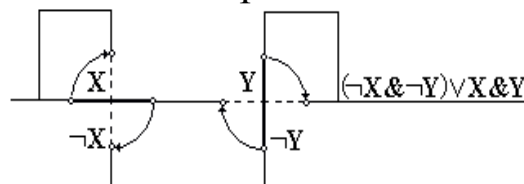
Пусть значение 1 этой формулы означает, что цепь замкнута (лампочка горит), а 0 - цепь разомкнута (лампочка не горит).

Итак, пусть выключатели  $X$  и  $Y$  оба замыкают цепь, то есть имеют значение 1. Тогда в этом случае и значение  $F$  есть 1 (лампочка горит). Изменение значения ровно одного (любого) из выключателей на 0 меняет значение  $F$  также на 0. Если же одновременно изменить на 0 значение каждого из двух выключателей  $X$  и  $Y$ , то значение  $F$  останется прежним, т.е. 1.

Результаты наших рассуждений можно представить в виде следующей таблицы истинности

| X | Y | F |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Эта таблица соответствует определению операции эквиваленции ( $\sim$ ), то есть  $F \equiv X \sim Y$ . Выражая теперь эквиваленцию через конъюнкцию, дизъюнкцию и отрицание, получаем:  $F \equiv \neg X \& \neg Y \vee X \& Y$ . Техническая реализация схемы дана на рис.12.



**Рис.12.** Схема, реализующая формулу  $\neg X \& \neg Y \vee X \& Y$

**Пример 2.** Требуется сконструировать систему тайного голосования, в которой используются кнопочные выключатели и лампочка. Голосуют  $n$  человек. Лампочка должна загораться, если кнопки нажимают  $[n/2]+1$  человек. (Квадратные скобки обозначают взятие целой части дроби  $n/2$ .)

Для простоты возьмем  $n=3$ . В таком случае лампочка должна загораться, если кнопки нажмут не менее 2-х человек.

**Решение.** Пусть  $X_1, X_2, X_3$  - участники голосования,  $F$  - лампочка. Если участник голосования с номером  $i$  нажимает кнопку, то значением переменной  $X_i$  будет **1**, в противном случае - **0**. Если лампочка загорается, то значит значение  $F$  равно **1**, в противном случае - **0**.

Построим таблицу истинности искомой схемы:

| $X_1$ | $X_2$ | $X_3$ | $F$ |
|-------|-------|-------|-----|
| 0     | 0     | 0     | 0   |
| 0     | 0     | 1     | 0   |
| 0     | 1     | 0     | 0   |
| 0     | 1     | 1     | 1   |
| 1     | 0     | 0     | 0   |
| 1     | 0     | 1     | 1   |
| 1     | 1     | 0     | 1   |
| 1     | 1     | 1     | 1   |

Теперь запишем формулу логики высказываний, соответствующую этой таблице истинности.

Прежде заметим, что каждой строке таблицы, в которой  $F = 1$ , соответствует **единственная** конъюнкция переменных  $X_1, X_2, X_3$  или их отрицаний, такая, что при соответствующих значениях истинности этих переменных  $F = 1$ . Так, четвертой строке таблицы соответствует конъюнкция  $\neg X_1 \& X_2 \& X_3$ , которая принимает значение 1 при  $X_1=0, X_2=1, X_3=1$ , шестой строке -  $X_1 \& \neg X_2 \& X_3$ , которая принимает значение 1 при  $X_1=1, X_2=0, X_3=1$ , и так далее. Такие конъюнкции называют **элементарными**.

Выпишем все такие конъюнкции для приведенной выше таблицы:



$$\neg X_1 \& X_2 \& X_3, X_1 \& \neg X_2 \& X_3, X_1 \& X_2 \& \neg X_3, X_1 \& X_2 \& X_3.$$

Очевидно, что искомая формула будет представлять собой дизъюнкцию всех таких конъюнкций:

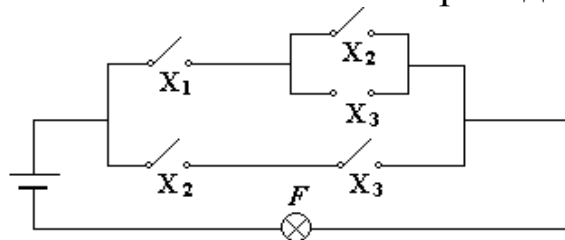
$$F = \neg X_1 \& X_2 \& X_3 \vee X_1 \& \neg X_2 \& X_3 \vee X_1 \& X_2 \& \neg X_3 \vee X_1 \& X_2 \& X_3.$$

Представление формулы логики высказываний в виде дизъюнкции элементарных конъюнкций указанного вида называют совершенной дизъюнктивной нормальной формой (СДНФ).

Выполняя равносильные преобразования этой формулы, получим:

$$\begin{aligned} & \neg X_1 \& X_2 \& X_3 \vee X_1 \& \neg X_2 \& X_3 \vee X_1 \& X_2 \& \neg X_3 \vee X_1 \& X_2 \& X_3 \\ \equiv & \\ \equiv & \\ & \neg X_1 \& X_2 \& X_3 \vee X_1 \& \neg X_2 \& X_3 \vee X_1 \& X_2 \& \neg X_3 \vee X_1 \& X_2 \& X_3 \vee X_1 \& X_2 \& X_3 \\ & \equiv (\neg X_1 \& X_2 \& X_3 \vee X_1 \& X_2 \& X_3) \vee (X_1 \& \neg X_2 \& X_3 \vee X_1 \& X_2 \& X_3) \vee ( \\ & \neg X_3 \vee X_1 \& X_2 \& X_3 \vee X_1 \& X_2 \& X_3) \\ & \equiv X_2 \& X_3 \& (\neg X_1 \vee X_1) \vee X_1 \& X_3 \& (\neg X_2 \vee X_2) \vee X_1 \& X_2 \& (\neg X_3 \vee X_3) \equiv \\ & \equiv X_2 \& X_3 \vee X_1 \& X_3 \vee X_1 \& X_2 \equiv \\ & \equiv X_1 \& (X_2 \vee X_3) \vee X_2 \& X_3. \end{aligned}$$

Техническое решение искомой схемы приведено на рис.13.



**Рис.13. Схема, реализующая формулу  $X_1 \& (X_2 \vee X_3) \vee X_2 \& X_3$**

(Контакт  $X_2$ , который указан дважды на приведенной схеме, обозначает один и тот же контакт. То же самое можно сказать и относительно  $X_3$ .)

## Лекция 7. Булевские функции и булевы алгебры

### Учебные вопросы:

1. Булевские функции.
2. Булевы алгебры.

### 1. Булевские функции

В математике изучают так называемые числовые функции: алгебраическими, тригонометрическими, логарифмическими и т.д. Области их определения и значения представляли собой подмножества множества действительных чисел.

Булевские (логические) функции характеризуются тем, что аргументы и сама функция принимают значения из множества логических констант  $\{1, 0\}$ , т.е. истина, ложь. Булевская функция в общем случае может содержать  $n$  аргументов:  $y=f(x_1, x_2, \dots, x_n)$ .

Как и математические булевские функции могут задаваться: словесно, таблично или аналитически. Мы будем использовать последние два способа задания булевских функций: табличный (в виде таблиц истинности) и аналитический (в виде формул логики высказываний). Одна и та же функция может задаваться по-разному.

#### *1.1. Булевские функции одной переменной*

Булевских функций от одной переменной всего 4. Эти функции и задающие их формулы логики высказываний приведены в следующей таблице:

| $x$         | 0 | 1 | Формулы логики высказываний, задающие функции                 |
|-------------|---|---|---|
| $\varphi_1$ | 0 | 0 | $\varphi_1(x) = 0$ (константа 0)                              |
| $\varphi_2$ | 0 | 1 | $\varphi_2(x) = x$ (совпадает с переменной $x$ )              |
| $\varphi_3$ | 1 | 0 | $\varphi_3(x) = \neg x$ (является отрицанием переменной $x$ ) |
| $\varphi_4$ | 1 | 1 | $\varphi_4(x) = 1$ (константа 1)                              |

#### *Булевские функции двух переменных*

Булевских функций от двух переменных всего насчитывается 16. Все они представлены в следующей таблице:

|                 |   |   |   |   |   |
|-----------------|---|---|---|---|---|
| x               | 0 | 0 | 1 | 1 | Формулы логики высказываний, задающие функции                 |
| y               | 0 | 1 | 0 | 1 |   |
| f <sub>1</sub>  | 0 | 0 | 0 | 0 | f <sub>1</sub> (x,y) = 0 (константа 0)                        |
| f <sub>2</sub>  | 0 | 0 | 0 | 1 | f <sub>2</sub> (x,y) = x&y (конъюнкция)                       |
| f <sub>3</sub>  | 0 | 0 | 1 | 0 | f <sub>3</sub> (x,y) = ¬(x→y) (отрицание импликации)          |
| f <sub>4</sub>  | 0 | 0 | 1 | 1 | f <sub>4</sub> (x,y) = x (совпадает с переменной x)           |
| f <sub>5</sub>  | 0 | 1 | 0 | 0 | f <sub>5</sub> (x,y) = ¬(y→x) (отрицание обратной импликации) |
| f <sub>6</sub>  | 0 | 1 | 0 | 1 | f <sub>6</sub> (x,y) = y (совпадает с переменной y)           |
| f <sub>7</sub>  | 0 | 1 | 1 | 0 | f <sub>7</sub> (x,y) = x⊕y (строгая дизъюнкция)               |
| f <sub>8</sub>  | 0 | 1 | 1 | 1 | f <sub>8</sub> (x,y) = x∨y (дизъюнкция)                       |
| f <sub>9</sub>  | 1 | 0 | 0 | 0 | f <sub>9</sub> (x,y) = ¬x&¬y (конъюнкция отрицаний)           |
| f <sub>10</sub> | 1 | 0 | 0 | 1 | f <sub>10</sub> (x,y) = x~y (эквиваленция)                    |
| f <sub>11</sub> | 1 | 0 | 1 | 0 | f <sub>11</sub> (x,y) = ¬y (отрицание y)                      |
| f <sub>12</sub> | 1 | 0 | 1 | 1 | f <sub>12</sub> (x,y) = y⊃x (обратная импликация)             |
| f <sub>13</sub> | 1 | 1 | 0 | 0 | f <sub>13</sub> (x,y) = ¬x (отрицание x)                      |
| f <sub>14</sub> | 1 | 1 | 0 | 1 | f <sub>14</sub> (x,y) = x⊃y (импликация)                      |
| f <sub>15</sub> | 1 | 1 | 1 | 0 | f <sub>15</sub> (x,y) = ¬(x&y) (отрицание конъюнкции)         |
| f <sub>16</sub> | 1 | 1 | 1 | 1 | f <sub>16</sub> (x,y) = 1 (константа 1)                       |

Естественно, многие из перечисленных функций могут быть заданы другими, но равносильными формулами логики высказываний.

### 1.3. Булевские функции *n* переменных

Областью определения такой булевой функции будет *n*-тая декартова степень множества {0,1}, то есть всевозможные двоичные наборы длины *n* вида  $\langle \alpha_1 \alpha_2 \dots \alpha_n \rangle$ , где  $\alpha_i \in \{0,1\}$ . Число таких всевозможных наборов (*n*-ок) составляет  $2^n$ .

Область значений булевой функции от *n* переменных - это множество {0,1}.

В дальнейшем мы будем рассматривать только всюду определенные булевские функции, то есть область определения таких функций совпадает с *n*-той декартовой степенью множества {0,1}.

Булевские функции от большего числа переменных могут быть так же заданы таблично, или с помощью формул логики высказы-

ваний, или в виде суперпозиции (взаимной подстановки) булевских функций одной и/или двух переменных.

Например, булевская функция  $w=f(x,y,z)$ , задаваемая формулой логики высказываний  $x \& y \vee \neg x \& \neg y \vee z$  может быть задана в виде следующей суперпозиции функций от одной и двух переменных:  $w=f_8(f_8(f_2(x,y), f_9(x,y)), \varphi_2(z))$ .

Учитывая принципиальную возможность выразить булевскую функцию от любого числа переменных в виде суперпозиции (взаимной подстановки) булевских функций от одной и двух переменных, мы чаще будем использовать либо табличный способ задания таких функций, либо с помощью формул логики высказываний.

Из комбинаторики известно, что число наборов истинностных значений, на которых определена булевская функция от  $n$  переменных, составляет  $2^n$  (при  $n=1$  это число составляет 2, при  $n=2$  – 4).

Т.к. значение каждой булевской функции от  $n$  переменных представляет собой двоичный набор длины  $2^n$ , то число  $N$  различных булевских функций от  $n$  переменных равно числу различных двоичных наборов длины  $2^n$ , то есть:  $2^{2^n}$ . При  $n=1$  это число равно 4, при  $n=2$  – 16,  $n=3$  – 256,  $n=4$  – 65536 и т.д.

#### ***1.4. Полные системы булевских функций***

В предыдущем пункте было отмечено, что можно выразить любую булевскую функцию от  $n$  переменных в виде суперпозиции (взаимной подстановки) булевских функций от одной и двух переменных. В свою очередь эти функции задаются формулами, содержащими логические операции: отрицание ( $\neg$ ), конъюнкцию ( $\&$ ), строгую дизъюнкцию ( $\oplus$ ), дизъюнкцию ( $\vee$ ), импликацию ( $\rightarrow$ ), эквиваленцию ( $\sim$ ).

Но как известно из логики высказываний, операции логики высказываний строгая дизъюнкция ( $\oplus$ ), импликация ( $\rightarrow$ ), эквиваленция ( $\sim$ ) могут быть выражены через дизъюнкцию ( $\vee$ ), конъюнкцию ( $\&$ ) и отрицание ( $\neg$ ).

В свою очередь, дизъюнкция ( $\vee$ ) может быть выражена через конъюнкцию ( $\&$ ) и отрицание ( $\neg$ ), а конъюнкция ( $\&$ ) – через дизъюнкцию ( $\vee$ ) и отрицание ( $\neg$ ).

Таким образом, конъюнкция и отрицание, а также дизъюнкция и отрицание, образуют полную систему логических связок, то есть через эти операции могут быть выражены все остальные. Более то-

го, можно определить логическую операцию, через которую выражаются все шесть операций: отрицание ( $\neg$ ), конъюнкция ( $\&$ ), дизъюнкция ( $\vee$ ), строгая дизъюнкция ( $\oplus$ ), импликация ( $\rightarrow$ ), эквиваленция ( $\sim$ ). Таковой, например, является операция, соответствующая сложному союзу "не А или не В" ("или" соединительное). Эта операция обозначается символом  $|$  (например,  $A | B$ ) и получила название штрих Шеффера. Штрих Шеффера определяется с помощью следующей таблицы:

| X | Y | X   Y |
|---|---|-------|
| 0 | 0 | 1     |
| 0 | 1 | 1     |
| 1 | 0 | 1     |
| 1 | 1 | 0     |

Как легко видеть, штрих Шеффера представляет собой отрицание конъюнкции:  $X | Y \equiv \neg(X \& Y) \equiv \neg X \vee \neg Y$ . Можно убедиться, что  $\neg X \equiv X | X$ . Отсюда  $X \& Y \equiv \neg(X | Y) \equiv (X | Y) | (X | Y)$ .

Таким образом, через штрих Шеффера могут быть выражены конъюнкция и отрицание, а значит и все остальные операции логики высказываний. То есть система логических связок, содержащая единственную операцию - штрих Шеффера, является полной.

Есть еще одна логическая операция, аналогичная штриху Шеффера и называемая стрелкой Пирса. Она обозначается символом  $\uparrow$  и представляет собой отрицание дизъюнкции (или конъюнцию отрицаний):  $X \uparrow Y \equiv \neg(X \vee Y) \equiv \neg X \& \neg Y$ .

Можно убедиться, что  $\neg X \equiv X \uparrow X$ . А отсюда:  $X \vee Y \equiv \neg(X \uparrow Y) \equiv (X \uparrow Y) \uparrow (X \uparrow Y)$ .

Таким образом, через стрелку Пирса могут быть выражены дизъюнкция и отрицание, а значит и все остальные операции логики высказываний. То есть система логических связок, содержащая единственную операцию - стрелку Пирса, является полной.

Систему булевских функций будем называть функционально полной, если любую булевскую функцию можно выразить в виде суперпозиции (взаимной подстановки) функций из этой системы.

В соответствии с высказанными выше соображениями о полноте системы логических связок и задании булевских функций фор-

мулами логики высказываний, можно сделать вывод о функциональной полноте следующих систем булевских функций:

$S_0 = \{\varphi_3(x), f_2(x,y), f_8(x,y)\}$  - отрицание, конъюнкция, дизъюнкция.

$S_1 = \{\varphi_3(x), f_2(x,y)\}$  - отрицание, конъюнкция.

$S_2 = \{\varphi_3(x), f_8(x,y)\}$  - отрицание, дизъюнкция.

$S_3 = \{f_{15}(x,y)\}$  - отрицание конъюнкции (штрих Шеффера).

$S_4 = \{f_9(x,y)\}$  - отрицание дизъюнкции (стрелка Пирса).

$S_5 = \{\varphi_3(x), f_{14}(x,y)\}$  - отрицание, импликация.

В последующем мы увидим, что с точки зрения проектирования вычислительных устройств особый интерес представляют  $S_3$  и  $S_4$ .

Общий критерий функциональной полноты системы булевских функций, выражающий необходимые и достаточные условия, носит название критерия Поста-Яблонского /См. книгу: Яблонский С.В., Гаврилов Г.П., Кудрявцев В.Б. Функции алгебры логики и классы Поста. – М.: Наука, 1966. – 120с./.

## 2. Булевы алгебры

Под алгеброй, как правило, понимается раздел математики, посвященный изучению свойств числовых (арифметических) операций (сложение, вычитание, умножение, деление, возведение в степень, извлечение корня и т.д.), выражений, тождеств, уравнений и т.п.

Но, вообще говоря, слово алгебра в математике понимается значительно шире. Алгеброй называют множество объектов любой природы с определенными и замкнутыми на этом множестве операциями.

Если какая-либо операция  $\lambda$  определена на множестве  $Q$  и результат ее также принадлежит этому множеству, то говорят, что операция  $\lambda$  замкнута на этом множестве или множество замкнуто относительно операции  $\lambda$ . Например, множество натуральных чисел  $N$  замкнуто относительно операций сложения и умножения натуральных чисел, но не замкнуто относительно операций вычитания и деления, которые могут в качестве результата давать значения, не принадлежащие множеству натуральных чисел (вычитание – отрицательные числа, а деление – дробные).

С этой точки зрения, арифметика – это алгебра с операциями

сложения и умножения, замкнутыми на множестве натуральных чисел. Если к числу операций добавить вычитание, то, строго говоря, это уже не будет алгебра.

Рассмотренная логика высказываний называется булевой алгеброй /Название *булевы* эти алгебры получили в честь известного английского математика Джорджа Буля (1815-1864)/. Это алгебра высказываний, объектами которой являются высказывания, а определенными и замкнутыми на этих объектах являются операции дизъюнкции, конъюнкции и отрицания.

Булева алгебра представляет собой множество объектов (любой, но одинаковой, природы) с двумя "особыми" объектами ("константами") – "единица" (1) и "нуль» (0) и двумя замкнутыми на множестве объектов операциями "сложения" (+) и "умножения" ( $\times$ ), обладающими следующими свойствами:

коммутативность + и  $\times$ :

$$A+B=B+A, A\times B=B\times A;$$

ассоциативность + и  $\times$ :

$$A+(B+C)=(A+B)+C, A\times(B\times C)=(A\times B)\times C;$$

дистрибутивность + относительно  $\times$  и  $\times$  относительно +:

$$A+(B\times C)=(A+B)\times(A+C), A\times(B+C)=(A\times B)+(A\times C);$$

идемпотентность + и  $\times$ :

$$A+A=A, A\times A=A.$$

Кроме того, «особые» объекты («константы») I и O обладают следующими свойствами:

$$A+O=A, A+I=I, A\times I=A, A\times O=O.$$

С точки зрения этого определения алгебра множеств является булевой алгеброй, в которой роль «сложения» играет операция объединения множеств ( $\rightarrow$ ), роль "умножения" – операция пересечения множеств ( $\cap$ ), роль константы "нуль" – пустое множество ( $\emptyset$ ), роль "единицы" – универсальное множество (U). Легко проверить, что все указанные выше свойства операций и констант булевой алгебры здесь выполняются.

Алгебра высказываний также является булевой алгеброй. В ней роль «сложения» играет операция дизъюнкции ( $\vee$ ), роль "умножения" – операция конъюнкции ( $\&$ ), роль "нуля" – логическая константа 0, роль "единицы" – логическая константа 1. И опять-таки легко проверить, что все указанные выше свойства операций и кон-

стант булевой алгебры выполняются.

Аналогично можно убедиться в том, что "алгебра переключательных схем" также является булевой: "сложению" будет соответствовать параллельное соединение контактов, "умножению" – последовательное соединение, константе "нуль" – всегда разомкнутый контакт, «единице» – всегда замкнутый контакт.

Приведем **пример** еще одной булевой алгебры – "алгебры максимумов и минимумов". Примем в качестве объектов (элементов) нашей алгебры, например, множество всех чисел отрезка  $[0,1]$ , т.е.  $0 \leq x \leq 1$ . В качестве операции "сложения" будем рассматривать операцию взятия максимального из двух чисел  $x$  и  $y$  и обозначать ее  $\max(x,y)$ , в качестве операции "умножения" - операцию взятия минимального из двух чисел  $x$  и  $y$  и обозначать ее  $\min(x,y)$ . В качестве константы "нуль" примем минимальное число из рассматриваемого отрезка, то есть число 0, а в качестве "единицы" – максимальное число из рассматриваемого отрезка – 1.

Легко проверить, что для определенных таким образом констант и операций выполняются все свойства булевой алгебры.

В самом деле:

коммутативность  $\min$  и  $\max$ :

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

ассоциативность  $\min$  и  $\max$ :

$$\min(x, \min(y,z)) = \min(\min(x, y), z),$$

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

дистрибутивность  $\min$  относительно  $\max$ :

$$\min(x, \max(y,z)) = \max(\min(x,y), \min(x,z)),$$

$$\max(x, \min(y,z)) = \min(\max(x,y), \max(x,z));$$

идемпотентность  $\min$  и  $\max$ :

$$\min(x,x)=x, \max(x,x)=x;$$

свойства констант 0 и 1:

$$\min(x,0)=0, \min(x,1)=x, \max(x,0)=x, \max(x,1)=1.$$



## Лекция 8. Применение аппарата булевой алгебры к анализу и синтезу комбинационных схем

### Учебные вопросы:

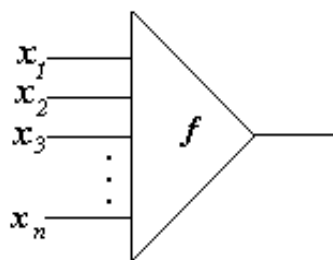
#### 1. Комбинационные схемы (схемы без памяти)

Рассмотрим так называемые схемы из функциональных элементов (комбинационные схемы), вычисляющие (реализующие) булевские функции.

Под функциональным элементом будем понимать некоторое устройство (внутренняя структура которого нас не интересует /Заинтересованному читателю можем порекомендовать, например, книгу: О.А. Маслюков. Вычислительная техника и программирование. - М.: Высшая школа, 1993. - 208с./), обладающее такими свойствами:

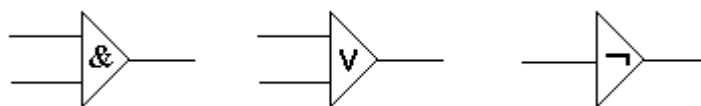
- оно имеет  $n \geq 1$  упорядоченных входов и один выход;
- на входы этого устройства могут подаваться сигналы, принимающие два значения, которые будем обозначать через 0 и 1;
- при каждом наборе сигналов на входах устройство выдает один из сигналов (0 или 1) в тот же момент, в который поступили сигналы на входе;
- набор сигналов на входах однозначно определяет сигнал на выходе, то есть, если в различные моменты времени на входы поступает одна и та же комбинация сигналов, то в эти моменты на выходе будет один и тот же сигнал.

С каждым функциональным элементом с  $n$  входами сопоставим булевскую функцию от  $n$  переменных  $f(x_1, x_2, \dots, x_n)$ , определяемую следующим образом: входу с номером  $i$  ( $i = 1, 2, \dots, n$ ) ставится в соответствие переменная  $x_i$  и с каждым набором  $\langle \alpha_1, \alpha_2, \dots, \alpha_i, \dots, \alpha_n \rangle$  этих переменных ( $\alpha_i \in \{0, 1\}$ ) сопоставляется число  $f(\alpha_1, \alpha_2, \dots, \alpha_i, \dots, \alpha_n)$ , равное 0 или 1 в зависимости от того, какой сигнал вырабатывается на выходе при подаче этого набора сигналов на входы данного функционального элемента. О функции  $f(x_1, x_2, \dots, x_n)$  будем говорить, что данный функциональный элемент ее реализует. Такой элемент будем изображать так, как представлено на рис.1.



**Рис.1. Общий вид функционального элемента**

В дальнейшем мы будем рассматривать функциональные элементы, реализующие полную систему логических операций: конъюнкцию, дизъюнкцию, отрицание. Эти элементы представлены на рис.2.

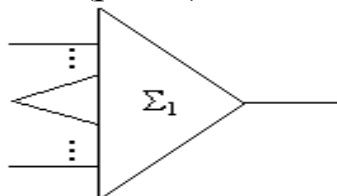


**Рис.2. Функциональные элементы, реализующие  $\&$ ,  $\vee$ ,  $\neg$**

Определим понятие "схема из функциональных элементов в базисе  $\{\&, \vee, \neg\}$ " и понятия ее выходов и входов. Определение будет носить индуктивный характер (подобно тому, как выше определялось понятие формулы логики высказываний).

Каждый функциональный элемент представляет собой схему из функциональных элементов с теми же входами и выходами, что и у этого элемента.

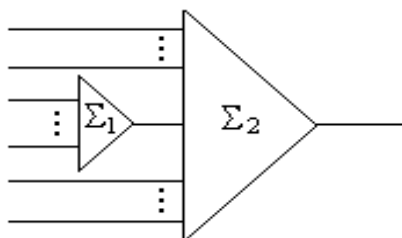
Если в схеме из функциональных элементов  $\Sigma_1$  два ее входа соединить вместе, то получится схема из функциональных элементов  $\Sigma$ , входами которой являются все несоединенные входы  $\Sigma_1$  и еще один вход, соответствующий двум соединенным входам схемы  $\Sigma_1$ , а выходом схемы  $\Sigma$  - выход  $\Sigma_1$  (рис.3).



**Рис.3. Соединение входов схемы**

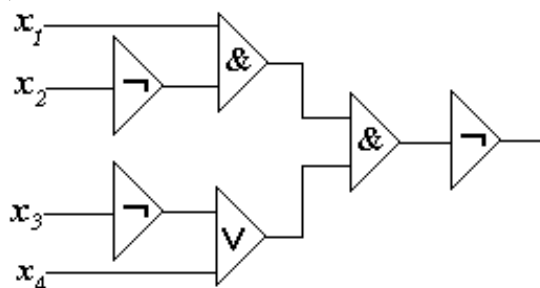
Если  $\Sigma_1$  и  $\Sigma_2$  - две схемы из функциональных элементов, то

конструкция  $\Sigma$ , получающаяся соединением какого-либо входа схемы  $\Sigma_2$  с выходом схемы  $\Sigma_1$ , также будет схемой из функциональных элементов. Входами схемы  $\Sigma$  будут все входы схемы  $\Sigma_1$  и все входы схемы  $\Sigma_2$ , за исключением того, который соединен с выходом схемы  $\Sigma_1$ , а выходом  $\Sigma$  является выход схемы  $\Sigma_2$ . (рис.4.).



**Рис.4. Подключение выхода схемы  $\Sigma_1$  к входу схемы  $\Sigma_2$**

Если в схеме из функциональных элементов  $\Sigma_1$  ее вход соединить с выходом некоторого функционального элемента из  $\Sigma_1$  без образования цикла для какого-либо функционального элемента (то есть его выход не должен соединяться, быть может, через другие функциональные элементы из  $\Sigma_1$  с его входом), то получившаяся конструкция  $\Sigma$  является схемой из функциональных элементов. Выходом  $\Sigma$  будет выход  $\Sigma_1$ , а входами  $\Sigma$  - все входы  $\Sigma_1$ , кроме того, который соединен с выходом функционального элемента. На рис. 5. приведен пример подобной схемы:



**Рис.5. Пример схемы, удовлетворяющей п.4 определения схемы**

Определение схемы из функциональных элементов завершено. Определим булевскую функцию, реализуемую данной схемой.

Если схема является функциональным элементом, то булевская функция, ею реализуемая, уже определена.

Если схема  $\Sigma_1$  реализует булевскую функцию  $f(x_1, x_2, \dots, x_n)$ , то схема  $\Sigma$ , построенная в п.2 определения схемы из функциональных элементов, реализует булевскую функцию, полученную из  $f(x_1, x_2, \dots, x_n)$  отождествлением переменных, отвечающих объединенным входам схемы  $\Sigma_1$ .

Пусть схема  $\Sigma_1$  реализует булевскую функцию  $f(x_1, x_2, \dots, x_n)$ , а схема  $\Sigma_2$  - булевскую функцию  $g(y_1, y_2, \dots, y_m)$ . Считаем, что все переменные  $x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_m$  попарно различны. Тогда схема  $\Sigma$ , построенная в п.3 определения схемы из функциональных элементов, реализует булевскую функцию  $g(y_1, y_2, \dots, y_{i-1}, f(x_1, x_2, \dots, x_n), y_{i+1}, \dots, y_m)$ , то есть функцию, получаемую путем подстановки в функцию  $g(y_1, y_2, \dots, y_m)$  вместо аргумента  $y_i$ , сопоставленного входу схемы  $\Sigma_2$ , соединенному с выходом  $\Sigma_1$ , функции  $f(x_1, x_2, \dots, x_n)$ .

Булевская функция, реализуемая схемой  $\Sigma$ , построенной в п.4 определения схемы из функциональных элементов, получается из булевой функции, реализуемой схемой  $\Sigma_1$ , операцией, типа описанной в п. 3 настоящего определения. Например, приведенная на рис.5. схема реализует функцию  $f(x_1, x_2, x_3, x_4) = \neg(x_1 \& \neg x_2 \& (\neg x_3 \vee x_4))$ .

Определение булевой функции, реализуемой схемой из функциональных элементов, завершено.

Поскольку, как отмечалось выше, любую булевскую функцию от  $n$  переменных можно выразить в виде суперпозиции функций, образующих полную систему булевских функций (в частности,  $S_0 = \{f_1(x), f_2(x, y), f_3(x, y)\}$  - отрицание, конъюнкция, дизъюнкция), то значит, что и любая булевская функция может быть реализована соответствующей схемой из функциональных элементов.

Схемы из функциональных элементов имеют разнообразное техническое применение. Во многих реальных автоматических устройствах есть блоки, представляющие собой соединение схем из функциональных элементов.

## Лекция 9. Примеры комбинационных схем

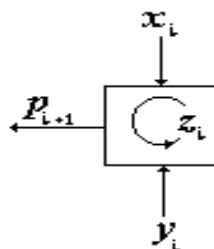
### Учебные вопросы:

1. Функциональные схемы двоичных сумматоров
2. Логические операции, выполняемые микропроцессором.

### 1. Функциональные схемы двоичных сумматоров

Схемы из функциональных элементов имеют разнообразное техническое применение. Во многих реальных автоматических устройствах есть блоки, представляющие собой соединение схем из функциональных элементов.

В качестве примера опишем в виде схемы из функциональных элементов один из основных узлов ЭВМ двоичный сумматор – устройство, предназначенное для сложения  $n$ -разрядных двоичных чисел. Пусть имеются два двоичных числа:  $X = x_n x_{n-1} x_{n-2} \dots x_2 x_1$  и  $Y = y_n y_{n-1} y_{n-2} \dots y_2 y_1$ , где  $x_i, y_j$  - двоичные цифры 0 или 1. Требуется получить число  $Z$ , равное сумме чисел  $X$  и  $Y$ . Рассмотрим вначале одноразрядный двоичный сумматор на два входа. Схематически он представлен на рис. 1.



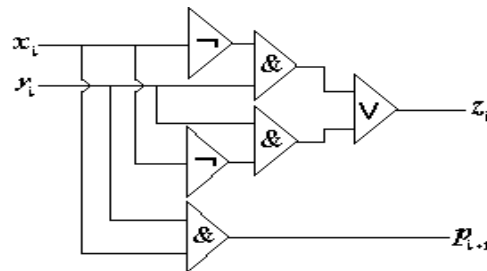
**Рис. 1. Схема одноразрядного двоичного сумматора на два входа**

Здесь:  $x_i$  -  $i$ -тая цифра числа  $X$ ;  $y_i$  -  $i$ -тая цифра числа  $Y$ ;  $z_i$  -  $i$ -тая цифра результата сложения - числа  $Z$ ;  $p_{i+1}$  - перенос в следующий  $(i+1)$ -ый разряд. Входы сумматора -  $x_i$  и  $y_i$ , выходы сумматора -  $z_i$  и  $p_{i+1}$ .

Опишем выходы сумматора как булевские функции его входов в виде следующей таблицы:

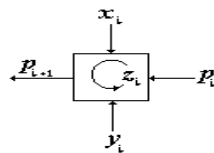
| $x_i$ | $y_i$ | $z_i$ | $p_{i+1}$ |
|-------|-------|-------|-----------|
| 0     | 0     | 0     | 0         |
| 0     | 1     | 1     | 0         |
| 1     | 0     | 1     | 0         |
| 1     | 1     | 0     | 1         |

Очевидно  $z_i = f_1(x_i, y_i) = \neg x_i \& y_i \vee x_i \& \neg y_i$ ;  $p_{i+1} = f_2(x_i, y_i) = x_i \& y_i$ .  
Соответствующие схемы из функциональных элементов представлены на рис.2.



**Рис. 2. Функциональная схема одноразрядного двоичного сумматора на два входа**

Рассмотрим одноразрядный двоичный сумматор на три входа (рис. 3).



**Рис.3. Схема одноразрядного двоичного сумматора на три входа**

Здесь:  $x_i$  -  $i$ - тая цифра числа  $X$ ;  $y_i$  -  $i$ -тая цифра числа  $Y$ ;  $p_i$  - перенос из предыдущего,  $i$ -го разряда;  $z_i$  -  $i$ -тая цифра результата сложения - числа  $Z$ ;  $p_{i+1}$  - перенос в следующий,  $(i+1)$ -ый разряд. Входы сумматора -  $x_i$ ,  $y_i$ ,  $p_i$ , выходы сумматора -  $z_i$  и  $p_{i+1}$ .

Опишем выходы сумматора как булевские функции его входов в виде следующей таблицы:

| $x_i$ | $y_i$ | $p_i$ | $z_i$ | $p_{i+1}$ |
|-------|-------|-------|-------|-----------|
| 0     | 0     | 0     | 0     | 0         |
| 0     | 0     | 1     | 1     | 0         |
| 0     | 1     | 0     | 1     | 0         |
| 0     | 1     | 1     | 0     | 1         |
| 1     | 0     | 0     | 1     | 0         |
| 1     | 0     | 1     | 0     | 1         |
| 1     | 1     | 0     | 0     | 1         |
| 1     | 1     | 1     | 1     | 1         |

Можно проверить, что:

$$z_i = \varphi_1(x_i, y_i, p_i) =$$

$$\neg x_i \& \neg y_i \& p_i \vee \neg x_i \& y_i \& \neg p_i \vee x_i \& \neg y_i \& \neg p_i \vee x_i \& y_i \& p_i;$$

$$p_{i+1} = \varphi_2(x_i, y_i, p_i) =$$

$$\neg x_i \& y_i \& p_i \vee x_i \& \neg y_i \& p_i \vee x_i \& y_i \& \neg p_i \vee x_i \& y_i \& p_i.$$

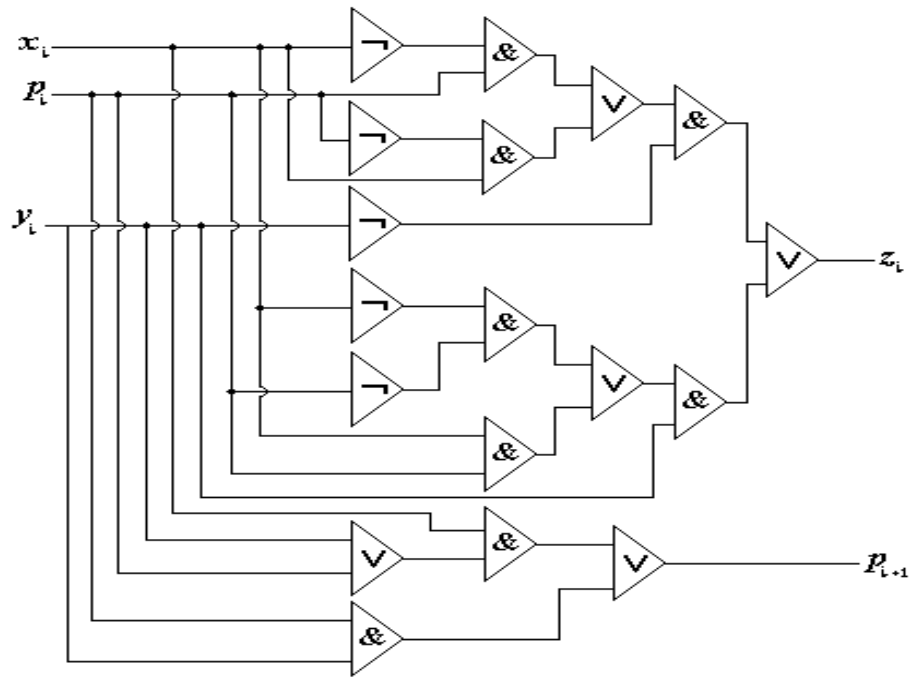
После равносильных преобразований имеем

$$z_i = \varphi_1(x_i, y_i, p_i) =$$

$$\neg y_i \& (\neg x_i \& p_i \vee x_i \& \neg p_i) \vee y_i \& (\neg x_i \& \neg p_i \vee x_i \& p_i);$$

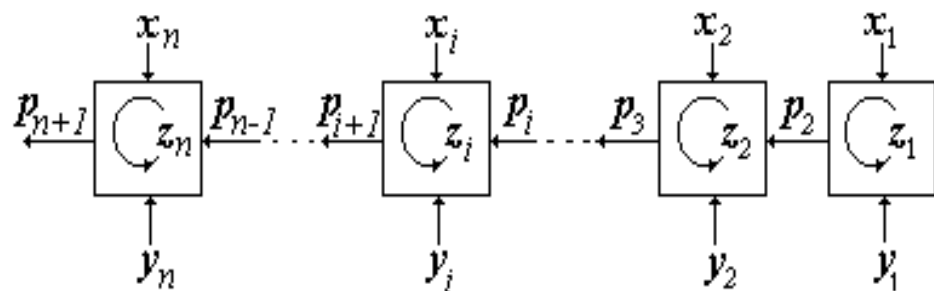
$$p_{i+1} = \varphi_2(x_i, y_i, p_i) = x_i \& (y_i \vee p_i) \vee y_i \& p_i.$$

Соответствующая схема представлена на рис. 4.



**Рис. 4. Функциональная схема одноразрядного двоичного сумматора на три входа**

Сложение  $n$ -разрядных двоичных чисел можно осуществить, соединив  $n$  сумматоров таким образом, как это показано на рис. 5.



## Рис. 5. Схема $n$ -разрядного двоичного сумматора

В данной схеме сумматора первый элемент имеет два входа, а все остальные элементы - по три. Появление 1 на выходе  $y_{n+1}$  означает переполнение разрядной сетки, то есть попытка представить число, содержащее больше, чем  $n$  двоичных разрядов.

В некоторых схемах сумматоров "замыкают" выход  $p_{n+1}$  последнего элемента на вход  $p_1$  первого. Тогда все элементы схемы сумматора будут иметь по три входа. Такое сложение называют циклическим. Оно находит свое применение при выполнении арифметических операций в ЭВМ.

Сложение на двоичном сумматоре осуществляется "параллельно" за один шаг (такт), и выход полностью определяется текущим состоянием входов, независимо от их количества. То есть, если в момент времени  $t$  подать  $2n$  сигналов на входы  $x_n, x_{n-1}, x_{n-2}, \dots, x_2, x_1$  и  $y_n, y_{n-1}, y_{n-2}, \dots, y_2, y_1$ , то  $n$  сигналов на выходах сумматора  $z_n, z_{n-1}, z_{n-2}, \dots, z_2, z_1$  появятся в тот же момент времени. Поэтому схемы из функциональных элементов называют комбинационными схемами или схемами без памяти. Они не запоминают результатов своей предыдущей работы.

Естественно, вычислительные устройства должны обладать "памятью". Ее наличие даст возможность конструировать счетчики, арифметические регистры и различные схемы, которые, выполнив одну функцию, начинают выполнять другую.

## 2. Логические операции, выполняемые микропроцессором

Микропроцессор компьютера способен выполнять основные логические операции: конъюнкцию  $\&$  (обозначение AND), дизъюнкцию  $\vee$  (обозначение OR), отрицание  $\neg$  (обозначение NOT), строгую дизъюнкцию (или сложение по модулю 2)  $\oplus$  (обозначение XOR).

Особенностью выполнения логических операций микропроцессором является то, что они выполняются над двоичными кодами (словами, полусловами, двойными словами) поразрядно.

Например, конъюнкций двух двоичных кодов 01101100 и 10101101 будет код 00101100:

01101100



$$\begin{array}{r} \& \\ 10101101 \\ \hline 00101100. \end{array}$$

Дизъюнкция этих же кодов дает код 11101101:

$$\begin{array}{r} 01101100 \\ \vee \\ 10101101 \\ \hline 11101101. \end{array}$$

Строгая дизъюнкция этих кодов дает код 11000001:

$$\begin{array}{r} 01101100 \\ \oplus \\ 10101101 \\ \hline 11000001. \end{array}$$

Отрицанием кода 10101101 является код 01010010.

Аналогично выполняются операции над более длинными кодами.

Логическая операция конъюнкция может быть использована для проверки значения интересующего нас бита данного байта. Например, если нас интересует значение 3-го бита байта XXXXXXXX, то достаточно взять конъюнкцию этого байта с двоичным кодом 00100000. Результатом конъюнкции, как легкообразить, будет код 00X00000. То есть, если полученный результат будет численно равен 0, то 3-ий бит исходного байта равен 0, если же полученный результат будет численно отличен от 0, то 3-ий бит этого байта равен 1.

Аналогично с помощью конъюнкции данного байта с кодом 00001000 можно выделить 5-й бит. и т.д. Такой способ обеспечивает пользователю доступ не только к отдельному байту памяти (по его адресу), но и доступ к отдельному биту этого байта.

## Лекция 10. Исчисление высказываний

### Учебные вопросы:

1. Понятие формулы исчисления высказываний.
2. Аксиомы исчисления высказываний.
3. Правила вывода. Доказуемые формулы.

При построении алгебры высказываний, использовались логические значения высказываний истина и ложь. Но понятия истинности и ложности не математические. Эти понятия во многих случаях субъективны и скорее относятся к философии.

В связи с этим желательно построить математическую логику, не пользуясь понятиями истинности и ложности. Необходимо также при этом построении не применять самих законов логики.

*Исчисление высказываний - это аксиоматическая логическая система, интерпретацией которой является алгебра высказываний.*

### 1. Понятие формулы исчисления высказываний

Описание всякого исчисления включает в себя описание символов этого исчисления (алфавита), формул, являющихся конечными конфигурациями символов, и определение выводимых формул.

**Алфавит исчисления высказываний состоит из символов трех категорий:**

*1. Символы первой категории:  $x, y, z, \dots$ . Эти символы будем называть переменными высказываниями.*

*2. Символы второй категории:  $\vee, \&, \rightarrow, \neg$ . Они носят общее название логических связок. Первый из них - знак дизъюнкции или логического сложения, второй - знак конъюнкции или логического умножения, третий - знак импликации или логического следования и четвертый - знак отрицания.*

*3. Третью категорию составляет пара символов ( ), называемая скобками.*

Других символов исчисление высказывания не имеет. Формулы исчисления высказываний представляют собой последовательности символов алфавита исчисления высказываний. Для обозначения формул будем пользоваться большими буквами латинского алфа-

вита. Эти буквы не являются символами исчисления. Они представляют собой только условные обозначения формул.

### **Определение формулы исчисления высказываний.**

- 1. Всякая переменная  $x, y, z, \dots$  является формулой.**
- 2. Если  $A$  и  $B$  - формулы, то слова  $(A \& B)$ ,  $(A \vee B)$ ,  $(A \rightarrow B)$ ,  $\neg A$  - также формулы.**
- 3. Никакая другая строчка символов не является формулой.**

Переменные высказывания будем называть элементарными формулами. Приведем примеры формул исчисления высказываний.

Переменные высказывания  $x, y, z$  являются формулами согласно п. 1 определения формулы. Но тогда слова  $(x \& y)$ ,  $(x \vee z)$ ,  $(y \rightarrow z)$ ,  $\neg x$  являются формулами согласно п. 2 определения. По этой же причине будут формулами:  $\neg (x \& y)$ ,  $((x \vee z) \& (y \rightarrow z))$ ,  $((x \& y) \rightarrow (y \rightarrow z))$ .

Очевидно, не являются формулами слова:  $x \neg y$ ,  $\& x$ ,  $(x \& y$ ,  $x y u$  (в третьем из этих слов содержится не закрытая скобка, а в четвертом - нет скобок).

### **Понятие подформулы или части формулы.**

**1. Подформулой элементарной формулы является только она сама.**

**2. Если формула имеет вид  $\neg A$ , то ее подформулами являются: она сама, формула  $A$  и все подформулы формулы  $A$ .**

**3. Если формула имеет вид  $(A * B)$  (здесь и в дальнейшем под символом  $*$  будем понимать любой из трех символов  $\vee, \&, \rightarrow$ ), то ее подформулами являются: она сама, формулы  $A$  и  $B$  и все подформулы формул  $A$  и  $B$ .**

Например, для формулы  $((x \& \neg y) \rightarrow \neg (\neg x \vee y))$  ее подформулами будут:

$((x \& \neg y) \rightarrow \neg (\neg x \vee y))$  - подформула нулевой глубины,  $(x \& \neg y)$ ,  $\neg (\neg x \vee y)$  - подформулы первой глубины,  $x, \neg y$ ,  $(\neg x \vee y)$  - подформулы второй глубины,  $y, \neg z$  - подформулы третьей глубины,  $z$  - подформула четвертой глубины.

Очевидно, что на самой большой глубине находятся лишь элементарные формулы. Однако элементарные формулы могут быть и на других глубинах.

Введем в запись формул некоторые упрощения. Будем опускать в записи формул скобки по тем же правилам, что и в алгебре высказываний.

## 2. Аксиомы исчисления высказываний

Следующим этапом в построении исчисления высказываний является выделение класса доказуемых формул. Сначала определяются исходные доказуемые формулы (аксиомы), а затем определяются правила вывода, которые позволяют из имеющихся доказуемых формул получить новые доказуемые формулы.

Образование доказуемой формулы из исходных доказуемых формул путем применения правил вывода, называется выводом данной формулы из аксиом.

### Система аксиом исчисления высказываний.

Система аксиом исчисления высказываний состоит из 11 аксиом, которые делятся на четыре группы.

#### *Первая группа аксиом:*

1.  $x \rightarrow (y \rightarrow x)$ .
2.  $(x \rightarrow (y \rightarrow z)) \rightarrow ((x \rightarrow y) \rightarrow (x \rightarrow z))$ .

#### *Третья группа аксиом:*

1.  $x \rightarrow x \vee y$
2.  $y \rightarrow x \vee y$
3.  $(x \rightarrow z) \rightarrow ((y \rightarrow z) \rightarrow (x \vee y \rightarrow z))$

#### *Вторая группа аксиом:*

1.  $x \& y \rightarrow x$
2.  $x \& y \rightarrow y$
3.  $(z \rightarrow x) \rightarrow ((z \rightarrow y) \rightarrow (z \rightarrow x \& y))$

#### *Четвертая группа аксиом:*

1.  $(x \rightarrow y) \rightarrow (\neg y \rightarrow \neg x)$
2.  $x \rightarrow \overline{\overline{x}}$
3.  $\overline{\overline{x}} \rightarrow x$

## 3. Правила вывода. Доказуемые формулы

### 1. Правило подстановки.

Если формула  $A$  доказуема в исчислении высказываний (ИВ),  $x$  переменная,  $B$  – произвольная формула ИВ, то формула, полученная в результате замены в формуле  $A$  переменной  $x$  всюду, где она входит, формулой  $B$ , является также доказуемой формулой.

Операция замены в формуле  $A$  переменной  $x$  формулой  $B$  носит название подстановки и символически записывается так:  $\int_x^B (A)$

Уточним сформулированное правило.

- а) Если формула  $A$  есть переменная  $x$ , то подстановка  $\int_x^B(A)$  дает  $B$ .
- б) Если формула  $A$  есть переменная  $y$ , отличная от  $x$ , то подстановка  $\int_x^B(A)$  дает  $A$ .
- в) Если  $A$  - формула, для которой подстановка уже определена, то подстановка  $B$  вместо  $x$  в отрицание  $\overline{A}$  есть отрицание подстановки, то есть подстановка  $\int_x^B(\overline{A})$  дает  $\overline{\int_x^B(A)}$ .
- г) Если  $A_1$  и  $A_2$  - формулы, для которых подстановки  $v$  уже определены, то подстановка  $\int_x^B(A_1 * A_2)$  дает  $\int_x^B(A_1) * \int_x^B(A_2)$ .

Если  $A$  - доказуемая формула, то будем писать  $\vdash A$ . Тогда правило подстановки можно записать схематически следующим образом:

$$\frac{\vdash A}{\vdash \int_x^B(A)}$$

Эта запись читается так: «Если формула  $A$  доказуема, то доказуема формула  $\int_x^B(A)$ ».

## 2. Правило заключения.

Если формулы  $A$  и  $A \rightarrow B$  доказуемы в исчислении высказываний, то формула  $B$  также доказуема. Схематическая запись этого правила имеет вид:  $\frac{\vdash A; \vdash A \rightarrow B}{\vdash B}$

### Определение доказуемой формулы.

- а) *Всякая аксиома является доказуемой формулой.*
- б) *Формула, полученная из доказуемой формулы путем применения подстановки вместо переменной  $x$  произвольной формулы  $B$  есть доказуемая формула.*
- в) *Формула  $B$ , полученная из доказуемых формул  $A$  и  $A \rightarrow B$  путем применения правила заключения, есть доказуемая формула.*
- г) *Никакая другая формула исчисления высказываний не счита-*

ется доказуемой.

Процесс получения доказуемых формул будем называть доказательством.

### Примеры доказательств.

#### 1. Доказать, что $\vdash A \rightarrow A$ (рефлексивность импликации).

Воспользуемся аксиомой :  $B = (x \rightarrow (y \rightarrow z)) \rightarrow ((x \rightarrow y) \rightarrow x \rightarrow z)$

и выполним подстановку  $\int_z^x(B)$ , тогда получим

$$(x \rightarrow (y \rightarrow z)) \rightarrow ((x \rightarrow y) \rightarrow x \rightarrow x) \quad (1)$$

Применяя правило заключения к аксиоме  $B$  и формуле (1), получим

$$\vdash (x \rightarrow y) \rightarrow (x \rightarrow x) \quad (2)$$

В формуле (2) осуществим подстановку  $\int_y^{\bar{x}}$ . В результате получим доказуемую формулу

$$\vdash (x \rightarrow \bar{x}) \rightarrow (x \rightarrow x) \quad (3)$$

Применим правило заключения к аксиоме  $(x \rightarrow \bar{x})$  и формуле (3). Это приводит к доказуемой формуле

$$\vdash x \rightarrow x \quad (4)$$

Подставляя в (4) вместо  $x$  формулу  $A$ , получим  $\vdash A \rightarrow A$ .

#### 2. Доказать, что $\vdash \overline{x \vee y} \rightarrow \bar{x} \& \bar{y}$

Возьмем аксиому  $(z \rightarrow x) \rightarrow ((z \rightarrow y) \rightarrow (z \rightarrow x \& y))$  и выполним в ней последовательно две подстановки, заменяя сначала  $x$  на  $\bar{x}$ , а затем  $y$  на  $\bar{y}$ . В результате получим доказуемую формулу

$$(z \rightarrow \bar{x}) \rightarrow ((z \rightarrow \bar{y}) \rightarrow (z \rightarrow \bar{x} \& \bar{y})) \quad (5)$$

В формуле (5) выполним подстановку  $\int_{\overline{x \vee y}}^z$ , получим

$$(\overline{x \vee y} \rightarrow \bar{x}) \rightarrow ((\overline{x \vee y} \rightarrow \bar{y}) \rightarrow (\overline{x \vee y} \rightarrow \bar{x} \& \bar{y})) \quad (6)$$

Докажем, что формулы  $\overline{x \vee y} \rightarrow \bar{x}$  и  $\overline{x \vee y} \rightarrow \bar{y}$  доказуемы.

Возьмем аксиому  $B = (x \rightarrow y) \rightarrow (\neg y \rightarrow \neg x)$  и выполним подста-

новку  $\int_{x \vee y}^y (B)$ ,  
получим

$$/-(x \rightarrow x \vee y) \rightarrow (\overline{x \vee y} \rightarrow \bar{x}) \quad (7)$$

Применяя к формуле (7) и аксиоме  $x \rightarrow x \vee y$  правило заключения, получаем доказуемость формулы  $\overline{x \vee y} \rightarrow \bar{x}$ . Аналогично устанавливается доказуемость формулы  $\overline{x \vee y} \rightarrow \bar{y}$ .

Применим правило заключения к формулам  $\overline{x \vee y} \rightarrow \bar{x}$  и (6), получим доказуемую формулу

$$(\overline{x \vee y} \rightarrow \bar{y}) \rightarrow (\overline{x \vee y} \rightarrow \bar{x} \& \bar{y}) \quad (8)$$

Применяя правило заключения к формуле (8), получаем доказуемость исходной формулы  $\overline{x \vee y} \rightarrow \bar{x} \& \bar{y}$ .

## Лекция 11. Производные правила вывода. Выводимость формул

### Учебные вопросы:

1. Производные правила вывода.
2. Понятие выводимости формулы из совокупности формул.

### 1. Производные правила вывода

Производные правила вывода, как и рассмотренные правила подстановки и заключения, позволяют получать новые доказуемые формулы. Они получаются с помощью правил подстановки и заключения, а поэтому являются производными от них.

#### Правило одновременной подстановки.

*Пусть  $A$  - доказуемая формула,  $x_1, x_2, \dots, x_n$  - переменные, а  $B_1, B_2, \dots, B_n$  - любые формулы исчисления высказываний. Тогда результат одновременной подстановки в  $A$  вместо  $x_1, x_2, \dots, x_n$  соответственно формул  $B_1, B_2, \dots, B_n$  является доказуемой формулой.*

Для доказательства этого правила выберем переменные  $z_1, z_2, \dots, z_n$  попарно различные и отличные от всех переменных, входящих в формулы  $A, B_1, B_2, \dots, B_n$ . Теперь последовательно сделаем в формуле  $A$   $n$  подстановок, заменяя сначала  $x_1$  на  $z_1$ , затем  $x_2$  на  $z_2$  и так далее и, наконец,  $x_n$  на  $z_n$ . Эти подстановки дают доказуемые формулы:

$$\begin{aligned} & \frac{z_1}{x_1} \int (A) \text{ дает } \vdash A_1, \frac{z_2}{x_2} \int (A_1) \text{ дает } \vdash A_2, \dots, \frac{z_{n-1}}{x_{n-1}} \int (A_{n-1}) \text{ дает} \\ & \vdash A_n. \end{aligned}$$

Далее проведем последовательно еще  $n$  подстановок в формулу  $A_n$ , заменяя сначала  $z_1$  на  $B_1$ , затем  $z_2$  на  $B_2$  и так далее и, наконец,  $z_n$  на  $B_n$ . Эти подстановки дают доказуемые формулы

$$\begin{aligned} & \frac{B_1}{z_1} \int (A_n) \text{ дает } \vdash C_1, \frac{B_2}{z_2} \int (C_1) \text{ дает } \vdash C_2, \dots, \frac{B_{n-1}}{z_{n-1}} \int (C_{n-1}) \text{ дает} \\ & \vdash C_n. \end{aligned}$$

Ясно, что доказуемая формула  $C_n$  получается в результате одновременной подстановки в формулу  $A$  вместо переменных  $x_1, x_2, \dots, x_n$  формул  $B_1, B_2, \dots, B_n$ .

Схематично операция одновременной подстановки записывается в виде:



$$\frac{\frac{|-A}{B_1, B_2, \dots, B_n}}{|-\int(A)}_{x_1, x_2, \dots, x_n}$$

### Правило сложного заключения.

Второе производное правило применяется к формулам вида  $(A_1 \rightarrow (A_2 \rightarrow (A_3 \rightarrow (\dots (A_n \rightarrow L) \dots))))$  и формулируется так.

**Если формулы  $A_1, A_2, \dots, A_n$  и  $(A_1 \rightarrow (A_2 \rightarrow (A_3 \rightarrow (\dots (A_n \rightarrow L) \dots))))$  доказуемы, то формула  $L$  доказуема.**

Это утверждение легко доказывается последовательным применением правила заключения. Действительно, если формулы  $A_1$  и  $A_1 \rightarrow (A_2 \rightarrow (A_3 \rightarrow (\dots (A_n \rightarrow L) \dots)))$  доказуемы, то по правилу заключения доказуема формула  $A_2 \rightarrow (A_3 \rightarrow (\dots (A_n \rightarrow L) \dots))$ . Но так как формулы  $A_2$  и  $(A_3 \rightarrow (\dots (A_n \rightarrow L) \dots))$  доказуемы, то доказуема формула  $A_3 \rightarrow (\dots (A_n \rightarrow L) \dots)$ . Продолжая эти рассуждения, мы докажем наконец, что формула  $L$  доказуема.

Правило сложного заключения схематично записывается так:

$$\frac{|-A_1, |-A_2, \dots, |-A_n, A_1 \rightarrow (A_2 \rightarrow (A_3 \rightarrow (\dots (A_n \rightarrow L) \dots)))}{|-L}$$

### Правило силлогизма.

**Если доказуемы формулы  $A \rightarrow B$  и  $B \rightarrow C$ , то доказуема формула  $A \rightarrow C$ , то есть  $\frac{|-A \rightarrow B, |-B \rightarrow C}{|-A \rightarrow C}$ .**

Для доказательства этого правила в аксиомах

$D = (x \rightarrow (y \rightarrow z)) \rightarrow ((x \rightarrow y) \rightarrow x \rightarrow z)$  и  $B = x \rightarrow (y \rightarrow x)$  сделаем

следующие одновременные подстановки:  $\int_{x,y,z}^{A,B,C}(D), \int_{x,y}^{B \rightarrow C, A}(B)$ , получим

доказуемые формулы

$$\int (A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C)) \quad (9)$$

$$\int (B \rightarrow C) \rightarrow (A \rightarrow (B \rightarrow C)). \quad (10)$$

Кроме того, по условию доказуемы формулы

$$\int -A \rightarrow B \quad (11)$$

$$\int -B \rightarrow C. \quad (12)$$

Из формул (12) и (10) по правилу заключения получаем

$$\vdash A \rightarrow (B \rightarrow C) \quad (13)$$

Но тогда из формул (13), (11) и (9) по правилу сложного заключения имеем

$$\vdash \neg A \rightarrow C.$$

### Правило контрапозиции.

Если доказуема формула  $A \rightarrow B$ , то доказуема формула  $\neg B \rightarrow \neg A$ , то есть

$$\frac{\vdash A \rightarrow B}{\vdash \neg A \rightarrow \neg B}.$$

Для доказательства этого правила сделаем в  $D = (x \rightarrow y) \rightarrow (\neg y \rightarrow \neg x)$  одновременную подстановку  $\int_{x,y,z}^{A,B,C}(D)$  получим доказуемую формулу

$$\vdash (A \rightarrow B) \rightarrow (\neg B \rightarrow \neg A). \quad (14)$$

Но по условию доказуема формула  $\vdash (A \rightarrow B)$ . С учетом (14) по правилу заключения имеем  $\vdash \neg B \rightarrow \neg A$ .

### Правило снятия двойного отрицания.

а) Если доказуема  $A \rightarrow \neg\neg B$ , то доказуема формула  $A \rightarrow B$ .

б) Если доказуема формула  $\neg\neg A \rightarrow B$ , то доказуема формула  $A \rightarrow B$ , то есть

$$\frac{\vdash A \rightarrow \overline{\overline{B}}}{\vdash A \rightarrow B} \quad \text{и} \quad \frac{\vdash \overline{\overline{A}} \rightarrow B}{\vdash A \rightarrow B}.$$

Для доказательства этого правила в аксиомах  $B = x \rightarrow \overline{\overline{x}}$ ,  $D = \overline{\overline{x}} \rightarrow x$  выполним подстановки  $\int_x^A(B)$ ,  $\int_x^B(D)$ , получим доказуемые формулы

$$\vdash \neg A \rightarrow \neg\neg A, \quad (15)$$

$$\vdash \neg\neg\neg B \rightarrow A \quad (16)$$

Но по условию а) доказуема формула

$$\vdash \neg A \rightarrow \neg\neg\neg B \quad (17)$$

а по условию б) доказуема формула

$$\vdash \neg\neg\neg A \rightarrow B. \quad (18)$$

Таким образом, если выполнено условие а), то из формул (17) и (16) по правилу силлогизма получаем.

Если же выполнено условие б), то из формул (15) и (18) получаем  $\neg A \rightarrow B$ .

## 2. Понятие выводимости формулы из совокупности формул

Будем рассматривать конечную совокупность формул  $H = \{A_1, A_2, \dots, A_n\}$ .

**Определение формулы, выводимой из совокупности  $H$ .**

1) *Всякая формула  $A_i \in H$  является формулой, выводимой из  $H$ .*

2) *Всякая доказуемая формула выводима из  $H$ .*

3) *Если формулы  $C$  и  $C \rightarrow B$  выводимы из совокупности  $H$ , то формула  $B$  также выводима из  $H$ .*

Если некоторая формула  $B$  выводима из совокупности  $H$ , то записывают

$$H \vdash B.$$

Нетрудно видеть, что класс формул, выводимых из совокупности  $H$ , совпадает с классом доказуемых формул в случае, когда совокупность  $H$  содержит только доказуемые формулы, и в случае, когда  $H$  пуста.

Если же совокупность формул  $H$  содержит хотя бы одну не доказуемую формулу, то класс формул, выводимых из  $H$ , шире класса доказуемых формул.

**Пример.** Доказать, что из совокупности формул  $H = \{A, B\}$  выводима формула  $A \& B$ .

**Доказательство.** Так как  $A \in H$  и  $B \in H$ , то по определению выводимой формулы

$$H \vdash \neg A, \quad (19)$$

$$H \vdash \neg B. \quad (20)$$

Возьмем аксиомы  $B = (z \rightarrow x) \rightarrow ((z \rightarrow y) \rightarrow (z \rightarrow x \& y))$ ,  
 $D = x \rightarrow (y \rightarrow x)$  и выполним подстановки  $\int_{x,y,z}^{A,B,A}(B)$ ,  $\int_{x,y}^{B,A}(D)$ . В резуль-

тате получим доказуемые формулы, которые выводимы из  $H$  по определению выводимой формулы, т.е.

$$H \vdash \neg(A \rightarrow A) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow A \& B)) \quad (21)$$

$$H \mid B \rightarrow (A \rightarrow B) \quad . \quad (22)$$

Так как формула  $A \rightarrow A$  доказуема, то

$$A \rightarrow A. \quad (23)$$

Из формул (23) и (21) по правилу заключения получаем:

$$H \mid -(A \rightarrow B) \rightarrow (A \rightarrow A \& B). \quad (24)$$

Из формул (20) и (22) по правилу заключения получаем:

$$H \mid -(A \rightarrow B) \quad (25)$$

Из формул (25) и (24) по правилу заключения получаем:

$$H \mid -A \rightarrow A \& B \quad (26)$$

И, наконец, из формул (19) и (26) получаем

$$H \mid -A \& B. \quad (27)$$

Ясно, что при доказательстве выводимости формулы из совокупности формул можно пользоваться не только основным правилом заключения, но и правилом сложного заключения. Тогда, пользуясь этим правилом, предложение (27) можно получить из предложений (23), (25), (19) и (21).

### Понятие вывода

**Определение.** Выводом из конечной совокупности формул  $H$  называется всякая конечная последовательность формул, всякий член которой удовлетворяет одному из следующих трех условий:

- 1) он является одной из формул совокупности  $H$ ,
- 2) он является доказуемой формулой,
- 3) он получается по правилу заключения из двух любых предшествующих членов последовательности  $B_1, B_2, \dots, B_k$ .

Как было показано в предыдущем примере, выводом из совокупности формул  $H = \{A, B\}$  является конечная последовательность формул:

$$A, B, (A \rightarrow A) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow A \& B)), B \rightarrow (A \rightarrow B) \\ A \rightarrow A, (A \rightarrow B) \rightarrow (A \rightarrow A \& B), A \rightarrow B, A \rightarrow A \& B, A \& B.$$

Если же здесь воспользоваться правилом сложного заключения, то вывод можно записать так:

$$A, B, (A \rightarrow A) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow A \& B)), B \rightarrow (A \rightarrow B), A \rightarrow A, A \rightarrow B, A \& B.$$

Из определений выводимой формулы и вывода из совокупности формул следуют *очевидные свойства вывода*:

1) *Всякий начальный отрезок вывода из совокупности  $H$  есть вывод из  $H$ .*

*В самом деле, все формулы начального отрезка вывода удовлетворяют определению вывода.*

2) *Если между двумя соседними членами вывода из  $H$  (или в начале, или в конце его) вставить некоторый вывод из  $H$ , то полученная новая последовательность формул будет выводом из  $H$ .*

Действительно, например, если совокупности формул  $B_1, B_2, \dots, B_i, B_{i+1}, \dots, B_k$  и  $C_1, C_2, \dots, C_m$  являются выводами из  $H$ , то совокупность  $B_1, B_2, \dots, B_i, \dots, B_i, C_1, C_2, \dots, C_m, B_{i+1}, \dots, B_k$  по определению вывода, является выводом из  $H$ .

3) *Всякий член вывода из совокупности  $H$  является формулой, выводимой из  $H$ . Всякий вывод из  $H$  является выводом его последней формулы.*

4) *Если  $H \subset W$ , то всякий вывод из  $H$  является выводом из  $W$ .*

5) *Для того, чтобы формула  $B$  была выводима из совокупности, необходимо и достаточно, чтобы существовал вывод этой формулы из  $H$ .*

## Лекция 12. Основные понятия теории алгоритмов

### Учебные вопросы:

1. Неформальное понятие алгоритма. Свойства алгоритма.
2. Способы описания алгоритмов.

Понятие алгоритма, определяемое требованиями дискретности и элементарности шагов, детерминированности, результативности и массовости, не является строгим: в формулировках встречаются слова "последовательность действий", "система величин", "простой" и т.п., точный смысл которых не установлен. В дальнейшем такое понятие алгоритма мы будем называть непосредственным или *интуитивным понятием алгоритма*, то есть соответствующим нашей интуиции и опыту.

Интуитивное понятие алгоритма нестрогое, но практически не возникало случаев, когда математики (прежде всего) разошлись бы во мнениях относительно того, является ли алгоритмом тот или иной конкретно заданный процесс. Однако ситуация оказывается принципиально иной, когда приходится иметь дело с задачами, решение которых не известно и относительно которых имеется предположение, что они по своей сути не могут быть решены алгоритмическими методами, то есть *алгоритмически неразрешимы*. Доказать алгоритмическую неразрешимость, то есть *отсутствие* алгоритма решения рассматриваемой проблемы, на основе интуитивного представления об алгоритме невозможно. Доказать существование алгоритма можно путем фактического описания процесса, решающего проблему. Доказать несуществование алгоритма таким путем невозможно. Для этого надо точно знать, отсутствие чего мы должны доказать.

В середине 30-х годов двадцатого века были предприняты попытки формализовать и тем самым уточнить это понятие. В результате было предложено множество различных формализаций понятия алгоритма. Решением этих проблем занимались Д. Гильберт, К. Гедель, А. Чёрч, С. Клини, Э. Пост, А. Тьюринг, А.А. Марков. Впоследствии было установлено, что различные формализмы эквивалентны в том смысле, что классы точно описываемых ими процессов (задач) совпадают. Это дает основания для *предположения о том, что класс задач, решаемых в любой из этих фор-*

*мальных систем, и есть класс всех задач, которые могут быть решены "интуитивно алгоритмическими методами".* Явно эта гипотеза (правда, в несколько ином виде) была высказана А. Чёрчем и носит название **тезиса Чёрча**. Сейчас она является общепризнанной.

Формальное описание алгоритма создало предпосылки для разработки *теории алгоритмов*. Исследования в этой области стимулировали развитие вычислительной техники. С другой стороны, прогресс вычислительной техники благотворно повлиял на развитие теории алгоритмов.

## **1. Неформальное понятие алгоритма. Свойства алгоритмов**

Понятие вычислительной машины тесно связано с понятием вычислительной процедуры, или **алгоритма**.

**Алгоритмом** называют *точное и понятное исполнителю описание последовательности действий, позволяющих от исходных данных перейти к искомому результату.*

Исполнителем алгоритма может быть человек, механическое, электрическое, электронное или иное устройство. Исходные данные представляют собой, как правило, конечную систему величин, которая "перерабатывается" в систему выходных, искомых величин. Простейшими алгоритмами являются правила, по которым выполняется то или иное из четырех арифметических действий в десятичной системе счисления.

В математике серия (класс) задач определенного типа считается решенной, если для ее решения найден алгоритм. Нахождение алгоритмов является естественной целью математики. Однако не только в математике, но и в других областях человеческой деятельности встречаются процессы, протекающие по строго определенному формальному предписанию, то есть алгоритму.

Алгоритм обладает рядом свойств, среди которых наиболее важными являются следующие.

**Дискретность алгоритма.** Алгоритм рассматривается как процесс преобразования исходной системы величин, протекающий в дискретном времени<sup>1)</sup> так, что в каждый следующий момент вре-

---

<sup>1)</sup> Дискретность времени понимается как последовательность временных интервалов:  $(t_0, t_1), (t_1, t_2), (t_2, t_3), \dots, (t_{n-1}, t_n)$ , в течение каждого из которых совершается то или иное действие. Удобнее эти временные интервалы рассматривать как

мени система величин получается по определенному закону (правилу) из системы величин, имевшихся в предыдущий момент.

**Элементарность шагов алгоритма.** Закон (правило) получения последующей системы величин из предыдущей должен быть простым и понятным исполнителю. Иными словами, решение задачи распадается на ряд шагов, каждый из которых должен быть достаточно простым.

**Детерминированность(определенность) алгоритма.** Система величин, получаемых в любой не начальный момент времени, однозначно определяется системой величин, полученных в предшествующие моменты времени. То есть после выполнения очередного шага однозначно определено, что делать на следующем шаге.

**Результативность (направленность) алгоритма.** Если способ получения последующей величины из какой-нибудь заданной величины не дает результата, то должно быть указано, что считать результатом алгоритма. Иными словами, алгоритм всегда должен давать результат, то есть он всегда должен *заканчиваться*, выдавая результат.

**Массовость алгоритма.** Начальная система величин может выбираться из некоторого потенциально бесконечного множества. Иными словами, алгоритм должен быть пригоден для решения всех задач из заданного класса, а не только для решения одной конкретной задачи.

## 2. Способы описания алгоритмов

Применяются несколько способов описания алгоритма (то есть процесса) преобразования исходных данных в искомый результат.

1. **Словесный.**
2. **В виде графических схем (блок-схем).**
3. **В виде текстов на специальных алгоритмических языках.**

Приведем примеры описания алгоритма решения задачи нахождения остатка от деления натурального числа  $n$  на натуральное

---

отдельные точки на непрерывной оси времени, отождествляя интервал с его началом:  $t_0, t_1, t_2, t_3, \dots, t_n$ . Если текущим является момент времени  $t$ , то следующий момент времени часто обозначают  $t+1$ .



число  $m$ .

**В словесной форме** алгоритм решения задачи можно описать следующим образом:

Если  $n < m$ , то считать результатом число  $n$  и завершить процесс. В противном случае перейти к п.2.

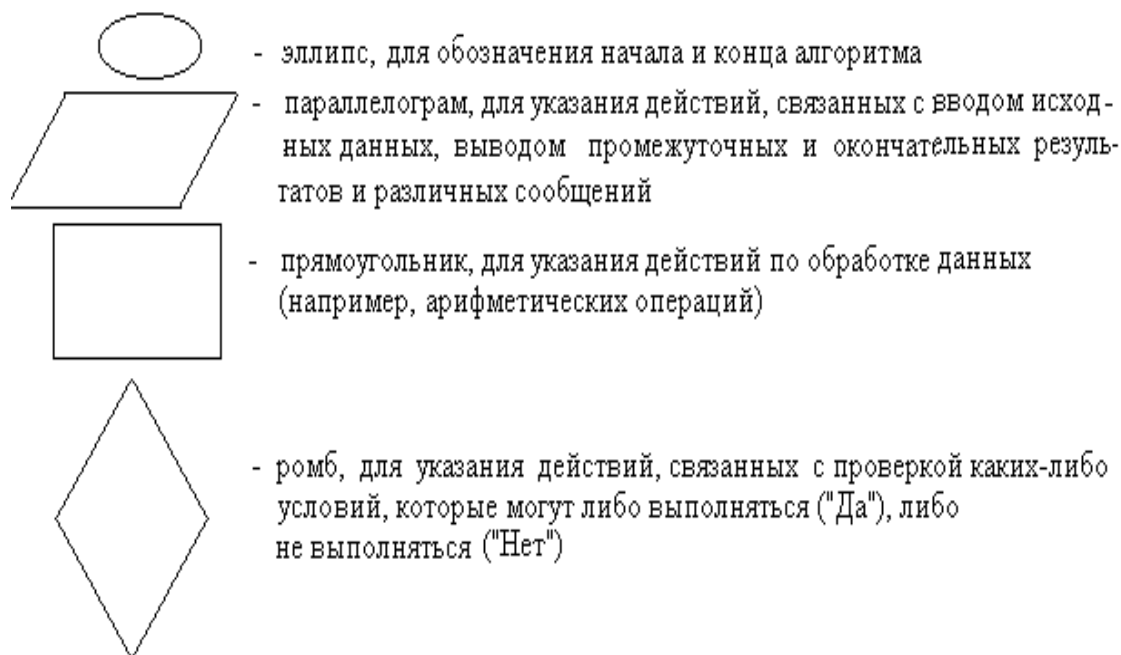
Положить  $r = n$ .

Если  $r = m$ , то считать результатом число 0 и завершить процесс. В противном случае перейти к п.4.

Вычислить  $r = r - m$ .

Если  $r < m$ , то считать результатом число  $r$  и завершить процесс. В противном случае перейти к п. 4.

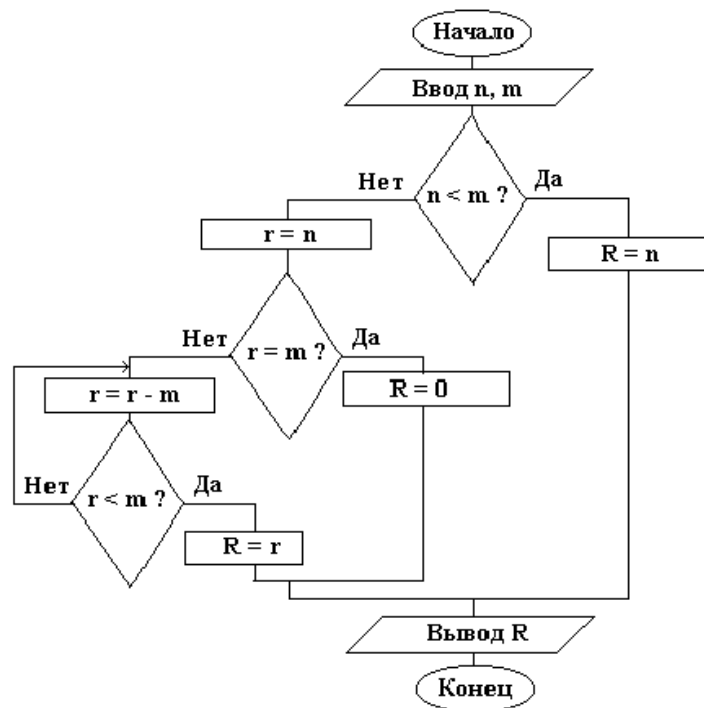
При описании алгоритмов в виде графических схем (блок-схем) исходят из следующего набора элементов - геометрических фигур, каждая из которых имеет определенное назначение (рис.1).



**Рис. 1. Базовые графические элементы для представления алгоритмов в виде блок-схем**

Эти фигуры соединяются между собой с помощью стрелок, указывающих порядок выполнения действий. Стрелки считаются ориентированными сверху вниз и слева направо. Если направление стрелок иное, то на стрелке ставится значок  $\leftarrow$  или  $\uparrow$ .

**Блок-схема алгоритма** решения задачи нахождения остатка от деления натурального числа  $n$  на натуральное число  $m$  представлена на рис. 2.



**Рис. 2. Блок-схема алгоритма нахождения остатка от деления натурального числа  $n$  на натуральное число  $m$**

*На алгоритмическом языке Pascal* запись рассматриваемого алгоритма имеет вид

```

function Rest (n,m: integer): integer;
var  r: integer;
label 1;
begin
  if n<m then Rest := n
  else
    begin
      r := n;
      if r = m then Rest := 0
      else begin
        1: r := r - m;
        if r < m then Rest := r
        else goto 1
      end
    end
  end
end.

```

Все три описания семантически (т. е. по смыслу) эквивалентны между собой.

## Лекция 13. Виды алгоритмических процессов

### Учебные вопросы:

1. Линейные и разветвляющиеся алгоритмические процессы.
2. Циклические алгоритмические процессы.

### 1. Линейные и разветвляющиеся алгоритмические процессы

Процесс обработки информации будем называть *алгоритмическим*, если существует алгоритм, описывающий этот процесс.

Различают три основных вида алгоритмических процессов:

- а) *линейные*;
- б) *разветвляющиеся*;
- в) *циклические*.

Процесс обработки информации называют линейным, если составляющие его действия выполняются последовательно друг за другом, т.е. "в одну линию". Графически линейный процесс представлен на рис.1.

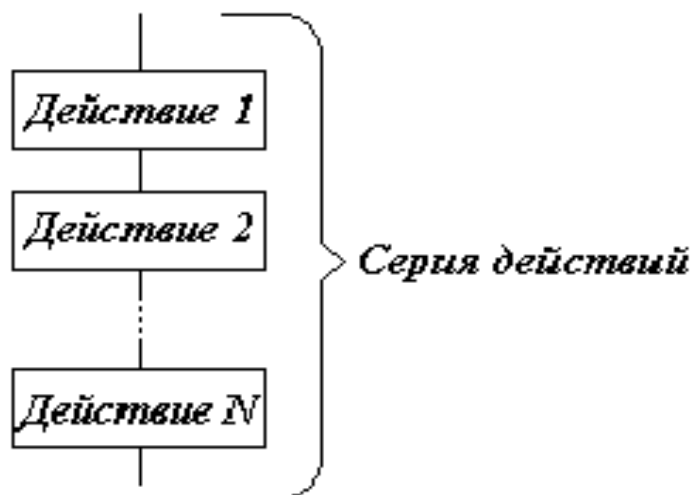
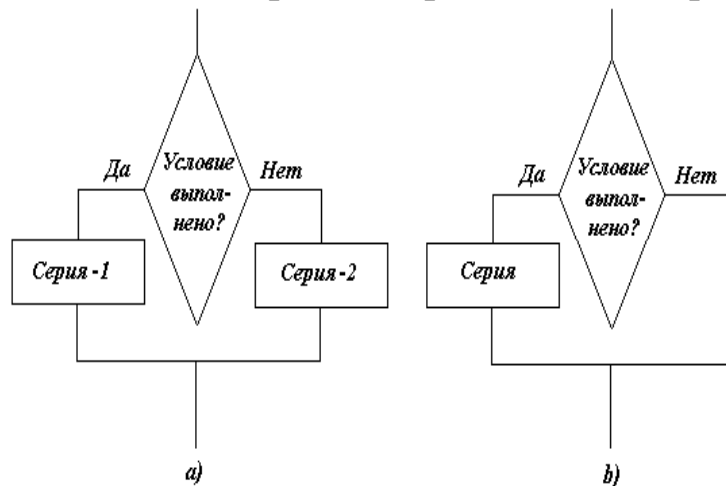


Рис. 1. Общий вид линейного процесса обработки информации

Последовательность действий линейного процесса называют серией действий или просто *серией*.

В простейшем случае линейный процесс обработки информации - это вычисление по некоторой формуле или формулам.

Процесс обработки информации называют **разветвляющимся**, если в ходе его осуществляется проверка некоторого условия, в зависимости от результата которой выполняется та или иная серия действий, т.е. процесс продолжается по той или иной "ветви". Графически разветвляющийся процесс представлен на рис. 2 .

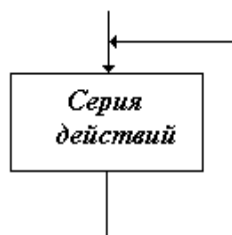


**Рис. 2. Графическое изображение разветвления процесса обработки информации: а) - разветвление типа "Если..., то..., иначе..."; б) - типа "Если..., то..."**

## 2. Циклические алгоритмические процессы

Процесс обработки информации называют **циклическим**, если в нем имеется повторяющаяся часть, или участок. Такой участок процесса называют **циклическим участком** или просто **циклом**.

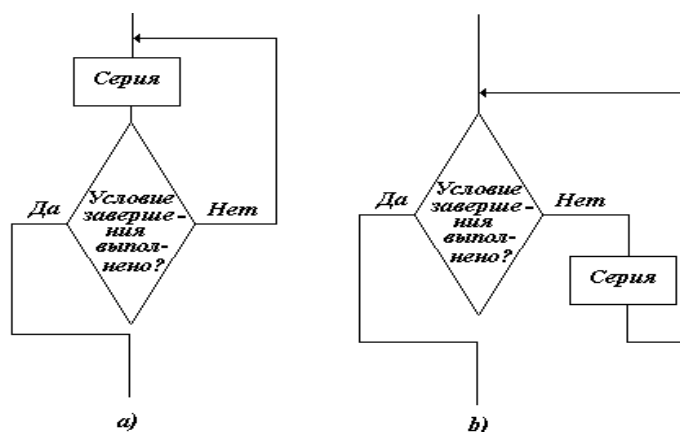
Графически простейший цикл представлен на рис. 3. Это так называемый "бесконечный" цикл. Он фактически "неуправляем". Процесс, содержащий такой участок, никогда не закончится. С точки зрения практики обычного программирования таких участков следует избегать. А для этого надо уметь управлять циклом.



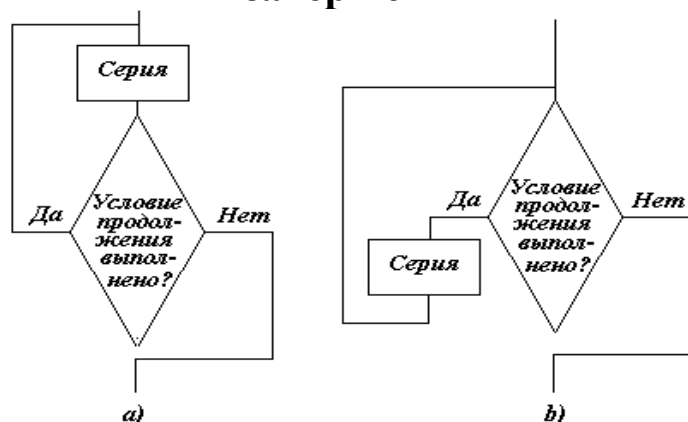
**Рис. 3. Простейший "бесконечный" цикл**

Возможны различные способы управления циклом. В основе любого из них лежит проверка некоторого условия, в зависимости от результата которой цикл либо завершается, либо продолжается. Проверяемое условие может быть сформулировано либо как условие завершения цикла, либо как условие продолжения цикла. В первом случае при выполнении условия осуществляется выход из цикла; если условие не выполняется, то цикл продолжается. Во втором случае при выполнении условия цикл продолжается; если условие не выполняется, то осуществляется выход из цикла. Таким образом, условия завершения и продолжения цикла являются противоположными: первое является отрицанием второго, и наоборот.

Проверка условия может осуществляться либо после выполнения серии действий, составляющей **"тело"** цикла, либо до выполнения серии действий. Различные способы управления циклом приведены на рис. 4, 5.



**Рис. 4. Схемы управления циклом посредством условия завершения**



**Рис. 5. Схемы управления циклом посредством условия продолжения**

Выбор условий завершения или продолжения цикла определяется содержанием конкретного процесса обработки информации. Относительно любого цикла можно поставить вопрос: "Известно ли заранее (т.е. до начала цикла) число повторений цикла?". В зависимости ответа на этот вопрос строится та или иная стратегия управления циклом: управление циклом с заданным числом повторений или управление циклом, число повторений которого заранее неизвестно.

Можно показать, что любой из перечисленных способов управления циклом может быть сведен к любому, возможно, за счет некоторой избыточности. Чаще всего используется схема, приведенная на рис. 8.b. Она соответствует обороту "Пока выполняется условие  $\Phi$ , выполняй действие  $F$ ".

Графические элементы, описывающие линейный процесс (следование), ветвление и цикл, называют *базовыми элементами*. Каждый из них характеризуется тем, что имеет **1** вход и **1** выход. Это дает возможность "конструировать" из этих элементов любые алгоритмы обработки информации путем взаимной подстановки одного элемента в другой по схеме: выход одного "подключается" к входу другого. Эта операция носит название *суперпозиции*. Посредством суперпозиции базовых элементов можно описать любой алгоритмический процесс. Это утверждение не может быть строго доказано, поскольку основывается на интуитивном, нестрогом понятии алгоритма.

## Лекция 14. Уточнение понятия алгоритма

### Учебные вопросы:

1. Различные подходы к определению алгоритма.
2. Машина Тьюринга.

#### 1. Различные подходы к определению алгоритма

В истории математики накопилось много случаев длительных и часто безрезультатных поисков тех или иных алгоритмов. При этом естественно возникало сомнение в существовании алгоритма. Одним из ярких примеров такого случая является история решения десятой проблемы Д. Гильберта.

В 1900 году на втором международном математическом конгрессе в Париже немецкий математик Давид Гильберт огласил список 23 трудных проблем, на важность решения которых он обращал внимание математической общественности. Среди них была и следующая 10-ая проблема Гильберта: требуется выработать алгоритм, позволяющий для любого диофантова уравнения выяснить, имеет ли оно целочисленное решение.

Рассмотрим всевозможные диофантовы уравнения, т.е. уравнения вида  $P = 0$ , где  $P$  является многочленом с целочисленными коэффициентами. Такими будут, например, уравнения  $x^2 + y^2 - 2xz = 0$ ,  $10x^5 + 7x^2 + 5 = 0$ , из которых первое с тремя неизвестными, а второе с одним неизвестным. В общем случае рассматривают уравнения с любым числом неизвестных. Такие уравнения могут иметь целочисленные решения, а могут и не иметь.

Так, уравнение  $x^2 + y^2 - 2xz = 0$  имеет бесконечное множество целочисленных решений, а уравнение  $10x^5 + 7x^2 + 5 = 0$  таких решений не имеет.

Для частного случая диофантова уравнения с одним неизвестным давно известен алгоритм, позволяющий найти все его целочисленные решения. Установлено, что если уравнение

$$a_0x^n + a_1x^{n-1} + \dots + a_{n-1}x + a_n = 0$$

с целочисленными коэффициентами имеет целый корень, то он обязательно является делителем  $a_n$ . В связи с этим можно предложить такой алгоритм:

- 1) Найти все делители числа  $a_n$ :  $d_1, d_2, \dots, d_k$ .
- 2) Вычислить  $P_n(x)$  для каждого из делителей числа  $a_n$
- 3) Если при некотором  $i$  из совокупности  $1, 2, \dots, k$   $P_n(d_i) = 0$ , то  $d_i$  — корень уравнения. Если при всех  $i = 1, 2, \dots, k$   $P_n(d_i) \neq 0$ , то уравнение не имеет целочисленных решений.

Поиски решения десятой проблемы Гильберта привлекли внимание многих математиков и длились около 70 лет. Только в 1968 году молодым математиком Ю. Матиясевичем было доказано, что нет алгоритма, дающего решение поставленной задачи.

Интуитивное определение алгоритма хотя и не строгое, но настолько ясное, что не дает оснований для сомнений в тех случаях, когда речь идет о найденном алгоритме решения конкретной задачи.

Положение существенно меняется, когда возникает алгоритмическая проблема, решение которой не найдено, и требуется установить, имеет ли она решение.

Действительно, в этом случае нужно либо доказать существование алгоритма, либо доказать его отсутствие.

В первом случае достаточно дать описание фактического процесса, решающего задачу. В этом случае достаточно и интуитивного понятия алгоритма, чтобы удостовериться в том, что описанный процесс есть алгоритм.

Во втором случае нужно доказать несуществование алгоритма, а для этого нужно точно знать, что такое алгоритм. Между тем для общего понятия алгоритма точного определения до тридцатых годов XX века не было, и поэтому выработка такого определения стала одной из важных задач современной математики. При формулировке этого определения пришлось преодолеть многие трудности.

Во-первых, такое определение должно было правильно отражать сущность интуитивного определения алгоритма.

Во-вторых, оно должно было быть совершенным с точки зрения формальной точности.

Наконец, различные исследователи этой проблемы исходили из разных технических и логических соображений, и вследствие



этого было выработано несколько определений алгоритма. Однако со временем выяснилось, что все эти определения равносильны, т.е. определяют одно и то же понятие. Это и есть современное понятие алгоритма.

В подходах к определению понятия алгоритма можно выделить три основных направления.

Первое направление связано с уточнением понятия эффективно вычислимой функции. Этим занимались А. Черч, К. Гедель, С. Клини. В результате был выделен класс так называемых частично-рекурсивных функций, имеющих строгое математическое определение. Анализ идей, приведших к этому классу функций, дал им возможность высказать гипотезу о том, что класс эффективно вычисляемых функций совпадает с классом частично рекурсивных функций.

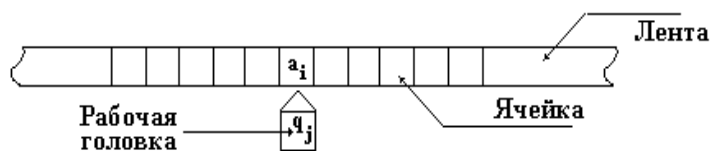
Второе направление связано с машинной математикой. Здесь сущность понятия алгоритма раскрывается путем рассмотрения процессов, осуществляемых в машине. Впервые это было сделано Тьюрингом, который предложил самую общую и вместе с тем самую простую концепцию вычислительной машины. Ее описание было дано Тьюрингом в 1937 году. При этом Тьюринг исходил лишь из общей идеи работы машины как работы вычислителя, оперирующего в соответствии с некоторым строгим предписанием.

Третье направление связано с понятием нормальных алгоритмов, введенным и разработанным российским математиком А. А. Марковым.

## 2. Машина Тьюринга

Различные формализации понятия алгоритма эквивалентны с точки зрения класса описываемых ими процессов. Поэтому теорию алгоритмов можно излагать на базе любого из формализмов. Рассмотрим уточнение понятия алгоритма, предложенное А. Тьюрингом в виде абстрактной "вычислительной" машины.

**Машина Тьюринга** состоит из бесконечной в обе стороны *ленты*, разбитой на *ячейки*, и *рабочей головки* (рис. 1.)



**Рис.1. «Устройство» машины Тьюринга**

Машина работает в дискретные моменты времени:  $t_0, t_1, t_2, \dots, t_n, \dots$ . В каждый момент во всякой ячейке ленты может быть записана одна из букв некоторого конечного алфавита  $A = \{a_0, a_1, \dots, a_{k-1}\}$ , называемого *внешним алфавитом* машины, а головка может находиться в одном из конечного числа внутренних состояний  $Q = \{q_0, q_1, \dots, q_{r-1}\}$ . ( $Q$  называют *внутренним алфавитом*.) Условимся, что символ  $a_0$  является "пустым" (будем обозначать его  $\Lambda$ <sup>2)</sup>). Предполагаем, что все клетки ленты заполнены этим символом, если не оговорено иное. Наличие символа  $\Lambda$  в клетке содержательно означает, что в ней ничего не записано, т.е. она пустая.

В каждый момент времени рабочая головка обзореваает одну ячейку ленты. В зависимости от того, что в ней записано и в каком внутреннем состоянии находится рабочая головка<sup>3)</sup>, она выполняет следующие действия:

- 1) заменяет символ в обзореваемой ячейке новым (возможно, прежним);
- 2) переходит в новое состояние (возможно, в прежнее);
- 3) сдвигается на одну ячейку (вправо или влево) либо остается на месте.

Таким образом, работа машины задается системой команд вида:  $q_i a_j \rightarrow q_l a_p D$ , где  $D \in \{R, L, H\}$ .<sup>4)</sup>

Эта команда интерпретируется следующим образом: если в состоянии  $q_i$  головка читает символ  $a_j$ , то в следующий момент времени она записывает в эту ячейку символ  $a_p$ , переходит в состояние  $q_l$  и, в зависимости от  $D$ , сдвигается на одну ячейку вправо ( $R$ ), влево ( $L$ ) или остается на месте ( $H$ ).

<sup>2)</sup> Читается: "лямбда".

<sup>3)</sup> Состояния головки иногда называют состояниями машины.

<sup>4)</sup>  $R$  (от right - правый) - сдвиг головки вправо на одну ячейку,  $L$  (от left - левый) - сдвиг головки влево на одну ячейку,  $H$  (от halt - стоять) - отсутствие сдвига, то есть головка остается на месте (возможно лишь изменяет обзореваемый символ и/или внутреннее состояние).

Среди состояний машины (головки) выделено одно, называемое **заключительным** (впредь будем считать, что это состояние  $q_0$ ). Если после некоторого шага головка оказывается в состоянии  $q_0$ , то машина прекращает свою работу (останавливается).

Будем предполагать, что для каждой пары  $q_i a_j$  ( $q_i \in Q \setminus \{q_0\}$ ,  $a_j \in A$ ) имеется ровно одна команда с левой частью  $q_i a_j$ , так что всего имеется  $(r-1) \cdot k$  команд. ***Совокупность этих команд будем называть программой машины Тьюринга.***

***Под конфигурацией машины Тьюринга понимается распределение букв алфавита  $A$  по ячейкам ленты, состояние головки и обозреваемую ячейку.***

Работа машины Тьюринга по программе состоит в смене конфигураций. Конфигурацию в момент времени  $t_i$  будем обозначать  $Kt_i$ . Если эта конфигурация не является заключительной, то машина в соответствии со следующей командой переходит в конфигурацию  $Kt_{i+1}$ .

Всякая начальная конфигурация  $Kt_0$  порождает последовательность конфигураций  $Kt_0, Kt_1, \dots, Kt_i, \dots$ . В случае, когда эта последовательность обрывается некоторой заключительной конфигурацией, будем говорить, что машина ***применима*** к конфигурации  $Kt_0$ , в противном случае - ***неприменима***.

Если нужно решить некоторую задачу на машине Тьюринга, исходным данным задачи сопоставляется начальная конфигурация  $Kt_0$ , а ответ задачи определяется заключительной конфигурацией, в которую программа машины переводит конфигурацию  $Kt_0$ .

## Лекция 15. Вычисления на машине Тьюринга

### Учебные вопросы:

#### 1. Вычисления на машинах Тьюринга. Примеры машин Тьюринга.

Обычно в начальной конфигурации на ленте машины записано лишь конечное число непустых символов. При этом все последующие конфигурации будут обладать тем же свойством. **Активной зоной** данной конфигурации называется минимальная связная часть ленты, содержащая обозреваемую ячейку и все ячейки, в которых записаны непустые символы. В конфигурациях, представленных на рис. 1. активные зоны выделены серым цветом:

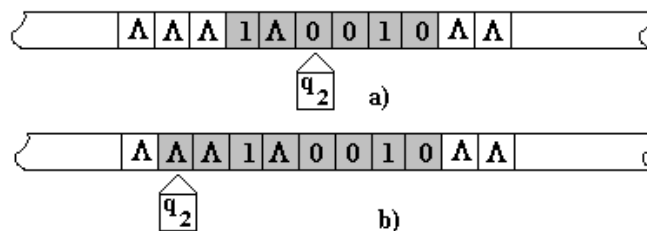


Рис. 1. Иллюстрация активной зоны конфигурации машины Тьюринга

Пусть активная зона имеет длину  $p$ , в ней записано слово  $a^{(1)}a^{(2)} \dots a^{(1)} \dots a^{(p)}$  и головка обозревает ячейку с символом  $a^{(1)}$ , находясь в некотором состоянии  $q_s$ . Такую конфигурацию будем задавать **словом** длины  $p+1$ :  $a^{(1)}a^{(2)} \dots a^{(1-1)}q_s a^{(1)} \dots a^{(p)}$ , в котором состояние головки указывается перед обозреваемым символом. В частности, конфигурациям, изображенным на рис. 1, соответствуют слова: а) -  $1\Lambda q_2 0010$ ; б) -  $q_2 \Lambda \Lambda 1 \Lambda 0010$ . В последующем мы не будем различать конфигурации и связанные с ними слова.

**Пример 1.** Пусть требуется добавить 1 к натуральному числу  $n$ , представленному на ленте машины Тьюринга в двоичной системе счисления, то есть в алфавите  $\{0,1\}$ . Иными словами, требуется построить машину Тьюринга, прибавляющую 1 к двоичному числу. Заметим, что записанное на ленте двоичное число  $\sigma_1\sigma_2 \dots \sigma_i \dots \sigma_i \dots \sigma_p$ , где  $\sigma_i \in \{0,1\}$  ( $i=1, 2, \dots, p$ ) слева и справа ограничено пу-

стыми символами  $\Lambda$ . Таким образом, внешний алфавит машины будет следующим:  $\{\Lambda, 0, 1\}$ .

Пусть в начальный момент времени  $t_0$  головка машины в состоянии  $q_1$  находится под первым непустым символом активной зоны, то есть начальная конфигурация имеет вид:  $q_1\sigma_1...\sigma_p$ . Добавим явно слева и справа к этому слову по пустому символу, чтобы упростить построение, то есть будем считать начальной следующей конфигурацию:  $\Lambda q_1\sigma_1...\sigma_p\Lambda$ .

Прежде, чем составить программу работы машины, представим процедуру добавления  $1$  к двоичному числу  $n$  на неформальном, содержательном уровне, но "мысля" уже в терминах машины Тьюринга.

Итак, для того, чтобы прибавить  $1$  к двоичному числу  $n$  сначала необходимо "отогнать" головку машины вправо и установить ее под последней (самой младшей) цифрой двоичного числа. Теперь далее, если последняя цифра числа есть  $0$ , то достаточно заменить  $0$  на  $1$  и завершить процесс, то есть перевести головку (машину) в заключительное состояние  $q_0$ .

Если же последняя цифра числа есть  $1$ , то необходимо заменить ее на  $0$ , а головку сдвинуть влево, чтобы "увидеть" следующий разряд двоичного числа. Если окажется, что этот разряд содержит  $0$ , то заменить  $0$  на  $1$  и опять-таки перевести головку (машину) в заключительное состояние  $q_0$ . Если же этот разряд содержит  $1$ , необходимо заменить ее на  $0$  и опять сдвинуть головку влево. И так далее, до тех пор, пока либо не встретится разряд, содержащий  $0$ , либо головка дойдет до первого слева пустого символа  $\Lambda$ . В любом из этих случаев  $0$  или  $\Lambda$  следует заменить на  $1$  и перевести головку (машину) в заключительное состояние  $q_0$ .

Итак, путем многократного применения команд вида

$$q_1 1 \rightarrow q_1 1 R$$

$$q_1 0 \rightarrow q_1 0 R$$

головка машины перемещается вправо, пока в состоянии  $q_1$  не дойдет до первого справа пустого символа  $\Lambda$ . В этом случае надо оставить пустой символ без изменения, а головку сдвинуть влево и перейти в новое состояние  $q_2$ , что достигается командой:  $q_1 \Lambda \rightarrow q_2 \Lambda L$ . Оставаться в прежнем состоянии  $q_1$  нельзя, поскольку какая-то из приведенных выше команд обязательно вновь переместит головку

вправо, а команда  $q_1\Lambda \rightarrow q_2\Lambda L$  вновь вернет головку влево, что приведет фактически к "заикливлению" и машина никогда не остановится. В результате головка машины установится под младшей цифрой двоичного числа  $n$  в состоянии  $q_2$ .

**Команды вида**

$$q_2 0 \rightarrow q_0 1 H$$

$$q_2 1 \rightarrow q_2 0 L$$

$$q_2 \Lambda \rightarrow q_0 1 H$$

обеспечивают в соответствии с описанной выше неформальной процедурой прибавление 1 и остановку машины.

Таким образом, программа работы машины будет следующей:

$$q_1 1 \rightarrow q_1 1 R$$

$$q_1 0 \rightarrow q_1 0 R$$

$$q_1 \Lambda \rightarrow q_2 \Lambda L$$

$$q_2 0 \rightarrow q_0 1 H$$

$$q_2 1 \rightarrow q_2 0 L$$

$$q_2 \Lambda \rightarrow q_0 1 H$$

Начальная конфигурация:  $\Lambda q_1 \sigma_1 \dots \sigma_p \Lambda$ .

Заключительная конфигурация:

$$\Lambda \sigma_1 \sigma_2 \dots \sigma_{i-1} q_0 \sigma'_i \dots \sigma'_p \Lambda \text{ или } \Lambda q_0 1 \sigma'_1 \sigma'_2 \dots \sigma'_{i-1} \sigma'_i \dots \sigma'_p \Lambda.$$

(Здесь  $\sigma'_i = 1$ , если  $\sigma_i = 0$  и  $\sigma'_i = 0$ , если  $\sigma_i = 1$ ).

Условимся, после выполнения необходимых действий машиной Тьюринга, перед переходом в заключительное состояние "отгонять" головку машины влево и устанавливать под первым непустым символом активной зоны, содержащей результат работы машины.

В соответствии с этим требованием программа машины, прибавляющей 1 к двоичному числу, будет иметь вид:

$$q_1 1 \rightarrow q_1 1 R$$

$$q_1 0 \rightarrow q_1 0 R$$

$$q_1 \Lambda \rightarrow q_2 \Lambda L$$

$$q_2 0 \rightarrow q_3 1 H$$

$$q_2 1 \rightarrow q_2 0 L$$

$$q_2 \Lambda \rightarrow q_0 1 H$$

$$\begin{aligned} q_3 1 &\rightarrow q_3 1 L \\ q_3 0 &\rightarrow q_3 0 L \\ q_3 \Lambda &\rightarrow q_0 \Lambda R. \end{aligned}$$

Для выполнения этого требования пришлось ввести дополнительное состояние  $q_3$ , в котором головка машины после прибавления 1 к числу прошла влево до первого пустого символа и "оттолкнувшись" от него стала в заключительном состоянии  $q_0$  под первым непустым символом результата.

**Пример 2.** Пусть требуется перевести унарную запись натурального числа  $n$ , изображенного в виде последовательности  $n$  "палочек" ( $|$ ), в двоичную запись в алфавите  $\{0,1\}$ . Другими словами, надо построить машину Тьюринга, которая конфигурацию  $\Lambda q_1 ||| \dots | \Lambda$  (где число "палочек" равно  $n \geq 1$ ) преобразовывала бы в конфигурацию  $\Lambda q_0 \sigma_1 \sigma_2 \dots \sigma_p \Lambda$ , где  $\sigma_1 \sigma_2 \dots \sigma_p$  - двоичная запись числа  $n$ .

В качестве внешнего алфавита машины берем алфавит:  $\{\Lambda, |, 0, 1\}$ .

Опишем процесс построения соответствующей машины  $M_2$  сначала на неформальном, содержательном уровне.

Начальная конфигурация говорит о том, что головка машины в начальный момент времени  $t_0$  в состоянии  $q_1$  находится под первым символом  $|$  исходной записи числа  $n$  в унарной системе счисления.

Идея перевода при построении машины будет следующей: отнимаем 1 от числа, представленного в унарной системе счисления и прибавляем 1 к получаемому в процессе перевода числу в двоичной системе счисления до тех пор, пока не исчерпаем исходное число. Результатом будет то же исходное число, но представленное в двоичной системе счисления.

Неформализованная процедура решения поставленной задачи перевода может быть описана следующим образом:

1. "Отогнуть" головку машины влево до первого пустого символа, заменить этот символ нулем (0).
2. "Отогнуть" головку машины вправо до последней черточки, заменить ее пустым символом. Запомнить, что 1 из унарного представления числа  $n$  вычтена.

3. Установить головку под младшим разрядом формируемого двоичного числа и прибавить к двоичному числу **1** (так как мы делали это при построении предыдущей машины). Запомнить, что **1** к двоичному числу прибавлена.

4. Пункты 2 и 3 повторять до тех пор, пока не исчерпается исходное число, то есть на ленте не останется "палочек".

5. Головку отогнать в крайнюю левую позицию полученного двоичного числа и остановить машину.

Итак, начальная конфигурация имеет вид:  $\Lambda q_1 ||| \dots | \Lambda$ . Командой  $q_1 | \rightarrow q_1 | L$  головка машины "отгоняется" влево до первого пустого символа, а командой  $q_1 \Lambda \rightarrow q_2 0 H$  этот пустой символ заменяется нулем, и машина переходит в состояние  $q_2$ .

Теперь в состоянии  $q_2$  многократным применением команды  $q_2 | \rightarrow q_2 | R$  головка машины "отгоняется" вправо до первого пустого символа, а командой  $q_2 \Lambda \rightarrow q_3 \Lambda L$  устанавливается под последней "палочкой". Команда  $q_3 | \rightarrow q_4 \Lambda L$  "стирает" эту черточку, запоминает это, переходя в новое состояние  $q_4$ , и сдвигает головку влево на одну позицию.

Команда  $q_4 | \rightarrow q_4 | L$  перемещает головку машины влево, пока в состоянии  $q_4$  не встретится первая двоичная цифра (в первый раз это будет **0**, установленный командой  $q_1 \Lambda \rightarrow q_2 0 H$ ).

После чего команды

$q_4 0 \rightarrow q_5 1 R; q_4 1 \rightarrow q_4 0 L; q_4 \Lambda \rightarrow q_5 1 R$

обеспечат прибавление **1** к формируемому двоичному числу и перевод головки машины в состояние  $q_5$ . Это состояние говорит нам о том, что **1** к двоичному числу прибавлена.

Команды

$q_5 1 \rightarrow q_5 1 R; q_5 0 \rightarrow q_5 0 R$

"прогонят" головку машины через двоичное число до первой "палочки", после чего команда  $q_5 | \rightarrow q_2 | H$  "зацикливает" процесс "качания" головки вправо-влево, переводя ее в ранее встречавшееся состояние  $q_2$ .

Процесс перевода завершится тогда, когда головка машины в состоянии  $q_5$  будет обозревать пустой символ (то есть все "палочки" вытерты). Поэтому команда  $q_5 \Lambda \rightarrow q_6 \Lambda L$  установит головку машины под последней цифрой двоичного числа. К этому времени на ленте останется лишь результат перевода – двоичное число  $n$ .



Чтобы добиться требуемой заключительной конфигурации  $\Lambda q_0 \sigma_1 \sigma_2 \dots \sigma_p \Lambda$ , где  $\sigma_1 \sigma_2 \dots \sigma_p$  - двоичная запись числа  $n$ , начинающаяся с 1, достаточно многократными применениями команд

$$q_6 1 \rightarrow q_6 1 L$$

$$q_6 0 \rightarrow q_6 0 L$$

"отогнать" головку машины до первого слева пустого символа, а затем командой  $q_6 \Lambda \rightarrow q_0 \Lambda R$  остановить машину.

Таким образом, программа работы машины имеет вид:

|   |   |
|---|---|
| $q_1   \rightarrow q_1   L$             | $q_4 \Lambda \rightarrow q_5 1 R$       |
| $q_1 \Lambda \rightarrow q_2 0 R$       | $q_5 1 \rightarrow q_5 1 R$             |
| $q_2   \rightarrow q_2   R$             | $q_5 0 \rightarrow q_5 0 R$             |
| $q_2 \Lambda \rightarrow q_3 \Lambda L$ | $q_5   \rightarrow q_2   H$             |
| $q_3   \rightarrow q_4 \Lambda L$       | $q_5 \Lambda \rightarrow q_6 \Lambda L$ |
| $q_4   \rightarrow q_4   L$             | $q_6 1 \rightarrow q_6 1 L$             |
| $q_4 0 \rightarrow q_5 1 R$             | $q_6 0 \rightarrow q_6 0 L$             |
| $q_4 1 \rightarrow q_4 0 L$             | $q_6 \Lambda \rightarrow q_0 \Lambda R$ |

В приведенных программах машин Тьюринга левые части команд все различны, поэтому в каждый конкретный момент времени может выполняться лишь одна команда программы. Кроме того, в приведенных программах присутствуют лишь команды для тех пар  $q_i a_j$ , которые встречаются при работе машины. Остальные команды несущественны и могут быть взяты произвольно.

## Лекция 16. Примеры машин Тьюринга и способы их задания

Учебные вопросы:

1. Примеры машин Тьюринга.
2. Задание машины Тьюринга в виде таблицы.
3. Машины Тьюринга как словарные функции.

### 1. Примеры машин Тьюринга

**Пример 3.** Составить программу машины Тьюринга, подсчитывающую число вхождений символа **a** в слово **P** в алфавите  $\{a, b, c\}$ .

Пусть начальная конфигурация машины имеет вид  $q_1P$ .

Наша задача перевести ее в конфигурацию  $q_0n^*P$ , где **n** – двоичное число, выражающее число вхождений символа **a** в слово **P** в алфавите  $\{a, b, c\}$ .

Внешний алфавит машины:  $A = \{a, b, c, a', 0, 1, *, \Lambda\}$ .

Внутренний алфавит машины:  $Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7\}$ .

Неформальное описание машины.

1. Слева от слова **P** приписываем символы **0** и **\***.
2. Находим в слове **P** вхождение символа **a**, заменяем его на **a'**, запоминаем, перемещаем головку влево, прибавляем 1 к двоичному числу **n** («счетчику»).
3. Повторяем п. 2 до тех пор, пока не пройдем все слово **P**.
4. Убираем все штрихи в слове **P**.
5. Устанавливаем головку машины под крайней левой цифрой двоичного числа **n** и останавливаем машину.

Программа:

|                                |                            |                                       |                                       |
|--------------------------------|----------------------------|---------------------------------------|---------------------------------------|
| $q_1a \rightarrow q_2aL$       | $q_4* \rightarrow q_5*R$   | $q_6a' \rightarrow q_6a'L$            | $q_7b \rightarrow q_7bL$              |
| $q_1b \rightarrow q_2bL$       | $q_5b \rightarrow q_5bR$   | $q_6* \rightarrow q_6*L$              | $q_7c \rightarrow q_7cL$              |
| $q_1c \rightarrow q_2cL$       | $q_5c \rightarrow q_5cR$   | $q_60 \rightarrow q_41R$              | $q_7a' \rightarrow q_7aL$             |
| $q_2\Lambda \rightarrow q_3*L$ | $q_5a' \rightarrow q_5a'R$ | $q_61 \rightarrow q_60L$              | $q_7* \rightarrow q_7*L$              |
| $q_3\Lambda \rightarrow q_40R$ | $q_5a \rightarrow q_6aH$   | $q_6\Lambda \rightarrow q_41L$        | $q_70 \rightarrow q_70L$              |
| $q_40 \rightarrow q_40R$       | $q_6b \rightarrow q_6bL$   | $q_5\Lambda \rightarrow q_7\Lambda L$ | $q_71 \rightarrow q_71L$              |
| $q_41 \rightarrow q_41R$       | $q_6c \rightarrow q_6cL$   | $q_7a \rightarrow q_7aL$              | $q_7\Lambda \rightarrow q_0\Lambda R$ |

**Пример 4.** Составить программу машины Тьюринга, находящую (и выделяющую штрихами) первое вхождение подслова **abba** в данное слово **P** в алфавите  $\{a, b\}$  или сообщающую о том, что подслово **abba** в слово **P** не входит. Последнее машина сообщает, записывая слово «нет» на ленте после слова **P**.

Пусть начальная конфигурация машины имеет вид  $q_1P$ .

Задача заключается в переводе ее в конфигурацию  $q_0Qa'b'b'a'S$ , ( $Q$  и  $S$  – слова в алфавите  $\{a, b\}$ ), если подслово **abba** входит в слово **P**, или  $q_0P*$ нет – если подслово **abba** не входит в слово **P**.

Внешний алфавит машины:  $A = \{a, b, a', b', *, \Lambda, н, е, т\}$ .

Внутренний алфавит машины:

$Q = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8, q_9, q_{10}, q_{11}, q_{12}\}$ .

Неформальное описание машины.

1. Перемещаем головку вправо по слову **P**. Пропускаем символы **b**, не меняя начального состояния  $q_1$ , но как только встречаем первый символ **a**, помечаем его (ставя штрих) и меняем состояние на  $q_2$ .

Если в состоянии  $q_2$  встречаем символ **b**, то помечаем его (ставя штрих) и меняем состояние на  $q_3$  (это состояние говорит о том, что найдено подслово **ab**). Если же в состоянии  $q_2$  встречаем символ **a**, то останавливаемся под ним и возвращаемся в исходное состояние  $q_1$ .

Если в состоянии  $q_3$  встречаем символ **b**, то помечаем его (ставя штрих) и меняем состояние на  $q_4$  (это состояние говорит о том, что найдено подслово **abb**). Если же в состоянии  $q_3$  встречаем символ **a**, то это означает «неудачу» поиска подслова **abba**, и все надо начинать сначала: то есть, остановить головку под последним встреченным символом **a** и вернуться в состояние  $q_1$ .

Если в состоянии  $q_4$  встречаем символ **a**, то помечаем его (ставя штрих) и меняем состояние на  $q_5$  (это состояние говорит о том, что найдено первое вхождение подслова **abba** в слово **P**). Если же в состоянии  $q_4$  встречаем символ **b**, то это означает «неудачу» поиска подслова **abba**, и все опять надо начинать сначала, то есть вновь возвращаться в состояние  $q_1$ .

2. Если машина оказалась в состоянии  $q_5$ , то следует, сдвигая головку влево, оставить штрихи над символами найденного вхождения подслова **abba** и *снять штрихи* над всеми другими поме-

ченными символами, остановить машину, установив головку под первым слева символом слова  $P$ , то есть перейти в заключительную конфигурацию вида  $q_0 Q a' b' b' a' S$ , ( $Q$  и  $S$  – слова в алфавите  $\{a, b\}$ ).

3. Если головка машины достигает первого справа пустого символа  $\Lambda$  в любом из состояний  $q_1, q_2, q_3, q_4$ , то это означает, что вхождений подслова **abba** в слово  $P$  не найдено. В этом случае справа к слову  $P$  необходимо (по условию заключительной конфигурации) приписать слово «нет», стереть в слове  $P$  все проставленные штрихи и остановить машину, установив головку под первым слева символом слова  $P$ , то есть перейти в заключительную конфигурацию  $q_0 P^* \text{нет}$ .

Программа:

|                                   |                                     |                                   |   |
|-----------------------------------|-------------------------------------|-----------------------------------|---|
| $q_1 a \rightarrow q_2 a' R$      | $q_3 a \rightarrow q_1 a H$         | $q_{11} b' \rightarrow q_2 b' L$  | $q_9 T \rightarrow$                     |
| $q_1 b \rightarrow q_1 b R$       | $q_3 \Lambda \rightarrow q_6^* R$   | $q_{12} a' \rightarrow q_9 a' L$  | $q_9 T L$                               |
| $q_1 \Lambda \rightarrow q_6^* R$ | $q_4 a \rightarrow q_5 a' H$        | $q_6 \Lambda \rightarrow q_7 H R$ | $q_9^* \rightarrow q_9^* L$             |
| $q_2 b \rightarrow q_3 b' R$      | $q_4 b \rightarrow q_1 b R$         | $q_7 \Lambda \rightarrow q_8 e R$ | $q_9 a \rightarrow$                     |
| $q_2 a \rightarrow q_1 a H$       | $q_4 \Lambda \rightarrow q_6^* R$   | $q_8 \Lambda \rightarrow q_9 T H$ | $q_9 a L$                               |
| $q_2 \Lambda \rightarrow q_6^* R$ | $q_5 a' \rightarrow q_{10} a' L$    | $q_9 H \rightarrow q_9 H L$       | $q_9 b \rightarrow$                     |
| $q_3 b \rightarrow q_4 b' R$      | $q_{10} b' \rightarrow q_{11} b' L$ | $q_9 e \rightarrow q_9 e L$       | $q_9 b L$                               |
|                                   |                                     |                                   | $q_9 a \rightarrow$                     |
|                                   |                                     |                                   | $q_9 a L$                               |
|                                   |                                     |                                   | $q_9 b' \rightarrow q_9 b L$            |
|                                   |                                     |                                   | $q_9 \Lambda \rightarrow q_0 \Lambda R$ |

Аналогично можно составлять программы для различных машин Тьюринга.

## 2. Задание машины Тьюринга в виде таблицы

Машину Тьюринга можно задать и в виде таблицы, подобно тому, как задавались конечные автоматы. В столбцах таблицы указываются состояния машины  $q_i$ , в строках - обозреваемые символы внешнего алфавита  $a_j$ , а на пересечение столбцов и строк - тройки  $\langle q_i', a_j', D \rangle$ , указывающие новое состояние, заменяющий символ и направление сдвига головки. Например, программа **примера 2**

предыдущей лекции может быть представлена в виде следующей таблицы:

| Символы<br>внешнего<br>алфавита | Состояния |         |                |                      |         |                |                |
|---------------------------------|-----------|---------|----------------|----------------------|---------|----------------|----------------|
|                                 | $q_0$     | $q_1$   | $q_2$          | $q_3$                | $q_4$   | $q_5$          | $q_6$          |
|                                 |           | $q_1 L$ | $q_2 R$        | $q_4$<br>$\Lambda L$ | $q_4 L$ | $q_2 H$        |                |
| $\Lambda$                       |           | $q_20H$ | $q_3\Lambda L$ |                      | $q_51R$ | $q_6\Lambda L$ | $q_0\Lambda R$ |
| 0                               |           |         |                |                      | $q_51R$ | $q_50R$        | $q_60L$        |
| 1                               |           |         |                |                      | $q_40L$ | $q_51R$        | $q_61L$        |

Пустые клетки в таблице указывают на то, что соответствующие пары  $q_i a_j$  никогда в процессе работы машины не встретятся. Если же встретятся (это может быть в случае некорректно составленной программы), то машина никогда не остановится ("зависнет"). Столбец с состоянием  $q_0$  из таблицы можно исключить.

### 3. Машины Тьюринга как словарные функции

Машина Тьюринга задается следующими параметрами:

- внешним алфавитом  $A = \{a_0, a_1, \dots, a_{k-1}\}$ ;
- внутренним алфавитом  $Q = \{q_0, q_1, \dots, q_{r-1}\}$ ;
- начальной конфигурацией  $K_0$ ;
- программой работы.

Как мы условились ранее и для упрощения последующих построений, будем считать, что начальной конфигурацией  $K_0$  всегда будет следующая:

$$q_1 a^{(1)} a^{(2)} \dots a^{(l-1)} a^{(l)} \dots a^{(p)} \quad (a^{(l)} \in A, l = 1, \dots, p),$$

т.е. в начальном состоянии  $q_1$  машина обзореваает самый левый символ активной зоны ленты.

Заключительную конфигурацию  $K_z$ , хотя она и не определяет машину Тьюринга, тоже будем "стандартизировать" следующим образом:

$$q_0 a^{(1)} a^{(2)} \dots a^{(l-1)} a^{(l)} \dots a^{(m)} \quad (a^{(l)} \in A, l = 1, \dots, m),$$

т.е. в заключительном состоянии  $q_0$  машина обозревает самый левый символ активной зоны ленты.

На ленте машины Тьюринга в начальный момент времени  $t_0$  записано какое-то слово  $P$  в алфавите  $A$ . В результате выполнения программы слово  $P$  перерабатывается машиной в какое-то (вполне определенное) слово  $S$  в том же алфавите  $A$ . **Таким образом, машина Тьюринга задает некоторое отображение слов в алфавите  $A$  в слова в том же алфавите.** Отображение это является функцией, чаще всего не всюду, а частично определенной.<sup>5)</sup> Если обозначить через  $A^*$  множество всех слов в алфавите  $A$ , то любая машина Тьюринга  $M$ , внешним алфавитом которой является множество  $A$ , отображает множество  $A^*$  во множество  $A^*$ , то есть:  $f_M : A^* \rightarrow A^*$ .

Такие функции называют словарными функциями. Поэтому можно говорить о том, что **машина Тьюринга  $M$  "вычисляет" некоторую частично определенную словарную функцию.**

Это вычисление понимается в следующем смысле:

Если в начальной конфигурации на ленте записано слово  $P$  и головка в состоянии  $q_1$  обозревает самый левый символ этого слова, то если значение  $f_M(P)$  определено, то после конечного числа шагов машина должна перейти в заключительную конфигурацию, в которой на ленте машины будет записано слово  $f_M(P)$  и головка в состоянии  $q_0$  находится у самого левого символа этого слова, то есть конфигурация  $q_1 P$  переводится в конфигурацию  $q_0 f_M(P)$ . В противном случае машина должна работать бесконечно, то есть никогда не останавливаться.

В первом случае говорят, что машина  $M$  применима к слову  $P$ , во втором- что неприменима.

<sup>5)</sup> Функция называется *частично определенной* на множестве  $X$ , если ее значения определены не на всех  $x \in X$ . Например, функция, задаваемая формулой  $y = \ln x$  частично определена на множестве всех действительных чисел и всюду определена на множестве положительных действительных чисел.

## Лекция 17. Методы построения программ для машины Тьюринга

Учебные вопросы:

1. Суперпозиция машин (программ).
2. Композиция машин (программ).

### 1. Суперпозиция машин (программ)

Для простоты изложения будем отождествлять понятия "машина Тьюринга" и "программа машины", поскольку конкретная машина задается программой ее работы.<sup>6)</sup>

Пусть имеются две машины Тьюринга  $M_1$  и  $M_2$ , которые соответственно вычисляют словарные функции  $f_1(P)$  и  $f_2(P)$  *в одном и том же алфавите*.<sup>7)</sup> Тогда можно построить машину  $M$ , вычисляющую суперпозицию функций  $f_2(f_1(P))$ . Значение суперпозиции  $f_2(f_1(P))$  считается определенным тогда и только тогда, когда определены  $f_1(P)$  и значение функции  $f_2$  на слове  $f_1(P)$ .

Программа машины  $M$  из программ машин  $M_1$  и  $M_2$  может быть получена следующим образом. Состояния машины  $M_2$  переобозначаются так, чтобы они, за исключением начального, были отличны от состояний машины  $M_1$ , а начальное совпадало с заключительным состоянием машины  $M_1$ . Программы приписываются друг к другу и начальным состоянием машины  $M$  объявляется начальное состояние машины  $M_1$ , а заключительным состоянием машины  $M$  - заключительное состояние машины  $M_2$ .

В качестве примера программы машины Тьюринга, построенной на основании суперпозиции двух более простых программ, приведем программу построения обратного кода<sup>8)</sup> двоичного числа  $n$ , заданного на ленте машины в виде последовательности из  $n$  "палочек", то есть в унарной системе счисления.

Программу машины  $M_1$  мы возьмем готовую из **примера 2** лекции 17, которая решает задачу перевода чисел из унарной си-

<sup>6)</sup> Конечно, не следует забывать о внешнем и внутреннем алфавитах машины, о начальной конфигурации. Но все же программа - это основной элемент в задании машины Тьюринга.

<sup>7)</sup> Этого всегда можно добиться, введя во внешний алфавит каждой машины недостающие символы из внешнего алфавита другой, то есть взять в качестве общего внешнего алфавита объединение алфавитов машин  $M_1$  и  $M_2$ .

<sup>8)</sup> Обратным кодом двоичного числа  $\sigma_1\sigma_2\dots\sigma_p$  является двоичное число, полученное из данного числа заменой всех 1 на 0, а 0 - на 1.

стемы счисления в двоичную. Напомним, что ее начальное состояние  $q_1$ , заключительное состояние  $q_0$ , внутренний алфавит  $Q_I = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6\}$ , внешний алфавит  $A_1 = \{\Lambda, |, 0, 1\}$ .

Построим программу машины  $M_2$ , обрабатывающей двоичное число.

Внешний алфавит машины  $M_2$  -  $A_2 = \{\Lambda, 0, 1\}$

В качестве начальной конфигурации для машины  $M_2$  возьмем:  $q_1\sigma_1\sigma_2\dots\sigma_p$ , где  $\sigma_i$  - двоичные цифры.

Программа искомой машины будет иметь вид:

$$\begin{aligned} q_1 0 &\rightarrow q_1 0 R & q_1 \Lambda &\rightarrow q_2 \Lambda L & q_2 1 &\rightarrow q_2 0 L \\ q_1 1 &\rightarrow q_1 1 R & q_2 0 &\rightarrow q_2 1 L & q_2 \Lambda &\rightarrow q_0 \Lambda R \end{aligned}$$

Переобозначим состояния машины  $M_2$  следующим образом:  
 $q_1 \rightarrow q_7, q_2 \rightarrow q_8$ .

После этих преобразований программа машины  $M_2$  будет иметь вид:

$$\begin{aligned} q_7 0 &\rightarrow q_7 0 R & q_7 \Lambda &\rightarrow q_8 \Lambda L & q_8 1 &\rightarrow q_8 0 L \\ q_7 1 &\rightarrow q_7 1 R & q_8 0 &\rightarrow q_8 1 L & q_8 \Lambda &\rightarrow q_0 \Lambda R \end{aligned}$$

(Внутренний алфавит машины  $M_2$ , таким образом, будет:  $Q_2 = \{q_0, q_7, q_8\}$ .)

Теперь заключительное состояние  $q_0$  машины  $M_2$  отождествим с состоянием  $q_7$ . Соединив программы двух машин в одну, получим искомую:

$$\begin{aligned} q_1 | &\rightarrow q_1 | L & q_4 \Lambda &\rightarrow q_5 1 R & q_7 0 &\rightarrow q_7 0 R \\ q_1 \Lambda &\rightarrow q_2 0 H & q_5 1 &\rightarrow q_5 1 R & q_7 1 &\rightarrow q_7 1 R \\ q_2 | &\rightarrow q_2 | R & q_5 0 &\rightarrow q_5 0 R & q_7 \Lambda &\rightarrow q_8 \Lambda L \\ q_2 \Lambda &\rightarrow q_3 \Lambda L & q_5 | &\rightarrow q_2 | H & q_8 0 &\rightarrow q_8 1 L \\ q_3 | &\rightarrow q_4 \Lambda L & q_5 \Lambda &\rightarrow q_6 \Lambda L & q_8 1 &\rightarrow q_8 0 L \\ q_4 | &\rightarrow q_4 | L & q_6 1 &\rightarrow q_6 1 L & q_8 \Lambda &\rightarrow q_0 \Lambda R \\ q_4 0 &\rightarrow q_5 1 R & q_6 0 &\rightarrow q_6 0 L & & \\ q_4 1 &\rightarrow q_4 0 L & q_6 \Lambda &\rightarrow q_7 \Lambda R & & \end{aligned}$$



Итак, машина **М** имеет:

- внешний алфавит  $A = A_1 \cup A_2 = \{\Lambda, |, 0, 1\} \cup \{\Lambda, 0, 1\} = \{\Lambda, |, 0, 1\}$ ;
- внутренний алфавит  $Q = Q_1 \cup Q_2 = \{q_0, q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8\}$ ;
- начальное состояние  $q_1$ ;
- заключительное состояние  $q_0$ ;
- начальная конфигурация -  $q_1|||...|$ ;
- заключительная конфигурация:  $q_0\sigma_1\sigma_2...\sigma_p$ .

Аналогично строится суперпозиция трех, четырех и т.д. машин Тьюринга.

## 2. Композиция машин (программ)

Пусть машины Тьюринга **М**<sub>1</sub> и **М**<sub>2</sub> вычисляют соответственно словарные функции  $f_1(P)$  и  $f_2(P)$ . Тогда можно построить машину **М**, которая выполняет над словом **Р** работу обеих машин, а именно: слово **Р** она перерабатывает в слово  $f_1(P)*f_2(P)$ , где  $*$  – символ, не встречающийся в алфавитах машин **М**<sub>1</sub> и **М**<sub>2</sub>. При этом, если хотя бы одно из значений  $f_1(P)$  или  $f_2(P)$  не определено, то машина **М** на слове **Р** будет работать бесконечно. Машину **М** будем называть композицией машин **М**<sub>1</sub> и **М**<sub>2</sub> и обозначать **М**<sub>1</sub>\***М**<sub>2</sub>.

При построении машины **М** используется прием, который является довольно распространенным. Иногда удобно считать, что машина имеет "двухэтажную" ленту. Это означает, что в качестве символов внешнего алфавита используются пары вида:  $\left(\frac{b}{a}\right)$ .

Переработка информации на каждом этаже ленты может осуществляться "независимо". Если, например, мы хотим, чтобы на нижнем этаже выполнялась команда  $q_i a_j \rightarrow q_l a_p D$ , независимо от того, что записано на верхнем этаже, то в программу машины должны быть введены команды  $q_j \left(\frac{b}{a_i}\right) \rightarrow q_l \left(\frac{b}{a_p}\right) D$  для всех **b**, принадлежащих алфавиту верхнего этажа. (Аналогично, если необходимо работать только на верхнем этаже.)

Машина **М** начинает работу на обычной "одноэтажной" ленте, на промежуточном этапе использует "двухэтажную", а в конце снова возвращается к "одноэтажной" ленте.

Записанное на "одноэтажной" ленте исходное слово  $P$  машина записывает на "двухэтажной" ленте: каждая "одноэтажная" буква  $a_i$  алфавита машины  $M_1$  или  $M_2$  заменяется «двухэтажным» символом  $\left(\frac{\Lambda}{a_i}\right)$ . Таким образом, слово  $P$  записывается на нижнем этаже. Затем оно переписывается также на верхний этаж, то есть делается его "копия" на верхнем этаже (для этого каждая непустая буква  $\left(\frac{\Lambda}{a_i}\right)$  заменяется на букву  $\left(\frac{a_i}{a_i}\right)$ ).

Далее машина работает на нижнем этаже как  $M_1$  до тех пор, пока не вычислит  $f_1(P)$ . После чего, работая на верхнем этаже как  $M_2$ , вычисляет там  $f_2(P)$ . Затем к слову  $f_1(P)$  на нижнем этаже она добавляет символ  $*$  и побуквенно дописывает слово  $f_2(P)$ , находящееся на верхнем этаже (переписанные буквы слова  $f_2(P)$  на верхнем этаже стираются).

Наконец, на последнем этапе машина избавляется от верхнего этажа, заменяя «двухэтажные» буквы  $\left(\frac{\Lambda}{a_i}\right)$  на «одноэтажные»  $a_i$ .

Более подробно на реализации этой идеи мы останавливаться не будем. Предлагаем читателю самому восстановить недостающие детали "конструкции" машины  $M$ .

***Суперпозиция и композиция программ позволяют реализовать на машине Тьюринга любое линейное или параллельное вычисление.***

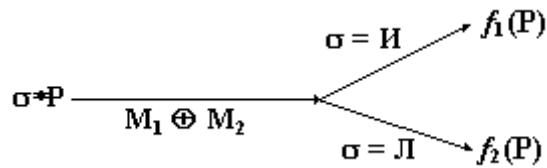
## Лекция 18. Ветвление программ и циклы для машины Тьюринга

Учебные вопросы:

1. Ветвление машин (программ).
2. Циклические вычисления на машинах Тьюринга.

### 1. Ветвление машин (программ)

Пусть имеются машины Тьюринга  $M_1$  и  $M_2$ , которые вычисляют словарные функции  $f_1(P)$  и  $f_2(P)$  в одном и том же внешнем алфавите  $A$ , и пусть символы **И** и **Л** ("истина" и "ложь") не входят в этот алфавит. Требуется построить машину  $M$ , которая слово  $\sigma * P$  (где  $\sigma \in \{\text{И}, \text{Л}\}$ ,  $* \notin A$ ) переводит в слово  $f_1(P)$ , если  $\sigma = \text{И}$ , и в слово  $f_2(P)$ , если  $\sigma = \text{Л}$ . Машину  $M$  будем обозначать  $M_1 \oplus M_2$  и изображать в виде:



Машина  $M$  может быть сконструирована следующим образом.

Обозначим через  $q_1^{(1)}$  и  $q_1^{(2)}$  начальные состояния машин  $M_1$  и  $M_2$  соответственно (считаем, что состояния машин  $M_1$  и  $M_2$  обозначены разными буквами). Программы машин  $M_1$  и  $M_2$  допишем друг к другу, введем новое начальное состояние  $q_1$  и дополнительные команды:

$$\begin{aligned} q_1 \text{И} &\rightarrow q_1^{(1)} \Lambda R \\ q_1^{(1)} * &\rightarrow q_1^{(1)} \Lambda R \\ q_1 \text{Л} &\rightarrow q_1^{(2)} \Lambda R \\ q_1^{(2)} * &\rightarrow q_1^{(2)} \Lambda R. \end{aligned}$$

Кроме того, заключительные состояния  $q_0^{(1)}$  и  $q_0^{(2)}$  машин  $M_1$  и  $M_2$  обозначим одной буквой  $q_0$  и состояние  $q_0$  будем считать заключительным для машины  $M$ .

Пусть на ленте записано слово  $\mathbf{I}*\mathbf{P}$  и машина  $\mathbf{M}$  запущена в начальном состоянии  $\mathbf{q}_1$ . Тогда головка сотрет символ  $\mathbf{I}$ , перейдет в состояние  $\mathbf{q}_1^{(1)}$  и переместится вправо. Теперь она будет в состоянии  $\mathbf{q}_1^{(1)}$  обозревать символ  $*$ . По команде  $\mathbf{q}_1^{(1)} * \rightarrow \mathbf{q}_1^{(1)} \Delta \mathbf{R}$  машина сотрет символ  $*$  и переместится в том же состоянии к первой букве слова  $\mathbf{P}$ . Поскольку состояние  $\mathbf{q}_1^{(1)}$  является начальным для машины  $\mathbf{M}_1$ , то машина  $\mathbf{M}$  будет дальше работать как машина  $\mathbf{M}_1$  и вычислит значение  $f_1(\mathbf{P})$ .

Случай  $\sigma = \mathbf{L}$  рассматривается аналогично.

Таким образом, мы можем реализовывать разветвляющиеся вычислительные процессы на машинах Тьюринга.

## 2. Циклические вычисления на машинах Тьюринга

Весьма часто процесс вычисления содержит циклические (повторяющиеся) участки, или циклы. После осуществления действий, составляющих повторяющийся участок (*тело* цикла), проверяется выполнимость некоторого условия. В случае утвердительного ответа выдается результат, в противном случае цикл повторяется.

Более точно, пусть имеется некоторое свойство  $\Phi$  слов в алфавите  $A$  и словарные функции  $f_1$  и  $f_2$ . Проверяется, обладает ли исходное слово  $\mathbf{P}$  свойством  $\Phi$ , и если обладает, то выдается ответ  $f_1(\mathbf{P})$ . В противном случае вычисляется слово  $\mathbf{P}' = f_2(\mathbf{P})$  и проверяется, обладает ли слово  $\mathbf{P}'$  свойством  $\Phi$ . Если обладает, то выдается ответ  $f_1(\mathbf{P}')$ , если нет – вычисляется слово  $\mathbf{P}'' = f_2(\mathbf{P}')$ , и т.д.

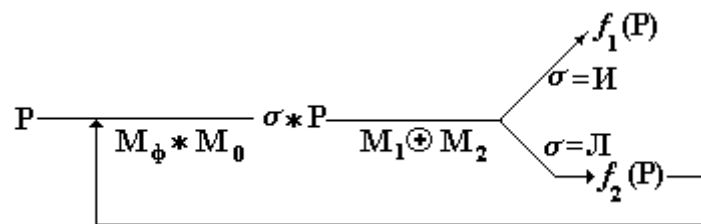
Указанная процедура может быть реализована на машинах Тьюринга. Пусть свойство  $\Phi$  проверяется машиной  $\mathbf{M}_\Phi$ , а функции  $f_1$  и  $f_2$  вычисляются машинами  $\mathbf{M}_1$  и  $\mathbf{M}_2$ . Обозначим через  $\mathbf{M}_0$  машину, которая всякое входное слово  $\mathbf{P}$  оставляет без изменения.

Машина  $\mathbf{M}$ , осуществляющая цикл, может быть построена так:

Вначале к слову  $\mathbf{P}$  применяется композиция машин  $\mathbf{M}_\Phi$  и  $\mathbf{M}_0$ , то есть машина  $\mathbf{M}_\Phi * \mathbf{M}_0$ , которая переводит его в слово  $\sigma * \mathbf{P}$ , где  $\sigma = \mathbf{I}$ , если  $\mathbf{P}$  обладает свойством  $\Phi$ , и  $\sigma = \mathbf{L}$  в противном случае.

Затем запускается машина  $\mathbf{M}_1 \oplus \mathbf{M}_2$  (заключительное состояние машины  $\mathbf{M}_\Phi * \mathbf{M}_0$  объявляется начальным состоянием машины  $\mathbf{M}_1 \oplus \mathbf{M}_2$ ). При этом, в отличие от описания ветвления машин, за-

ключительные состояния  $q_0^{(1)}$  и  $q_0^{(2)}$  машин  $M_1$  и  $M_2$  не объединяются в одно, а считаются разными. Состояние  $q_0^{(1)}$  объявляется заключительным состоянием машины  $M$ , а  $q_0^{(2)}$  отождествляется с начальным состоянием. В результате, если машина  $M_1 \oplus M_2$  работала как  $M_1$ , то полученное значение является выходным, если же она работала как  $M_2$ , то полученное значение снова подается на вход машины  $M$ . Схематично работу машины  $M$  можно представить следующим образом:



Приведенные способы сочетания машин (программ) существенно облегчают процесс программирования. Они позволяют разбить задачу на подзадачи, а затем устраивать последовательное соединение программ для отдельных подзадач (суперпозицию), параллельное соединение (композицию), использовать связки типа «если  $\Phi$ , то выполняй  $f_1$ , иначе выполняй  $f_2$ », «пока  $\Phi$ , повторяй  $f_2$ , иначе  $f_1$ ». Таким образом, язык программирования для может быть сделан достаточно богатым. Это служит дополнительным доводом в пользу тезиса, что всякий интуитивно алгоритмический процесс может быть реализован на машине Тьюринга.

**Пример 5.** Построить машину Тьюринга, складывающую двоичные числа  $m$  и  $n$  ( $n \geq 1$ ).

Будем считать, что числа  $m$  и  $n$  представлены на ленте в следующем виде:

$\alpha^{(1)}_1 \alpha^{(1)}_2 \dots \alpha^{(1)}_p * \alpha^{(2)}_1 \alpha^{(2)}_2 \dots \alpha^{(2)}_s$ , где  $\alpha^{(1)}_i, \alpha^{(2)}_j$  – двоичные цифры чисел  $m$  и  $n$  соответственно,  $*$  – разделитель.

Кроме того, слева и справа слово  $\alpha^{(1)}_1 \alpha^{(1)}_2 \dots \alpha^{(1)}_p * \alpha^{(2)}_1 \alpha^{(2)}_2 \dots \alpha^{(2)}_s$  ограничено пустыми символами ( $\Lambda$ ).

Неформально работу машины Тьюринга, решающей указанную задачу, можно описать следующим образом. Головка машины должна перемещаться по ленте справа налево и слева направо («качаться, как маятник»): вычитать по 1 из числа  $n$  и прибавлять по 1 к

числу  $m$  до тех пор, пока число  $n$  не исчерпается (то есть, пока  $n$  не станет равно 0).

Разобьем нашу задачу на подзадачи и построим следующие машины Тьюринга, которые будут решать эти подзадачи:

- 1) машину  $M_1$ , проверяющую условие " $n=0$ ?", и ставящую перед записью  $m*n$  символ  $I$ , если  $n=0$ , и символ  $L$ , в противном случае;
- 2) машину  $M_2$ , вычитающую 1 из числа  $n$ ;
- 3) машину  $M_3$ , прибавляющую 1 к числу  $m$ ;
- 4) машину  $M_4$ , завершающую процесс вычисления.

Построим указанные машины в предположении, что они полностью независимы, то есть всякий раз будем указывать все составляющие машины Тьюринга: внешний алфавит, внутренний алфавит, начальную конфигурацию, заключительную конфигурацию, программу. Затем «соберем» машину  $M$ , решающую поставленную задачу сложения двоичных чисел.

#### Машина $M_1$ .

Внешний алфавит  $A_1=\{0, 1, *, I, L, \Lambda\}$ . Внутренний алфавит  $Q_1=\{q_0, q_1, q_2, q_3, q_4\}$ .

Начальная конфигурация:  $q_1\alpha^{(1)}_1\alpha^{(1)}_2\ldots\alpha^{(1)}_p*\alpha^{(2)}_1\alpha^{(2)}_2\ldots\alpha^{(2)}_s$ , где  $\alpha^{(1)}_i, \alpha^{(2)}_j$  – двоичные цифры (чисел  $m$  и  $n$  соответственно);

Заключительная конфигурация:  $q_0\sigma\alpha^{(1)}_1\alpha^{(1)}_2\ldots\alpha^{(1)}_p*\alpha^{(2)}_1\alpha^{(2)}_2\ldots\alpha^{(2)}_s$ , где  $\sigma\in\{I, L\}$ ,  $\alpha^{(1)}_i, \alpha^{(2)}_j$  – двоичные цифры (чисел  $m$  и  $n$  соответственно).

Программа:

|                                      |                         |                               |
|--------------------------------------|-------------------------|-------------------------------|
| $q_11\rightarrow q_11R$              | $q_21\rightarrow q_31L$ | $q_3\Lambda\rightarrow q_0II$ |
| $q_10\rightarrow q_10R$              | $q_2*\rightarrow q_4*L$ | $q_40\rightarrow q_40L$       |
| $q_1*\rightarrow q_1*R$              | $q_30\rightarrow q_30L$ | $q_41\rightarrow q_41L$       |
| $q_1\Lambda\rightarrow q_2\Lambda L$ | $q_31\rightarrow q_31L$ | $q_4\Lambda\rightarrow q_0LH$ |
| $q_20\rightarrow q_20L$              | $q_3*\rightarrow q_3*L$ |                               |

#### Машина $M_2$ .

Внешний алфавит  $A_2=\{0, 1, *, \Lambda\}$ . Внутренний алфавит  $Q_2=\{q_0, q_1, q_2, q_3\}$ .

Начальная конфигурация:  $q_1 \alpha^{(1)}_1 \alpha^{(1)}_2 \dots \alpha^{(1)}_p * \alpha^{(2)}_1 \alpha^{(2)}_2 \dots \alpha^{(2)}_s$ , где  $\alpha^{(1)}_i, \alpha^{(2)}_j$  – двоичные цифры (чисел  $m$  и  $n$  соответственно);

Заключительная конфигурация:  $q_0 \alpha^{(1)}_1 \alpha^{(1)}_2 \dots \alpha^{(1)}_p * \alpha^{(2)}_1 \alpha^{(2)}_2 \dots \alpha^{(2)}_s$ , где  $\alpha^{(1)}_i, \alpha^{(2)}_j$  – двоичные цифры (чисел  $m$  и  $n-1$  соответственно).

Программа:

$$\begin{array}{lll} q_1 1 \rightarrow q_1 1R & q_2 1 \rightarrow q_3 0L & q_3 * \rightarrow q_3 *L \\ q_1 0 \rightarrow q_1 0R & q_2 0 \rightarrow q_2 1L & q_3 \Lambda \rightarrow q_0 \Lambda R \\ q_1 * \rightarrow q_1 *R & q_3 0 \rightarrow q_3 0L & \\ q_1 \Lambda \rightarrow q_2 \Lambda L & q_3 1 \rightarrow q_3 1L & \end{array}$$

### Машина $M_3$

Внешний алфавит  $A_3 = \{0, 1, *, \Lambda\}$ .

Внутренний алфавит  $Q_3 = \{q_0, q_1, q_2, q_3\}$ .

Начальная конфигурация:  $q_1 \alpha^{(1)}_1 \alpha^{(1)}_2 \dots \alpha^{(1)}_p * \alpha^{(2)}_1 \alpha^{(2)}_2 \dots \alpha^{(2)}_s$ , где  $\alpha^{(1)}_i, \alpha^{(2)}_j$  – двоичные цифры (чисел  $m$  и  $n$  соответственно).

Заключительная конфигурация:  $q_0 \alpha^{(1)}_1 \alpha^{(1)}_2 \dots \alpha^{(1)}_k * \alpha^{(2)}_1 \alpha^{(2)}_2 \dots \alpha^{(2)}_s$ , где  $k \geq p, \alpha^{(1)}_i, \alpha^{(2)}_j$  – двоичные цифры (чисел  $m+1$  и  $n$  соответственно).

Программа:

$$\begin{array}{lll} q_1 1 \rightarrow & q_2 0 \rightarrow & q_3 0 \rightarrow \\ q_1 1R & q_3 1L & q_3 0L \\ q_1 0 \rightarrow & q_2 1 \rightarrow & q_3 1 \rightarrow \\ q_1 0R & q_2 0L & q_3 1L \\ q_1 * \rightarrow & q_2 \Lambda \rightarrow & q_3 \Lambda \rightarrow \\ q_2 *L & q_0 1H & q_0 \Lambda R \end{array}$$

### Машина $M_4$

Внешний алфавит  $A_4 = \{0, 1, *, \Lambda\}$ .

Внутренний алфавит  $Q_4 = \{q_0, q_1, q_2, q_3, q_4\}$ .

Начальная конфигурация:  $q_1 \alpha^{(1)}_1 \alpha^{(1)}_2 \dots \alpha^{(1)}_k * 00 \dots 0_s$ , где  $k \geq p, \alpha^{(1)}_i$  – двоичные цифры числа  $m+n$ .

Заключительная конфигурация:  $q_0 \alpha^{(1)}_1 \alpha^{(1)}_2 \dots \alpha^{(1)}_k$ , где  $k \geq p, \alpha^{(1)}_i$  – двоичные цифры числа  $m+n$ .

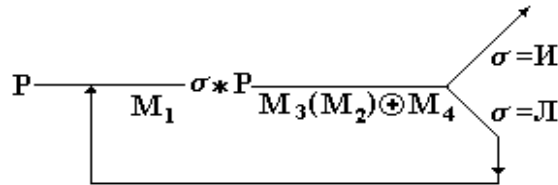
Программа:

$$\begin{array}{lll}
 q_1 1 \rightarrow q_1 1R & q_2 \Lambda \rightarrow q_3 \Lambda L & q_4 0 \rightarrow q_4 0L \\
 q_1 0 \rightarrow q_1 0R & q_3 0 \rightarrow q_3 \Lambda L & q_4 \Lambda \rightarrow q_0 \Lambda R \\
 q_1 * \rightarrow q_2 *R & q_3 * \rightarrow q_4 \Lambda L & \\
 q_2 0 \rightarrow q_2 0R & q_4 1 \rightarrow q_4 1L & 
 \end{array}$$

Построим теперь машину  $M$  как сочетание машин  $M_1 - M_4$ . Опишем неформально это сочетание.

Вначале работает машина  $M_1$  и ставит перед словом  $m*n$  символ  $\sigma \in \{И, Л\}$ , а именно: символ  $И$ , если  $n=0$ ; или символ  $Л$ , если  $n \neq 0$ .

Затем в зависимости от значения  $\sigma$  работает либо машина  $M_4$  (если  $\sigma=И$ ), либо машина  $M_3(M_2)$  (если  $\sigma=Л$ ). Последняя машина является суперпозицией машин  $M_3$  и  $M_2$ , обеспечивающей вычитание 1 из числа  $n$  и прибавление 1 к числу  $m$ . Если работает машина  $M_4$ , то работа машины  $M$  завершается; если же работает машина  $M_3(M_2)$ , то вновь запускается на выполнение машина  $M_1$  и цикл повторяется.



Графическая схема, иллюстрирующая работу машины  $M$ , имеет вид:

Переобозначим состояния машин  $M_1 - M_4$  следующим образом:

$$\begin{aligned}
 Q_1 &= \{q'_{,0}, q'_{,1}, q'_{,2}, q'_{,3}, q'_{,4}\}; \\
 Q_2 &= \{q_{,,0}, q_{,,1}, q_{,,2}, q_{,,3}\}; \\
 Q_3 &= \{q_{,,,0}, q_{,,,1}, q_{,,,2}, q_{,,,3}\}; \\
 Q_4 &= \{q_{,,,,0}, q_{,,,,1}, q_{,,,,2}, q_{,,,,3}, q_{,,,,4}\}.
 \end{aligned}$$

Определим теперь суперпозицию машин  $M_3(M_2)$ . Для этого отождествим заключительное состояние  $q''_0$  машины  $M_2$  с началь-



ным состоянием  $q''_1$  машины  $M_3$  и обозначим их через  $q_2$ . Начальное состояние суперпозиции  $M_3(M_2)$  -  $q''_1$ , заключительное состояние -  $q''_0$ .

Состояние  $q'_0$  машины  $M_1$  объявим незаключительным и добавим к ее программе следующие команды:

$$\begin{aligned} q'_0 I &\rightarrow q''_1 \Lambda R \\ q'_0 L &\rightarrow q''_1 \Lambda R \end{aligned}$$

Заключительное состояние  $q''_0$  машины  $M_3(M_2)$  отождествим с начальным состоянием  $q'_1$  машины  $M_1$  и обозначим их через  $q_1$ .

Заключительное состояние  $q''_0$  машины  $M_4$  объявим заключительным состоянием машины  $M$  и обозначим его через  $q_0$ .

Итак, машина  $M$  имеет:

внешний алфавит  $A = \{0, 1, *, I, L, \Lambda\}$ ;

внутренний алфавит  $Q = \{q_0, q_1, q_2, q'_0, q'_2, q'_3, q'_4, q''_1, q''_2, q''_3, q''_4, q''_1, q''_2, q''_3, q''_4\}$ ;

начальное состояние -  $q_1$ ; заключительное состояние -  $q_0$ ;

начальная конфигурация:  $q_1 \alpha^{(1)}_1 \alpha^{(1)}_2 \dots \alpha^{(1)}_p * \alpha^{(2)}_1 \alpha^{(2)}_2 \dots \alpha^{(2)}_s$ , где  $\alpha^{(1)}_i, \alpha^{(2)}_j$

– двоичные цифры (чисел  $m$  и  $n$  соответственно);

заключительная конфигурация:  $q_0 \alpha^{(1)}_1 \alpha^{(1)}_2 \dots \alpha^{(1)}_k$ , где  $k \geq p$ ,  $\alpha^{(1)}_i$  – двоичные цифры числа  $m+n$ ;

программа:

|   |                                      |   |  |   |
|---|--------------------------------------|---|--|---|
| $q_1 1 \rightarrow q_1 1R$                | $q'_3 * \rightarrow q'_3 *L$         | $q''_1 * \rightarrow q''_1 *R$              | $q_2 0 \rightarrow q_2 0R$                 | $q''''_1 0 \rightarrow q''''_1 0R$              |
| $q_1 0 \rightarrow q_1 0R$                | $q'_3 \Lambda \rightarrow q'_0 IH$   | $q''_1 \Lambda \rightarrow q''_2 \Lambda L$ | $q_2 * \rightarrow q''_2 *L$               | $q''''_1 * \rightarrow q''''_2 *R$              |
| $q_1 * \rightarrow q'_1 *R$               | $q'_4 0 \rightarrow q'_4 0L$         | $q''_2 1 \rightarrow q''_3 0L$              | $q''_2 0 \rightarrow q''_3 1L$             | $q''''_2 0 \rightarrow q''''_2 0R$              |
| $q'_1 \Lambda \rightarrow q'_2 \Lambda L$ | $q'_4 1 \rightarrow q'_4 1L$         | $q''_2 0 \rightarrow q''_2 1L$              | $q''_2 1 \rightarrow q''_2 0L$             | $q''''_2 \Lambda \rightarrow q''''_3 \Lambda L$ |
| $q'_2 0 \rightarrow q'_2 0L$              | $q'_4 \Lambda \rightarrow q'_0 LH$   | $q''_3 0 \rightarrow q''_3 0L$              | $q''_2 \Lambda \rightarrow q'_1 IH$        | $q''''_3 0 \rightarrow q''''_3 \Lambda L$       |
| $q'_2 1 \rightarrow q'_3 1L$              | $q'_0 I \rightarrow q''_1$           | $q''_3 1 \rightarrow q''_3 1L$              | $q''_3 0 \rightarrow q''_3 0L$             | $q''''_3 * \rightarrow q''''_4 \Lambda L$       |
| $q'_2 * \rightarrow q'_4 *L$              | $\Lambda R$                          | $q''_3 * \rightarrow q''_3 *L$              | $q''_3 1 \rightarrow q''_3 1L$             | $q''''_4 1 \rightarrow q'_4 1L$                 |
| $q'_3 0 \rightarrow q'_3 0L$              | $q'_0 L \rightarrow q''_1 \Lambda R$ | $q''_3 \Lambda \rightarrow q_2 \Lambda R$   | $q''_3 \Lambda \rightarrow q'_1 \Lambda R$ | $q''''_4 0 \rightarrow q''''_4 0L$              |
| $q'_3 1 \rightarrow q'_3 1L$              | $q''_1 1 \rightarrow q''_1 1R$       | $q_2 1 \rightarrow q_2 1R$                  | $q''_1 1 \rightarrow q''_1 1R$             | $q''''_4 \Lambda \rightarrow q_0 \Lambda R$     |
|   | $q''_1 0 \rightarrow q''_1 0R$       |   |  |   |

Построенная методом «сборки» программа машины  $M$  является далеко не оптимальной. Можно составить более короткую и с меньшим числом состояний программу машины Тьюринга для

сложения двоичных чисел.<sup>9)</sup> Здесь показана возможность разложения задачи на подзадачи, составления программ машин Тьюринга для решения этих подзадач и соединения («сборки») этих программ в одну.

---

<sup>9)</sup> Например:

|                                       |                                       |                                 |
|---------------------------------------|---------------------------------------|---------------------------------|
| $q_1 1 \rightarrow q_1 1R$            | $q_4 1 \rightarrow q_4 \Delta R$      | $q_3 0 \rightarrow q_3 0L$      |
| $q_1 0 \rightarrow q_1 0R$            | $q_4 \Delta \rightarrow q_5 \Delta H$ | $q_3 1 \rightarrow q_3 1L$      |
| $q_1 * \rightarrow q_1 *R$            | $q_5 \Delta \rightarrow q_5 \Delta L$ | $q_3 * \rightarrow q_7 *L$      |
| $q_1 \Delta \rightarrow q_2 \Delta L$ | $q_5 * \rightarrow q_6 \Delta L$      | $q_7 0 \rightarrow q_1 1H$      |
| $q_2 0 \rightarrow q_2 1L$            | $q_6 0 \rightarrow q_6 0L$            | $q_7 1 \rightarrow q_7 0L$      |
| $q_2 1 \rightarrow q_3 0L$            | $q_6 1 \rightarrow q_6 1L$            | $q_7 \Delta \rightarrow q_1 1H$ |
| $q_2 * \rightarrow q_4 *R$            | $q_6 \Delta \rightarrow q_0 \Delta R$ |                                 |

## Лекция 19. Рекурсивные функции

### Учебные вопросы:

#### 1. Вычислимые функции. Частично рекурсивные и общерекурсивные функции.

Уточнением понятия эффективно вычислимой функции занимались А. Черч, К. Гедель, С. Клини. В результате был выделен класс так называемых частично-рекурсивных функций, имеющих строгое математическое определение.

Для алгоритмических проблем типичным является то обстоятельство, что требуется найти алгоритм для решения задачи, в условия которой входят значения некоторой конечной системы целочисленных параметров  $x_1, x_2, \dots, x_n$ , а искомым результатом также является целое число  $y$ . Следовательно, стоит вопрос о существовании алгоритма для вычисления значений числовой функции  $y$ , зависящей от целочисленных значений аргументов  $x_1, x_2, \dots, x_n$ .

**Определение 1.** Функция  $g = f(x_1, x_2, \dots, x_n)$  называется эффективно вычислимой, если существует алгоритм, позволяющий вычислить ее значения.

Так как в этом определении алгоритм понимается в интуитивном смысле, то и понятие эффективно вычислимой функции является интуитивным.

Однако переход от алгоритма к эффективно вычислимой функции дает определенные преимущества. Дело в том, что те требования, которые предъявляются к алгоритму в его характерных чертах, выполняются для совокупности всех вычислимых функций, которая носит название совокупности рекурсивных функций.

Гёдель впервые описал класс всех рекурсивных функций как класс всех числовых функций, определяемых в некоторой формальной системе. Черч в 1936 году пришел к тому же классу функций, исходя из других предпосылок. Здесь построение класса вычислимых функций строится следующим образом.

Выбираются простейшие функции

$$L(x) = x + 1 \text{ (оператор сдвига),}$$

$$O(x) = 0 \text{ (оператор аннулирования),}$$

$I_n^m \{ x_1, x_2, \dots, x_n \} = x_m, 1 \leq m \leq n$  (оператор проектирования).

Ясно, что все три простейшие функции всюду определены и интуитивно вычислимы.

Далее вводятся операции над функциями.

### Суперпозиция функций

Рассмотрим функции  $f_1(x_1, x_2, \dots, x_n)$ ,  $f_2(x_1, x_2, \dots, x_n), \dots, f_m(x_1, x_2, \dots, x_n)$ ,  $\varphi(y_1, y_2, \dots, y_m)$  и функцию  $\psi(x_1, x_2, \dots, x_n)$ , определяемую равенством

$$\psi(x_1, x_2, \dots, x_n) = \varphi(f_1(x_1, x_2, \dots, x_n), f_2(x_1, x_2, \dots, x_n), \dots, f_m(x_1, x_2, \dots, x_n))$$

Будем говорить, что функция  $\psi$  получена из функций  $\varphi$  и  $f_1, f_2, \dots, f_m$  суперпозицией.

Если мы каким-либо образом умеем вычислять функции  $f_1, f_2, \dots, f_m$  и  $\varphi$ , то функция  $\psi$  может быть вычислена так: придадим переменным  $x_1, x_2, \dots, x_n$  некоторые значения  $a_1, a_2, \dots, a_n$ .

Вычисляя все  $f_i(a_1, a_2, \dots, a_n)$ , найдём  $b_i = f_i(a_1, a_2, \dots, a_n)$ . Вычисляя теперь  $\varphi(b_1, b_2, \dots, b_m)$  найдём  $c = \psi(a_1, a_2, \dots, a_n)$ .

Ясно, что если все функции  $f_1, f_2, \dots, f_m$  и  $\varphi$  всюду определены, то функция  $\psi$  всюду определена. Функция  $\psi$  будет не всюду определенной, если хотя бы одна из функций  $f_1, f_2, \dots, f_m$  не всюду определена, или если можно найти такие значения аргументов  $a_1, a_2, \dots, a_n$ , что

$$b_i = f_i(a_1, a_2, \dots, a_n), \text{ но } \varphi(b_1, b_2, \dots, b_m) \text{ не определено } (i=1, 2, \dots, m).$$

Таким образом, если функции  $f_1, f_2, \dots, f_m$ ,  $\varphi$  интуитивно вычислимы, то будет интуитивно вычислима и функция  $\psi$ .

Отметим, что возможны случаи, когда не все функции  $f_1, f_2, \dots, f_m$  зависят от всех  $n$  аргументов  $x_1, x_2, \dots, x_n$ . В этих случаях для получения суперпозиции используются фиктивные аргументы и функции  $I_n^m \{ x_1, x_2, \dots, x_n \} = x_m$ .

Например, функция  $\psi(x,y,z) = \varphi(f_1(x), f_2(x,y,z), y, x)$  получается суперпозицией из функций  $\varphi(y_1, y_2, y_3, y_4)$  и  $F_1(x,y,z) = f_1(x)$ ,  $F_2(x,y,z) = f_2(x,y,z)$ ,  $F_3(x,y,z) = I_3^2(x,y,z)$ ,  $F_4(x,y,z) = I_3^1(x,y,z)$ .

## Схема примитивной рекурсии

Пусть имеется две функции  $\varphi(x_1, x_2, \dots, x_n)$  и  $\psi(x_1, x_2, \dots, x_n, x_{n+1})$  ( $n > 1$ ). Рассмотрим новую функцию, которая удовлетворяет следующим равенствам:

$$\begin{aligned} f(0, x_2, x_3, \dots, x_n) &= \varphi(x_2, \dots, x_n), \\ f_1(y+1, x_2, x_3, \dots, x_n) &= \psi(y, f_1(y, x_2, \dots, x_n), x_2, \dots, x_n). \end{aligned} \quad (1)$$

Отметим, что функция  $\varphi$  зависит от  $n-1$  аргументов, функция  $\psi$  от  $n+1$  аргументов, а функция  $f$  - от  $n$  аргументов.

Если функция  $f_1(x_1, x_2, \dots, x_n)$  получается из функций  $\varphi, \psi$  с помощью равенств (1), то говорят, что функция  $f$  получена из функций  $\varphi, \psi$  по схеме примитивной рекурсии.

Если функции  $\varphi, \psi$  и  $V$  интуитивно вычислимы, то будет интуитивно вычислима и функция  $f$ . Действительно, пусть  $a_1, a_2, \dots, a_n$  - набор значений аргументов  $x_1, x_2, \dots, x_n$ . Тогда последовательно находим

$$\begin{aligned} f_1(0, a_2, \dots, a_n) &= \varphi(a_2, a_3, \dots, a_n) \\ f_1(1, a_2, \dots, a_n) &= \psi(0, b_0, a_2, a_3, \dots, a_n) = b_1 \\ f_1(2, a_2, \dots, a_n) &= \psi(1, b_1, a_2, a_3, \dots, a_n) = b_2 \text{ и т.д.} \end{aligned}$$

Очевидно, что если функции  $\varphi$  и  $\psi$  всюду определены, то будет всюду определена и функция  $f$ .

Рассмотрим примеры получения функций по схеме примитивной рекурсии.

**Пример 1.** Пусть функция  $f(y, x)$  задана равенствами:

$$\begin{aligned} f(0, x) &= x, \\ f(y+1, x) &= f(y, x) + 1. \end{aligned}$$

Здесь функция  $\varphi = x$ , а  $\psi\{x, y, z\} = y + 1$ .

Вычислим значение функции  $f(y, x)$  при  $y = 5, x = 2$ .

Т.к.  $f(0, 2) = \varphi(2) = 2$ , то из второго равенства последовательно имеем:

$$\begin{aligned} f(1, 2) &= \psi(0, 2, 2) = 2 + 1 = 3 \\ f(2, 2) &= \psi(1, 3, 2) = 3 + 1 = 4 \end{aligned}$$

$$f(3,2) = \psi(2,4,2) = 4+1=5$$

$$f(4,2) = \psi(3,5,2) = 5+1=6$$

$$f(5,2) = \psi(4,6,2) = 6+1=7$$

Нетрудно показать, что  $f(y, x) = y + x$ . Действительно,  $f(y+z, x) = f(y, x) + z$ . Полагая в этом равенстве  $y = 0$ , получим  $f(z, x) = f(0, x) + z$  или  $f(z, x) = x + z$ .

**Пример 2.** Пусть функция  $f(y, x)$  задана равенствами:

$$f(0, x) = 0$$

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

Здесь  $\varphi(x) = 0$ ,  $\psi(x, y, z) = y + z$ .

Вычислим значение функции  $f(y, x)$  при  $y = 2$ ,  $x = 2$ . Так как  $f(0, x) = \varphi(x) = 0$ ,

то  $f(0, 2) = 0$ , а значения  $f(1, 2)$  и  $f(2, 2)$  находим последовательно:

$$f(1, 2) = \psi(1, 0, 2) = 0 + 2 = 2$$

$$f(2, 2) = \psi(2, 2, 2) = 2 + 2 = 4$$

$$f(3, 2) = \psi(2, 4, 2) = 4 + 1 = 5$$

Легко показать, что в этом примере  $f(y, x) = x \cdot y$ . Действительно,  $f(y + z, x) = f(y, x) + z \cdot x$ . Полагая в этом равенстве  $y = 0$ , получим  $f(z, x) = f(0, x) + z \cdot x$  или  $f(z, x) = z \cdot x$ .

### 1.3. Операция минимизации ( $\mu$ -оператор)

Пусть задана некоторая функция  $f(x, y)$ . Зафиксируем значение  $x$  и выясним, при каком  $y$   $f(x, y) = 0$ .

Более сложной задачей является отыскание для данной функции  $f(x, y)$  и фиксированного  $x$  наименьшего из тех значений  $y$ , при которых функция  $f(x, y) = 0$ . Так как результат решения задачи зависит от  $x$ , то наименьшее значение  $y$ , при котором функция  $f(x, y) = 0$  есть функция  $x$ . Принято обозначение

$$\varphi(x) = \mu y [f(x, y) = 0].$$

(Читается: «наименьшее  $y$  такое, что  $f(x, y) = 0$ ».)

Аналогично определяется функция многих переменных:

$$\varphi(x_1, x_2, \dots, x_n) = \mu y [f(x_1, x_2, \dots, x_n, y) = 0].$$

Переход от функции  $f(x_1, x_2, \dots, x_n, y)$  к функции  $\varphi(x_1, x_2, \dots, x_n)$  принято называть применением  $\mu$ -оператора.

Для вычисления функции  $\varphi$  можно предложить следующий алгоритм:

1. Вычислим  $f(x_1, x_2, \dots, x_n, 0)$ . Если это значение  $f$  равно нулю, то полагаем  $\varphi(x_1, x_2, \dots, x_n) = 0$ . Если  $f(x_1, x_2, \dots, x_n, 0) \neq 0$ , то переходим к следующему шагу.

2. Вычислим  $f(x_1, x_2, \dots, x_n, 1)$ . Если  $f(x_1, x_2, \dots, x_n, 1) = 0$ , то полагаем  $\varphi(x_1, x_2, \dots, x_n) = 1$ . Если же  $f(x_1, x_2, \dots, x_n, 1) \neq 0$ , то переходим к следующему шагу. и т.д.

Если окажется, что для всех  $y$  функция  $f(x_1, x_2, \dots, x_n, y) \neq 0$ , то функцию  $\varphi(x_1, x_2, \dots, x_n)$  в этом случае считают неопределенной. Но возможно, что существует такое  $y_0$ , что  $f(x_1, x_2, \dots, x_n, y_0) = 0$  и, значит, есть и наименьшее  $y$ , при котором  $f(x_1, x_2, \dots, x_n, y) = 0$ ; и в то же время может случиться, что при некотором  $z$  ( $0 < z < y_0$ ) значение функции  $f(x_1, x_2, \dots, x_n, z)$  не определено. Очевидно, что в этом случае процесс вычисления наименьшего  $y$ , при котором  $f(x_1, x_2, \dots, x_n, y) = 0$  не дойдет до  $y_0$ . И здесь функцию  $\varphi(x_1, x_2, \dots, x_n)$  считают неопределенной.

**Пример.** Рассмотрим функцию  $f(x, y) = x - y$ , которая может быть получена с помощью оператора минимизации

$$f(x, y) = \mu z (y + z = x) = \mu z [I_3^2(x, y, z) + I_3^3(x, y, z) = I_3^1(x, y, z)]$$

Вычислим, например,  $f(7, 2)$ , то есть значение функции при  $y = 2$ ,  $x = 7$ . Для этого положим  $y = 2$  и будем придавать  $x$  последовательно значения:

$$z=0, 2+0=2 \neq 7$$

$$z=1, 2+1=3 \neq 7$$

$$z=2, 2+2=4 \neq 7$$

$$z=3, 2+3=5 \neq 7$$

$$z=4, 2+4=6 \neq 7$$

$$z=5, 2+5=7=7$$

Таким образом,  $f(7, 2) = 5$ .

**Определение 2.** Функция  $f(x_1, x_2, \dots, x_n)$  называется частично рекурсивной, если она может быть получена в конечное число шагов из простейших функций при помощи операций суперпозиции, схем примитивной рекурсии и  $\mu$ -оператора.

**Определение 3.** Функция  $f(x_1, x_2, \dots, x_n)$  называется общерекурсивной, если она частично рекурсивна и всюду определена.

Примерами общерекурсивных функций являются функции:

- 1)  $\lambda(x)$ ,                      2)  $O(x)$ ,                      3)  $I_n^m(x)$ ,  
 4)  $f(y, x) = y + x$ ,      б)  $f(y, x) = xy$ ,      б)  $f(y, x) = x + n$ .

**Тезис А. Чёрча.** Каждая интуитивно вычислимая функция является частично рекурсивной.

Этот тезис нельзя доказать, т.к. он связывает нестрогое математическое понятие интуитивно вычислимой функции со строгим математическим понятием частично рекурсивной функции. Но этот тезис может быть опровергнут, если построить пример функции интуитивно вычислимой, но не являющейся частично рекурсивной.



## Лекция 20. Уточнение понятия алгоритма Маркова

### Учебные вопросы:

1. Нормальные алгоритмы Маркова.
2. Неразрешимые алгоритмические проблемы.

### 1. Нормальные алгоритмы Маркова

Как и ранее, будем называть алфавитом  $A$  всякое непустое конечное множество символов, а сами символы алфавита будем называть буквами.

Словом в алфавите  $A$  называется всякая конечная последовательность букв алфавита  $A$ . Пустая последовательность букв называется пустым словом и обозначается через  $\square$ .

Если  $P$  обозначает слово  $S_1S_2...S_k$  и  $Q$  обозначает слово  $C_1C_2...C_m$ , то  $PQ$  обозначает объединение  $S_1S_2...S_k C_1C_2...C_m$ . В частности,  $PA = AP = P$ . Кроме того,  $(P_1P_2)P_3 = P_1(P_2P_3)$ .

Алфавит  $A$  называется расширением алфавита  $B$ , если  $B \subset A$ . Очевидно, что в этом случае всякое слово в алфавите  $B$  является словом в алфавите  $A$ .

Алгоритмом в алфавите  $A$  называется эффективно вычислимая функция, областью определения которой служит какое-нибудь подмножество множества всех слов в алфавите  $A$  и значениями которой являются также слова в алфавите  $A$ . Пусть  $P$  есть слово в алфавите  $A$ ; говорят, что алгоритм  $U$  применим к слову  $P$ , если  $P$  содержится в области определения  $U$ . Если алфавит  $B$  является расширением алфавита  $A$ , то всякий алгоритм в алфавите  $B$  называется алгоритмом над алфавитом  $A$ .

Большинство известных алгоритмов можно разбить на некоторые простейшие шаги. Следуя А. А. Маркову, в качестве элементарной операции, на базе которой строятся алгоритмы, выделим подстановку одного слова вместо другого. Если  $P$  и  $Q$  - слова в алфавите  $A$ , то выражения  $P \rightarrow Q$  и  $P \rightarrow \cdot Q$  будем называть формулами подстановки в алфавите  $A$ . При этом предполагается, что символы стрелка  $\rightarrow$  и точка  $\cdot$  не являются буквами алфавита  $A$ , а каждое слово  $P$  и  $Q$  может быть и пустым словом. Формула подстановки  $P \rightarrow Q$  называется простой, а формула подстановки  $P \rightarrow \cdot Q$  называется заключительной.

Пусть  $P \rightarrow (\cdot)Q$  обозначает одну из формул подстановки  $P \rightarrow Q$  или  $P \rightarrow \cdot Q$ . Конечный список формул подстановки в алфавите

$$P_1 \rightarrow (\cdot)Q_1$$

$$P_2 \rightarrow (\cdot)Q_2$$

.....

$$P_r \rightarrow (\cdot)Q_r$$

называется схемой алгоритма и порождает следующий алгоритм в алфавите  $A$ .

Принято говорить, что слово  $T$  входит в слово  $Q$ , если существуют такие (возможно пустые) слова  $W, V$ , что  $Q = WTV$ .

Пусть  $P$  - слово в алфавите  $A$ . Здесь может быть одно из двух:

1. Ни одно из слов  $P_1, P_2, \dots, P_r$  не входит в слово  $P$  (обозначается:  $U:P \supset$ ).

2. Среди слов  $P_1, P_2, \dots, P_r$  существуют такие, которые входят в  $P$ . Пусть  $m$  - наименьшее целое число такое, что  $1 \leq m \leq r$ , и  $P_m$  входит в  $P$ , и  $R$  - слово, которое получается, если самое левое вхождение слова  $P_m$  в слово  $P$  заменить словом  $Q_m$ . Тот факт, что  $P$  и  $R$  находятся в описанном отношении, коротко запишем в виде

$$U:P \vdash R, \quad (a)$$

если формула подстановки  $P_m \rightarrow (\cdot)Q_m$  - простая, или в виде

$$U:P \vdash \cdot R, \quad (b)$$

если формула  $U:P \vdash \cdot R$  - заключительная.

В случае (a) говорят, что алгоритм  $U$  просто переводит слово  $P$  в слово  $R$ ; в случае (b) говорят, что алгоритм  $U$  заключительно переводит слово  $P$  в слово  $R$ .

Пусть далее,  $U:P \models R$  означает, что существует такая последовательность  $R_0 R_1 \dots R_k$  слов в алфавите  $A$ , что  $P = R_0$ ,  $R = R_k$ ,  $U:R_j \vdash R_{j+1}$  для  $j=0, 1, \dots, k-2$  и либо  $U:R_{k-1} \vdash R_k$  либо  $U:R_{k-1} \vdash \cdot R_k$  (в этом последнем случае вместо  $U:P \models R$  пишут  $U:P \vdash \cdot R$ ).

Положим теперь  $U(P) = R$  тогда и только тогда, когда либо  $U:P \vdash \cdot R$ , либо  $U:P \models R$  и  $U:P \supset$ .

Алгоритм, определенный таким образом, называется нормальным алгоритмом или алгоритмом Маркова.

Работа алгоритма  $U$  может быть описана следующим образом. Пусть дано слово  $P$  в алфавите  $A$ . Находим первую в схеме алгоритма  $U$  формулу подстановки  $P_m \rightarrow (\cdot)Q_m$  такую, что  $P_m$  входит в  $P$ .

Совершаем подстановку слова  $Q_m$  вместо самого левого вхождения слова  $P_m$  в слово  $P$ . Пусть  $R_1$  – результат такой подстановки. Если  $P_m \rightarrow (\cdot) Q_m$  – заключительная формула подстановки, то работа алгоритма заканчивается, и его значением является  $R_1$ . Если формула подстановки  $P_m \rightarrow (\cdot) Q_m$  – простая, то применим к  $R$ , тот же поиск, который был только что применен к  $P$  и т.д. Если на конечном этапе будет получено такое слово  $R$ , что  $U:R_i \supset$ , то есть ни одно из слов  $P_1, \dots, P_r$  не входит в  $R_i$ , то работа алгоритма заканчивается, и  $R_i$  будет его значением.

Если описанный процесс на конечном этапе не заканчивается, то говорят, что алгоритм  $U$  не применим к слову  $P$ .

**Пример 1.** Пусть  $A$  есть алфавит  $\{b, c\}$ . Рассмотрим схему

$b \rightarrow \cdot \Lambda$

$c \rightarrow c.$

Определяемый этой схемой нормальный алгоритм  $U$  перерабатывает всякое слово  $P$  в алфавите  $A$ , содержащее хотя бы одно вхождение буквы  $b$ , а слово, которое получается вычеркиванием в  $P$  самого левого вхождения буквы  $b$ .

Действительно, всякая буква  $c$ , находящаяся в слове левее самой левой буквы  $b$ , простой подстановкой  $c \rightarrow c$  переводится в букву  $c$ , а самая левая буква  $b$  заключительной подстановкой переводится в пустое слово  $\square$ .

Например, если  $P = cscbbcs$ , то  $P \rightarrow \cdot Q$ , где  $Q = cscbc$ .

Пустое слово  $U$  перерабатывает в само себя.  $U$  не применим к непустым словам, не содержащим вхождения буквы  $b$ . Действительно, если слово  $P$  содержит только буквы  $c$ , то простой подстановкой  $c \rightarrow c$  оно будет перерабатываться в себя, но тогда всегда  $P \rightarrow P$ , а мы не приходим к заключительной подстановке, т.е. процесс будет продолжаться бесконечно.

## 2. Некоторые неразрешимые алгоритмические проблемы

Переход от интуитивного понятия алгоритма к точному понятию машины Тьюринга позволяет уточнить и вопрос об алгоритмической разрешимости данной массовой проблемы. Теперь этот вопрос можно сформулировать так: существует ли машина Тьюринга,

решающая данную массовую проблему или же такой машины не существует?

На этот, вопрос теория алгоритмов в ряде случаев дает отрицательный ответ. Один из первых результатов такого типа получен американским математиком Чёрчем в 1936 году. Он касается проблемы распознавания выводимости в математической логике.

## **2.1. Неразрешимость проблемы распознавания выводимости в математической логике**

Как известно, аксиоматический метод в математике заключается в том, что все предложения (теоремы) данной теории получаются посредством формально-логического вывода из нескольких предложений (аксиом), принимаемых в данной теории без доказательства.

В математической логике описывается специальный язык формул, позволяющий любое предложение математической теории записать в виде вполне определенной формулы, а процесс логического вывода из посылки  $A$  следствия  $B$  может быть описан в виде процесса формальных преобразований исходной формулы. Это достигается путем использования логического исчисления, в котором указана система допустимых преобразований, изображающих элементарные акты логического умозаключения, из которых складывается любой, как угодно сложный формально-логический вывод.

Вопрос о логической выводимости предложения  $B$  из посылки  $A$  в избранном логическом исчислении является вопросом о существовании дедуктивной цепочки, ведущей от формулы  $A$  к формуле  $B$ .

В связи с этим возникает проблема распознавания выводимости: для любых двух формул  $A$  и  $B$  в логическом исчислении узнать, существует ли дедуктивная цепочка, ведущая от  $A$  к  $B$ , или нет.

Решение этой проблемы понимается в смысле вопроса о существовании алгоритма, дающего ответ при любых  $A$  и  $B$ . Результат Чёрча формулируется следующим образом:

**Теорема Чёрча.** Проблема распознавания выводимости алгоритмически неразрешима.

## 2.2. Проблема эквивалентности слов для ассоциативных исчислений

Первые результаты об алгоритмической неразрешимости были установлены для проблем, возникающих в самой математической логике и в теории алгоритмов. Сюда относятся и рассмотренная проблема распознавания выводимости. Позже выяснилось, что аналогичные проблемы возникают в самых различных специальных разделах математики. Сюда относятся, в первую очередь, алгебраические проблемы, приводящие к различным вариантам проблемы слов.

Рассмотрим некоторый алфавит  $A = \{a, b, c, \dots\}$  и множество слов в этом алфавите. Если слово  $L$  является частью слова  $M$ , то говорят, что слово  $L$  входит в слово  $M$ . Так, слово  $аса$  входит в слово  $бсасаб$ , начиная с буквы  $a$ .

Будем рассматривать преобразование одних слов в другие с помощью некоторых допустимых подстановок вида  $P \rightarrow Q$  или  $P \rightarrow Q$ , где  $P$  и  $Q$  два слова в том же алфавите  $A$ .

Применение ориентированной подстановки  $P \rightarrow Q$  к слову  $R$  возможно в том случае, когда в нем имеется хотя бы одно вхождение левой части  $P$ ; оно заключается в замене любого одного такого вхождения соответствующей правой частью  $Q$ .

Применение неориентированной подстановки  $P \rightarrow Q$  допускает как замену вхождения левой части правой, так и замену вхождения правой части левой.

Будем рассматривать, в основном, неориентированные подстановки.

**Пример.** Подстановка  $ас - аса$  применима к слову  $бсасаб$  двумя способами; замена вхождения  $аса$  в это слово дает слово  $бсасб$ , а замена вхождения  $ас$  дает слово  $бсасааб$ . К слову  $абсаб$  эта подстановка не применима.

**Определение.** Ассоциативным исчислением называется совокупность всех слов в некотором алфавите вместе с какой-нибудь конечной системой допустимых подстановок.

Для задания ассоциативного исчисления достаточно указать соответствующие алфавит и систему подстановок.

Если слово  $R$  может быть преобразовано в слово  $S$  посредством однократного применения допустимой подстановки, то и  $S$  может быть преобразовано в  $R$  таким же путем. В таком случае  $R$  и

$S$  называют смежными словами. Последовательность слов  $R_1, R_2, \dots, R_{n-1}, R_n$  таких, что каждая пара слов  $R_i$  и  $R_{i+1}$  ( $i=1, 2, \dots, n-1$ ) являются смежными, называют дедуктивной цепочкой, ведущей от слова  $R$  к слову  $S$ .

Если существует дедуктивная цепочка, ведущая от слова  $R$  к слову  $S$ , то, очевидно, существует и дедуктивная цепочка, ведущая от слова  $S$  к слову  $R$ , в этом случае слова  $R$  и  $S$  называют эквивалентными и обозначают:  $R \sim S$ .

Для каждого ассоциативного исчисления возникает своя специальная проблема эквивалентности слов:

Для любых двух слов в данном исчислении требуется узнать, эквивалентны они или нет.

Проблема эквивалентности слов для ассоциативных исчислений была сформулирована в 1911 году. Тогда же был предложен алгоритм для распознавания эквивалентности слов в некоторых ассоциативных исчислениях специального вида.

Естественно возникла задача об отыскании такого общего алгоритма, который был бы применим к любому ассоциативному исчислению.

В 1946 и 1947 годах российский математик А. А. Марков и американский математик Э. Пост, независимо один от другого, построили конкретные примеры ассоциативных исчислений, для каждого из которых проблема эквивалентности слов алгоритмически не разрешима, и, следовательно, не существует алгоритма для распознавания эквивалентности слов в любом исчислении.

## Список литературы

1. Лихтарников Л.М., Сукачева Т.Г. Курс лекций по математической логике. – СПб.: Издательство «Лань», 1998. – 288 с.
2. Колесников Н.Г. Математические и логические основы информатики. – Краснодар: КубГАУ, 2000. – 224 с.
3. Калбертсон Дж.Т. Математика и логика цифровых устройств. – М.: Просвещение, 1965. – 267 с.
4. Кук Д., Бейз Г. Компьютерная математика. – М.: Наука, 1990.
5. Ершов Ю.Л., Палютин Е.А. Математическая логика. – М.: Наука, 1979.
6. Верещагин Н.К., Шень А. Начала теории множеств. – М.: МЦНМО, 1999. – 127 с.