1. **Возникновение и развитие дисциплины «Теория алгоритмов». Математическое понятие алгоритма.**

Считается, что слово *алгоритм* произошло от имени выдающегося математика средних веков аль-Хорезми (Абу Абдаллах Мухаммеад ибн Муса аль-Хорезми), жившего ориентировочно в VIII-IX в. Около 825 г. он опубликовал трактат «Книга об индийской арифметике», где обобщил правила десятичных операций сложения, вычитания, умножения, деления. В дальнейшем все десятичные вычисления стали называть «алгоризм» или «алгорисмус». Затем слово трансформировалось в «алгорифм», а во второй половине XX века в «*алгоритм*».

Таким образом, в XX веке общепринятым стало следующее определение:   
«*Алгоритм* – конечный набор правил, позволяющий чисто механически решать любую конкретную задачу из некоторого класса однотипных задач».

До 30-ых годов XX века это устраивало всех, общей теории алгоритмов не существовало. Если какой-либо исследователь утверждал, что для некоторого класса задач существует алгоритм, он просто приводил пример. Но со временем положение изменилось, стали появляться проблемы с поиском алгоритмов; впервые встал вопрос об отсутствии алгоритмов для некоторых классов задач. В современной математике принято следующее определение:

**Алгоритм** *– конструктивно задаваемые соответствия между словами в абстрактных алфавитах.*

**Абстрактный алфавит** – *любая конечная совокупность объектов, называемых буквами.*

**Слово** в алфавите – *любая упорядоченная последовательность букв.*

1. **Алфавитный оператор. Общность понятия, способы задания. Примеры простых кодирующих операторов.**

**Алфавитным оператором (или отображением)** *называется всякое соответствие (или функция), сопоставляющее словам в том или ином алфавите слова в том же или другом алфавите. Первый алфавит называется при этом входным, а второй - выходным алфавитом данного оператора.*

С понятием оператора интуитивно связано понятие *сложности*. Наиболее простые операторы те, которые осуществляют побуквенное отображение. Более сложными являются кодирующие операторы, когда слова в алфавите A кодируются словами в другом алфавите B. Если коды всех букв A имеют одинаковую длину, то кодирование называется *нормальным*.

*Необходимыми и достаточными условиями кодирования являются:*

* Коды различных букв алфавита *A* различны.
* Ни один код не должен являться префиксом другого кода.

**Лемма**. *Любой реальный прибор, преобразующий информацию, может рассматриваться как устройство, реализующее некоторый алфавитный оператор.*

В основе теории алфавитных операторов лежит способ задания операторов.

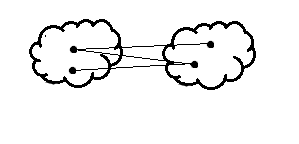
* *Составление таблицы* *соответствий* (если область определения оператора конечна)
* *Составление формулы, описывающей правило, при помощи которого за конечное число шагов по входному слову можно установить выходное слово* (область определения бесконечна)

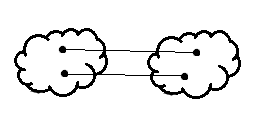
1. **Многозначные и однозначные алфавитные операторы. Примеры.**

Суть однозначных операторов заключается в однозначном установлении соответствия между элементами множества входных слов (область определения) и элементами множества выходных слов (область значения).

Примером однозначного алфавитного оператора может служить попарное отображение элементов множества P = {1, 2, 3, ...} в элементы множества Q = {1, 2, 6, ...}.

Тем не менее, нередки случаи использования многозначных алфавитных операторов, которые позволяют сопоставить некоторому входному слову любой элемент из подмножества множества выходных слов (или наоборот – подмножеству множества входных слов сопоставляют одно выходное слово).

Примером многозначного алфавитного оператора может служить отображение элементов множества P = {(1, 2, 3),(1, 3, 2),(2, 1, 3), ...} в элементы множества Q = {(1, 2, 3)}.



1. **Соотношение между алфавитным оператором и алгоритмом. Стохастические алгоритмы. Самоизменяющиеся алгоритмы.**

*Алфавитные операторы*, *задаваемые с помощью конечных систем правил, называются* **алгоритмами***.*

Из определения вытекает, что любой алфавитный оператор, который можно задать, непременно оказывается алгоритмом. Тем не менее, алфавитный оператор задает только соответствие между словами двух алфавитов A и B; в отличие от него, алгоритм дает возможность по входному слову найти выходное слово. Другими словами, алгоритм есть не что иное, как алфавитный оператор вместе с правилами, определяющими его действие.

Два **алфавитных оператора** считаются **равными**, если у них одна область определения и любому входному слову из этой области они сопоставляют одинаковые выходные слова.

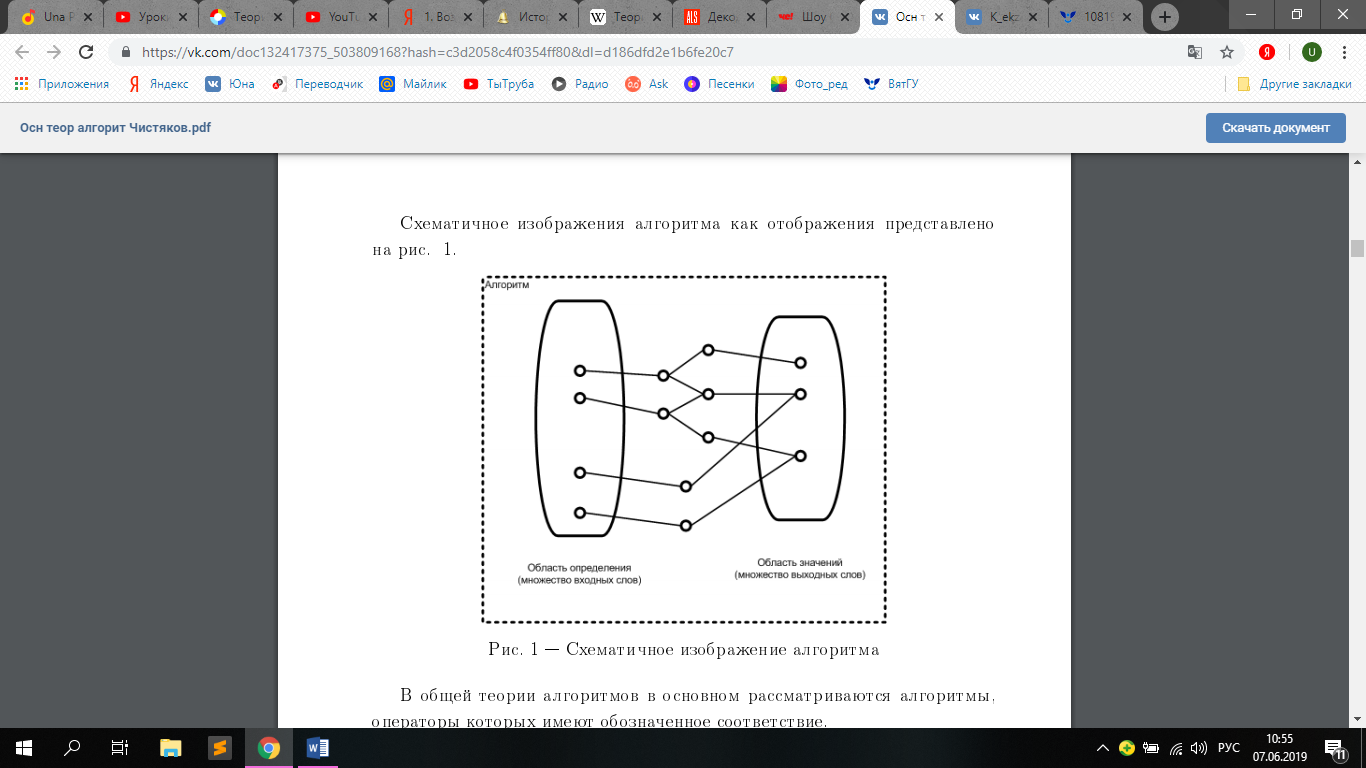
Два **алгоритма** считаются **равными**, если их алфавитные операторы равны, а системы правил совпадают. В случае, когда системы правил различны, алгоритмы называются **эквивалентными**.

При решении некоторых практических задач требуется расширить понятие алгоритма, вводя в систему правил его реализации такие механизмы, которые позволяют выбирать входные слова случайным образом, так же, как и сами правила. Вероятность выбора может фиксироваться заранее или определяться по ходу выполнения алгоритма. Такие алгоритмы называют **стохастическими**, или случайными.

Нужно отметить как отдельный класс **самоизменяющиеся** алгоритмы, которые не только перерабатывают входные слова, но и сами претерпевают изменения в процессе переработки. Результат обработки входного слова P такого алгоритма зависит не только от этого слова, но и от предыстории работы алгоритма.

Понятие самоизменяемости применимо в общем случае как к детерминированным, так и к стохастическим алгоритмам. В последнем случае это может выразиться в изменении вероятностей различных выходных слов.

Схематичное изображения алгоритма как отображения:



В общей теории алгоритмов в основном рассматриваются алгоритмы, операторы которых имеют обозначенное соответствие.

1. **Свойства алгоритмов. Понятие алгоритмической системы.**

Основные свойства алгоритмов:

* **Дискретность** – процесс сопоставления входному слову выходного должен быть представим как последовательное выполнение конечного числа некоторых шагов. Кроме того, для выполнения каждого шага должно требоваться конечное время.
* **Детерминированность** – в каждый момент времени следующий шаг процесса сопоставления должен быть однозначно определяем. Кроме того, каждому входному слову P должно соответствовать только одно выходное слово Q. Любой стохастический алгоритм может быть представлен как детерминированный путем включения вероятностных характеристик в состав алгоритма.
* **Понятность** – все шаги процесса сопоставления должны быть доступны для осуществления исполнителю.
* **Завершаемость** – для любого входного слова из области определения алгоритм должен выполнять процесс сопоставления ему выходного слова за конечное число шагов.
* **Массовость** – алгоритм должен быть применим для решения не только одной задачи, а целого класса однотипных задач (мощность области определения должна быть больше единицы).
* **Результативность** – в конечном итоге применение алгоритма должно приводить к получению выходного слова.

*Всякий общий способ задания алгоритмов называется* **алгоритмической системой**.

Все направления в прикладной теории алгоритмов условно делятся на два направления – *алгебраические* и *геометрические*.

**Алгебраические** формальные средства используют символику, когда алгоритмы рассматриваются в виде линейных текстов (рекурсивные функции, операторные системы Ван-Хао, логические схемы алгоритмов Ляпунова, Янова и другие).

В **геометрической** теории алгоритмы строятся в виде множеств, в которых реализуются отображения или бинарные отношения, а также используется математический аппарат теории графов (нормальные алгоритмы Маркова, граф-схемы алгоритмов, матричные схемы алгоритмов и прочее).

1. **Понятие абстрактного вычислителя. Машина Поста.**

Наиболее простой формальной алгоритмической системой является машина Поста, предложенная американским математиком польского происхождения Эмилем Леоном Постом.

С точки зрения структуры машины Поста может быть представлена как совокупность двух элементов.

* Лента (бесконечная полоска, разделенная на ячейки равной длины)
* Каретка (считывающая и записывающая головка).

*Жестко сопоставленные каждой секции уникальные номера образуют целочисленную* **базовую систему координат машины Поста**.

**Временной системой координат** *машины Поста называется полученная в результате сдвига на фиксированное число относительно базовой система координат.*

*Информация о том, какие секции отмечены, а какие не отмечены называется* **состоянием ленты**.

1. **Понятие программы машины Поста. Диаграмма и схема Поста.**

Работа машины Поста состоит в том, что каретка передвигается вдоль ленты и печатает или стирает метки. Этот процесс происходит согласно инструкциям определенного вида. Пронумерованная натуральными числами совокупность таких инструкций называется **программой машины Поста**.

* **l α** – команда сдвига каретки влево.
* **r α –** команда сдвига каретки вправо.
* **p α –** команда постановки отметки в секцию, напротив которой находится каретка.
* **e α –** команда удаления отметки из секции, напротив которой находится каретка.
* **j α β** **–** команда передачи управления. Если напротив каретки находится неотмеченная секция, то осуществляется переход к команде с номером α, иначе к команде с номером β.
* **s –** команда остановки.

Номер следующей команды, расположенный в конце каждой инструкции, называется **отсылкой**.

**Программой машины Поста** называется конечный непустой список команд, обладающий двумя свойствами. Во-первых, на k-ой позиции списка должна стоять команда с номером k. Во-вторых, отсылка любой из команд списка должна совпадать с номером некоторой (другой или той же самой) команды списка.

Для некоторой программы и начального состояния ленты работа машины Поста может закончиться одним из двух вариантов.

* В ходе выполнения программы машина дойдет до выполнения невыполнимой команды (печать метки в отмеченной секции, удаление метки из неотмеченной секции, отсылка к команде с номером, превышающим число инструкций в программе). В таком случае выполнение программы прекращается. Происходит так называемая безрезультатная остановка.
* В ходе выполнения программы машина дойдет до выполнения команды остановки. В таком случае программа считается выполненной, машина останавливается, происходит результативная остановка.
* Кроме того, возможен еще один вариант. В ходе выполнения программы машина не дойдет до выполнения ни одной команды остановки, также не произойдет безрезультатной остановки. Таким образом, выполнение программы никогда не прекратится – процесс работы будет происходить бесконечно.

Совокупность положения каретки в базовой системе координат и номер текущей команды в выполняемой программе называется **состоянием каретки**.

Совокупность состояния ленты и состояния каретки называется **состоянием машины** **Поста**.

Состояние машины Поста позволяет однозначно определить следующее выполняемое действие. Фактически, процесс работы машины Поста заключается в последовательных переходах машины от одного состояния к другому до тех пор, пока не произойдет останов.

Эффективным способом анализа программ для машины Поста является построения специализированной **диаграммы Поста**. На такой диаграмме команды изображаются кружками (называемыми также узлами), а переходы от одной команды к другой – стрелками (называемыми также дугами).

Для лучшего понимания сути анализируемой программы целесообразным, как правило, оказывается разделение узлов на группы в зависимости от их назначения. Каждый блок в таком случае изображается в виде прямоугольника, содержащего уникальный номер, список включенных в него команд из диаграммы и текстовую метку, описывающую функциональное назначение блока. Описанная структура получила название **схемы Поста**.

1. **Понятие абстрактного вычислителя. Машина Тьюринга.**

Одной из самых популярных алгоритмических систем является так называемая машина Тьюринга (МТ). Она была предложена в 1936 году одним из наиболее значимых исследователей в области computer science - англичанином Аланом Тьюрингом.

С технической точки зрения *Алгоритмическая система МТ* состоит из следующих составных элементов.

* Неограниченная в обе стороны лента, разделенная на ячейки.
* Управляющее устройство.
* Чувствительная головка для анализа содержимого ячеек и работы с ним.

**МТ** *называется стандартной, если при сдвиге ленты может быть изменено содержимое обозреваемой ячейки.*

*Любую совокупность из последовательности содержимого ячеек МТ и одного из внутренних состояний будем называть* **конфигурацией машины Тьюринга**.

*Конечная совокупность команд, заданная на одних и тех же внутренних и внешних алфавитах*, называется **программой машины Тьюринга**. Программа МТ называется также «системой команд».

Таким образом, «задать МТ» - значит решить следующие задачи.

* Задать внешний алфавит (с пустым символом).
* Задать внутренний алфавит (с заключительными состояниями).
* Задать программу работы МТ.
* Задать начальную конфигурацию МТ.

1. **Тезис Тьюринга. Полнота по Тьюрингу. Вычислимые по Тьюрингу функции.**

**Тезис**. Любой процесс, который было бы естественно назвать эффективной процедурой (алгоритмом), может быть реализован алгоритмической системой машины Тьюринга.

Данный тезис является главным результатом, полученным А. Тьюрингом. Он имеет чрезвычайно важное значение и используется для доказательства существования алгоритмов при решении самых различных классов задач. С математической точки зрения тезис недоказуем, так как понятие «любой процесс» не является строгой категорией. Но вся практическая деятельность подтверждает его высокую достоверность.

**Теорема**. Любая функция, которая может быть вычислена с помощью машины Тьюринга, может быть также вычислена с помощью машины Поста.

**Функция f(x) вычислима по Тьюрингу**, если ее значения могут быть вычислены некоторой МТ, на ленте которой первоначально было задано стандартное представление аргумента x. Значением f(x) будет запись на ленте после останова МТ.

Выяснилось, что существует МТ для любой функции, вычислимой по Тьюрингу, если надлежащим образом подобрать представление аргументов (это и есть стандартное представление x).

Вычислитель полный по Тьюрингу если может реализовать любую вычислимую функцию.

1. **Универсальная машина Тьюринга. Пример работы.**

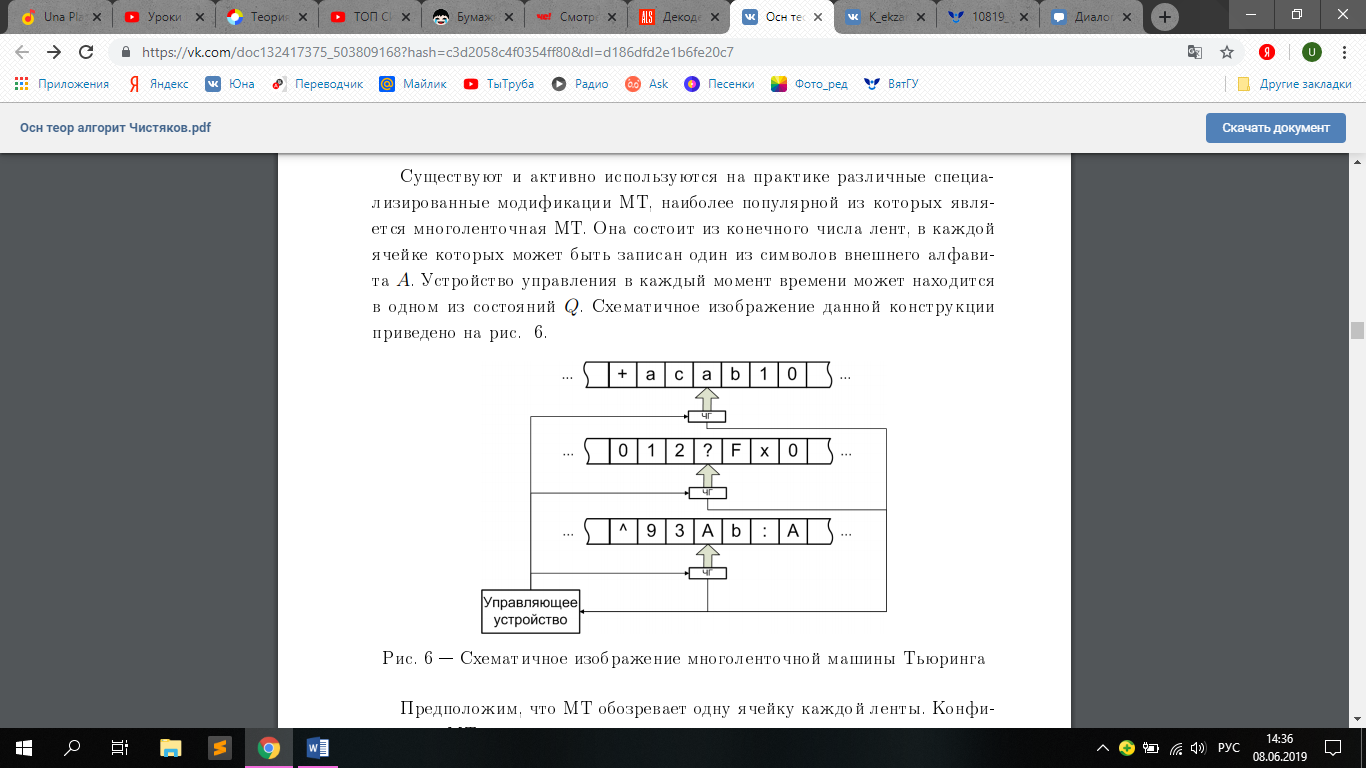
Можно заметить, что различные алгоритмы реализуются на различных машинах Тьюринга. Эта проблема решается путем кодирования конфигураций и программы некоторой заданной МТ в символах внешнего алфавита универсальной МТ по следующим правилам.

* Различные символы заменяются кодовыми группами.
* Строки кодовых записей должны быть разбиты на кодовые группы.
* Отдельные кодовые группы соответствуют операциям движения ленты МТ.

Современные ЭВМ в основном строятся как универсальные, но в отличие от теоретической МТ физически реализованные машины не имеют неограниченной памяти.

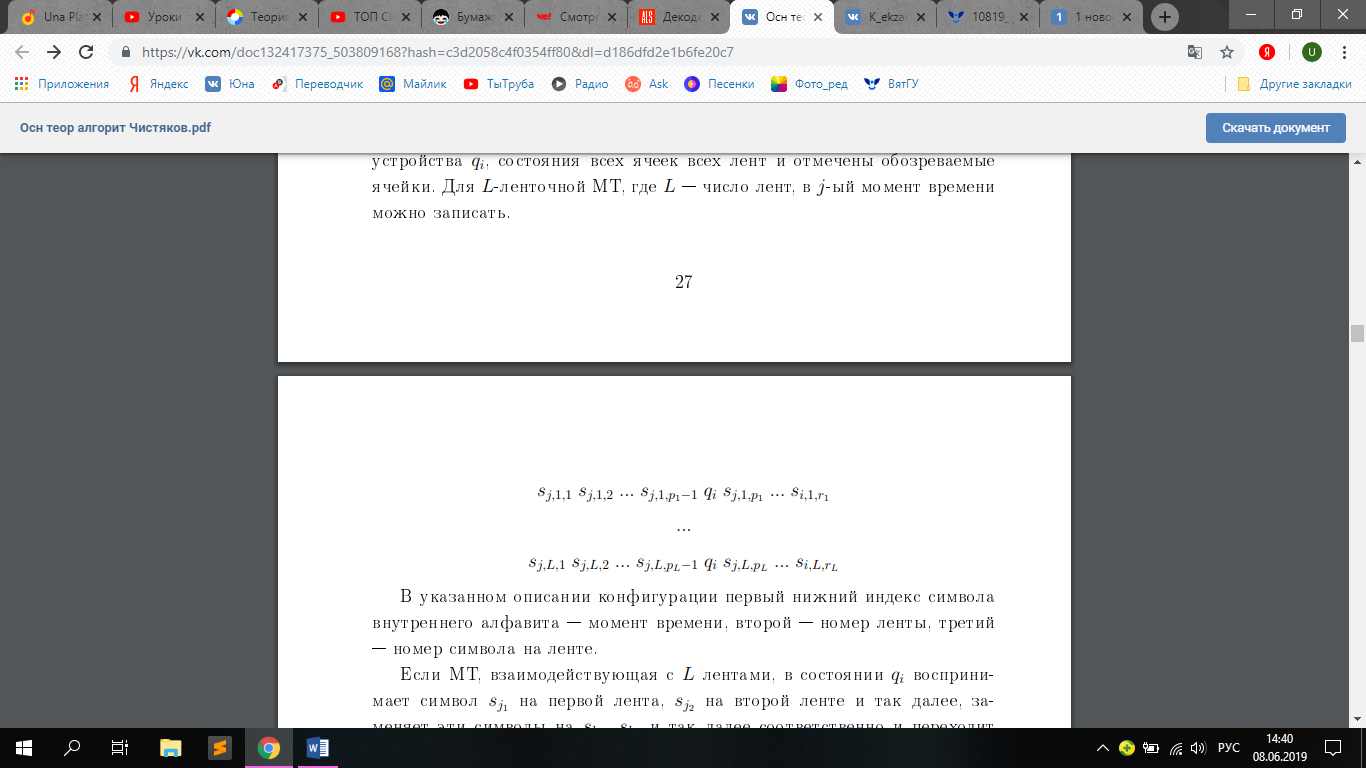
Универсальные МТ могут иметь различные структуры. Для сопоставления этих структур между собой К. Шеннон предложил специальную меру – произведение числа символов на число внутренних состояний. Эта мера определяет «сложность» универсальной МТ. Известны следующие результаты - теоремы о существовании универсальной МТ.

* *Ватанабе* - пять символов на шесть состояний.
* *Минский* - четыре символа на семь состояний.
* *Триттер* - четыре символа на шесть состояний.

1. **Модификация машины Тьюринга. Примеры работы.**

Существуют и активно используются на практике различные специализированные **модификации МТ**, наиболее популярной из которых является *многоленточная МТ.* Она состоит из конечного числа лент, в каждой ячейке которых может быть записан один из символов внешнего алфавита. Устройство управления в каждый момент времени может находится в одном из состояний.

Предположим, что МТ обозревает одну ячейку каждой ленты. Конфигурация МТ считается заданной, если известно состояние управляющего устройства qi, состояния всех ячеек всех лент и отмечены обозреваемые ячейки. Для L-ленточной МТ, где L - число лент, в j-ый момент времени можно записать.



В указанном описании конфигурации первый нижний индекс символа внутреннего алфавита - момент времени, второй - номер ленты, третий - номер символа на ленте. Если МТ, взаимодействующая с L лентами, в состоянии qi воспринимает символ sj1 на первой лента, sj2 на второй ленте и так далее, заменяет эти символы на sk1, sk2 и так далее соответственно и переходит в следующее состояние ql, возможно, осуществляя при этом сдвиг чувствительных головок на одну позицию влево или вправо, то будем говорить, что МТ выполняет команду. Необходимо отметить, что МТ останавливается после выполнения команды, если ql принадлежит множеству заключительных состояний.

1. **Композиция машины Тьюринга: произведение, возведение   
   в степень, итерация. Пример.**

Совершенно очевидно, что с математической точки зрения МТ - определенный алгоритм. Известно, что существуют операции над алгоритмами, позволяющие получать более сложные алгоритмы.

**Произведением MT1 на MT2** *будем называть MT с общим внешним алфавитом A = {a0, a1, ..., am} и суммарным внутренним алфавитом Q = {q0, q1, ..., qn, ..., qn+k} и программой задаваемой следующим образом. Во всех командах MT1, содержащих переход в заключительное состояние q˙0, производится замена его на символ qn+1. Во всех команда MT2 наоборот: переход в q˙0 остается неизменным, но все остальные символы qj (j = 1, 2, ..., k) заменяются символом qn+j. Совокупность команд MT1 и MT2 и будет представлять программу произведения MT = MT1 · MT2. Другими словами работа МТ эквивалента работе последовательно включенных MT1 и MT2.*

**Возведением в степень машины MT** *назовем последовательное произведение MT на саму себя n раз.*

**Операцией итерации** *называется отождествление одного из конечных состояний MT с начальным.*

Следует заметить, что при этом МТ должна иметь более одного заключительного состояния. В противном случае будет получена МТ без возможности останова.

1. **Понятие абстрактного вычислителя. Машина с бесконечными регистрами.**

Около 80 лет назад появилось понятие вычислимой функции. Это послужило основанием для создания новой ветви математических исследований в области теории машинных вычислений, в философии, во многих областях обработки информации.

Начальной целью теории вычислимости было придать научную строгость интуитивной идее вычислимой функции, чтобы затем приступить к математическому исследованию данного понятия.

Если алгоритм используется для вычисления значений числовой функции, то говорят, что эта функция «эффективно вычислима».

Рассмотрим один из идеализированных компьютеров, который получил название «**машина с бесконечными регистрами**» (МБР), лишенный ограничений по разрядности представления чисел и по объему памяти.

МБР включает в себя бесконечное число регистров R1, R2, ..., каждый из которых может содержать любое натуральное число. Машина может изменять содержимое регистров по некоторой команде, которая является чрезвычайно простой операцией над числами. В МБР существует четыре вида команд.

* Команда **обнуления Z(n).**
* Команда **прибавления единицы S(n).**
* Команда **переадресации T(m, n).**
* Команда **условного перехода J(m, n, q).**

Чтобы производить вычисления, нужно задать программу P и так называемую начальную конфигурацию (заполнение регистров R1, R2, ...). Вычисления идут до тех пор, пока не кончится список команд.

1. **МБР-вычислимые функции. Элементарные вычислимые функции.**

В общем случае будем использовать обозначения:

* **↓** - вычисление в определенный момент останавливается (сходится);
* ↑ - вычисление никогда не останавливается (расходится);

Если вычисление расходящееся, то есть никогда не кончается, это означает, что значение функции вычислимо не на всех наборах аргументов; функция в этом случае является **частичной**.

*Пусть F - частичная функция на множестве целых неотрицательных N = {0, 1, 2, 3, ...}. Если задана программа P, начальная конфигурация (a1, a2, ..., ai, ..., as), и получается некоторый результат b, где ai, b ∈ N, то*

* Вычисление P(a1, a2, ..., as) сходится к b, когда P(a1, a2, ..., as) ↓ и в заключительной конфигурации в регистре R1 находится b (R1 := b);
* Программа P вычисляет F, если для всех a1, a2, ..., as, b имеет место P (a1, ..., as) ↓ b, тогда и только тогда, когда (a1, a2, ..., as) ∈ DomF и F(a1, a2, ..., as) = b;
* Функция F является МБР-вычислимой, если есть программа, которая МБР-вычисляет F.

Класс всех МБР-вычислимых функций обозначим как φ, класс n-местных МБР-вычислимых функций через φn.

**Лемма**. Следующие функции вычислимы:

* Нуль-функция 0(x) = 0 для всех x ∈ N;
* Функция следования s(x) = x + 1;
* Для всех n ≥ 1 и 1 ≤ i ≤ n функция проекции определена как Vni (x1, ..., xn) = xi.

Вычислимость данных функций на МБР не требует доказательства, т.к. они соответствуют арифметическим командам обнуления (Z(1)), прибавления единицы (S(1)) и переадресации (T(i, 1)).

1. **Порождение вычислимых функций в машине с бесконечными регистрами.**

Для построения более сложных вычислительных функций в теории МБР используются некоторые специальные операции.

**Соединение программ**.

**Программа** P = I1, I2, ..., Is **имеет «стандартный вид»**, *если для всякой команды условного перехода J(m, n, q) в P имеем q ≤ s+1.*

Даны P и Q - программы стандартного вида длины s и t соответственно. **Конкатенацией (соединением)** P и Qназываетсяпрограмма P · Q = I1, ..., Is, Is+1, ..., Is+t , где каждый условный переход из программы Q J(m, n, q) заменен на J(m, n, s + q).

**Подстановка программ.**

**Теорема**. Пусть f(y1, ..., yk), g1(x1, ..., xn), g2(x1, ..., xn), ..., gk(x1, ..., xn) - вычислимые функции. Тогда функция h(x1, ..., xn) = f(g1, ..., gk), также вычислима.

**Рекурсия.**

Рекурсия - это способ задания функции путем определения каждого из ее значений в терминах ранее определенных значений для младших аргументов (и, возможно, использования других уже определенных функций).

**Теорема**. Пусть f(x1, ..., xn) и g(x1, ..., xn, y, z) вычислимые функции, тогда функция h(x1, ..., xn, y), полученная из f и g с помощью рекурсии, вычислима.

**Минимизация.**

**Теорема**. Пусть f(x1, ..., xn, y) вычислимая функция, тогда вычислима функция g(x1, ..., xn) = µy(f(x1, ..., xn, y) = 0).

Используя операцию минимизации совместно с операциями подстановки и рекурсии, можно построить большее число функций, чем только при помощи последних двух.

Как показали теоретические исследования все новые функции, какими бы они не были, относятся к МБР-вычислимым. Другими словами, *класс φ замкнут относительно выше указанных операций.*

1. **Параллельная машина с бесконечными регистрами. Пример.**

*Под* **параллельной машиной с бесконечными регистрами** *(пМБР) понимается расширение классической машины, способное одновременно выполнять произвольное число программ P1, P2, .... Доступ к регистрам при этом возможен из каждой программы, то есть регистры являются общей (разделяемой) памятью.*

Множество команд пМБР дополнено пятью элементами.

* Команда **захвата ресурса G(n).** Эта команда предписывает заблокировать регистр Rn и разрешить доступ к нему только из программы, выполнившей блокировку. Другие обратившиеся к такому регистру программы входят в состояние ожидания и пребывают в нем до того момента, пока регистр не будет освобожден.
* Команда **освобождения ресурса P(n).** Эта команда предписывает разблокировать регистр Rn и разрешить доступ к нему из всех программ.
* Команда **запуска программы s(k, n).** Предписывает начать выполнение программы с номером k, сопоставив ему регистр с номером Rn.
* Команда **ввода I(n)** предписывает прочитать из проассоциированного с машиной устройства ввода значение и занести его в регистр Rn.
* Команда **вывода O(n)** предписывает перенаправить на проассоциированное с машиной устройство вывода значение регистра Rn.

Другим важным усовершенствованием является поддержка косвенной регистровой адресации. В командах Z(n), S(n), T(m, n), J(m, n, q), I(n), O(n), G(n), P(n) и s(k, n) вместо непосредственного указания номера регистра n или m разрешается использовать передачу значения через вспомогательный регистр. Например, команда S(n) с использованием косвенной регистровой адресации примет вид S(R(n)), что будет означать «увеличить значение регистра с номером, соответствующим значению регистра с номером n, на единицу».

1. **Нормальные алгоритмы Маркова. Определение. Граф-схема алгоритма. Заключительная подстановка. Пример.**

*Всякий общий способ задания алгоритма принято называть* **алгоритмической системой**. Обычно алгоритмические системы включают в себя объекты двоякой природы - элементарные операторы и элементарные распознаватели.

**Элементарные операторы** - *это достаточно простые (то есть просто задаваемые) алфавитные операторы, с помощью последовательного выполнения которых могут быть реализованы любые алгоритмы в данной алгоритмической системе.*

**Элементарные распознаватели** - *это объекты для распознавания наличия тех или иных свойств обрабатываемой информации и изменения последовательности, в зависимости от результата распознавания, элементарных операторов.*

Практика показала, что для указания набора элементарных операторов и порядка их следования при реализации алгоритма удобно использовать направленные графы особого рода **- граф-схемы алгоритмов** (ГСА), предложенные Л.А. Калужниным, и представляющие собой конечное множество соединенных между собой стрелками вершин - узлов ГСА. Каждому узлу, кроме входного и выходного, сопоставляется какой-либо элементарный оператор или распознаватель. Из каждого узла выходит либо одна стрелка (операторный узел), либо две (распознавательный узел), а число входных стрелок неограниченно.

В 1953 г. академик **А.А. Марков** предложил и исследовал функционирующую схожим образом алгоритмическую систему, которая получила название «**нормальные** **алгоритмы**».

В нормальных алгоритмах используют только один тип элементарного оператора – *подстановка* и один тип элементарного распознавателя - *вхождение*.

*Если p и q - два слова алфавита A, то говорят, что* **слово q входитв слово p***, если p может быть представлено p = p1 · q · p2, где p1 и p2 некоторые слова (возможно, пустые). При этом вхождение слова q в слово p называется первым, если слово p1 имеет наименее возможную длину среди всех допустимых представлений слова p.*

**Соединением слов p и q** *называется слово R = p · q, получаемое путем приписывания (конкатенации) слова q справа к слову p.*

**λ - пустое** **слово** в алфавите A: λ · p = p · λ = p.

*Говорят, что* **слово p начинается словом q***, если есть слово R, такое, что p = q · R.*

**Слово p равно слову q***, если они содержат одинаковые буквы алфавита A в одинаковом порядке.*

*Если R = p · q · s и V - некоторые слова в A, то образование нового слова p · V · s называется* **подстановкой слова V вместо слова****q** *везде, где оно встречается, а новое слово X = p · V · s - результат подстановки.*

Традиционно, НА задаются не ГСА, а упорядоченными системами подстановок всех операторов (схема алгоритма). Порядок выполнения подстановок полностью определяется указанными выше пунктами. Произвольная подстановка i должна выполняться тогда и только тогда, если она является первой из применимых подстановок. Процесс оканчивается, когда ни одна из подстановок не применима или выполнена заключительная подстановка.

Таким образом, необходимым условием для универсальности НА является *использования как обычных, так и заключительных подстановок*.

1. **Обобщенный нормальный алгоритм. Принцип нормализации. Виды композиций нормальных алгоритмов.**

*Алгоритмы, заданные ГСА, составленные исключительно из распознавателей вхождения слов и операторов подстановки, называются* **обобщенными нормальными алгоритмами***. При этом предполагают, что каждому оператору подстановки вида (p1 → p2) подсоединяется только одна стрелка: стрелка ѕ+ї от распознавателя вхождения p1.*

**Нормальными алгоритмами** *(НА) называют такие обобщенные нормальные алгоритмы, ГСА которых имеют некоторый специальный вид, когда всякий оператор (p1 → p2) входит в паре с распознавателем.*

Объединенная ГСА, в случае нормальных алгоритмов, должна удовлетворять ряду следующих условий.

* Все объединенные узлы ГСА упорядочиваются путем нумерации от 1 до N, стрелка со знаком ѕ-ї i-го узла присоединяется к (i + 1)-ому узлу, а N-ого узла к выходному;
* Положительные выходы подсоединяются либо к первому, либо к выходному узлу ГСА;
* Входной узел подсоединяется к первому объединенному (распознавательному и, одновременно, преобразовательному) узлу.

**Тезис (принцип нормализации)**. *Для любого алгоритма (конструктивно задаваемого алфавитного отображения) в произвольном конечном алфавите A можно построить эквивалентный ему нормальный алгоритм над алфавитом A.*

Как показал А.А. Марков, если можно построить НА, эквивалентный заданному, добавив к алфавиту A некоторое количество букв (возможно, очень большое), то оказывается можно провести это построение НА, добавив к алфавиту A всего лишь одну букву.

Принцип нормализации нельзя доказать, поскольку понятие «произвольного» алгоритма не является строгим математическим понятием. Его необходимо воспринимать как один из естественнонаучных законов. Однако его считают в высшей степени достоверным. В сокращенном варианте он звучит так: «Все алгоритмы нормализуемы».

Одним из распространенных видов **композиции** алгоритмов является «**суперпозиция**», когда выходное слово некоторого алгоритма A является входным для алгоритма B. Результатом этой операции следует считать D(p) = B(A(p)); суперпозиция может быть применена к любому количеству алгоритмов.

**«Объединением»** двух алгоритмов A и B в одном и том же алфавите X называют алгоритм C в том же алфавите, который преобразует любое входное слово p из области определения, являющейся пересечением областей определения A и B, в записанные рядом слова A(p)B(p). На других входных словах алгоритм C считается неопределенным.

**«Разветвлением»** алгоритмов называют композицию трех алгоритмов A, B, C, результатом которой является алгоритм D. Считается, что область определения D совпадает с пересечением областей определения всех трех алгоритмов, а для любого слова p из этого пересечения D(p) = A(p), если C(p) = e или D(p) = B(p), если C(p) 6= e.

**«Повторение» (итерация)** - это композиция двух алгоритмов A и B с результатом P. Для любого входного слова q выходное P(q) определяют так: существует ряд слов q = q0, q1, q2, ..., qn = P(q) такой, что для всех i = 1, 2, 3, ..., n qi = A(qi − 1), для всех i = 1, 2, 3, ..., n − 1 B(qi) 6= e, а B(qn) = e, т.е. алгоритм A применяется последовательно несколько раз до тех пор, пока не получится слово, перерабатываемое алгоритмом B в пустое e.

1. **Универсальный нормальный алгоритм.**

Для всякой универсальной алгоритмической системы важным обстоятельством является задача построения так называемого **«универсального» алгоритма**. Поэтому встает задача построить нормальный алгоритм, способный выполнять функции любого алгоритма, если задана схема (то есть система подстановок) последнего.

**Теорема**. *Существует такой НА V, называемый универсальным, который для любого НА A и любого входного слова p переводит слово AUpU, полученное приписыванием изображения слова p к изображению алгоритма A, в слово, являющееся изображением соответствующего входного слова A(p).*

Эта теорема является очень важным математическим результатом. Дело в том, что из этой теоремы как следствие вытекает возможность построения машины, способной реализовать любой алгоритм нормализованного вида, то есть в принципе любого алгоритма. А специальное кодирование - это программа.

1. **Алгоритмическая система рекурсивных функций. Понятие рекурсивной функции. Понятие вычислимой функции. Пример.**

Данная алгоритмическая система сложилась исторически первой (1931- 1934 гг.), ввиду чего получила достаточно полное и всестороннее развитие.

В ее основании лежит использование конструктивно определяемых арифметических целочисленных функций, которые получили специальное название - *рекурсивные функции.*

**Рекурсией** *называют способ задания функции, при котором значение определяемой функции для произвольных значений аргументов выражается через значения определяемой функции для меньших значений аргументов.*

Применение рекурсивных функций в теории алгоритмов основано на идее нумерации слов в произвольном алфавите последовательными числами.

*Функция y = f(x1, x2, ..., xn) называется* **алгоритмически вычислимой***, если существует алгоритм, позволяющий определить значение функции при любых значениях переменных x1, x2, ..., xn, входящих в область определения f.*

*Класс вычислительных функций шире класса всех элементарных функций.*

1. **Соотношение элементарной и вычислимой функции. Определение функции по индукции. Пример.**

Попробуем ответить на вопрос: все ли вычислительные функции являются элементарными? Другими словами - шире ли класс вычислимых функций класса элементарных? Чтобы получить ответ на этот вопрос, будем усложнять элементарные функции, взяв за критерий сложности скорость роста значений функций.

Из всех элементарных функций быстрее всего растет произведение, так как a\*n - это n раз повторенное сложение a+a+a+a+...+a. Очевидно, что умножение есть итерация сложения.

Операция возведения в степень, в свою очередь, есть итерация умножения: an = a · ... · a. И та и другая функции являются элементарными, так как выражаются через сложение и умножение.

Рассмотрим еще более быстро растущую функцию, которая является итерацией возведения в степень. Эта функция растет чрезвычайно быстро. Оказывается, что она не может быть реализована с помощью элементарных функций.

Таким образом, итерация операции возведения в степень позволяет получить неэлементарную функцию. В то же время она заведомо вычислима. Это позволяет сделать вывод о том, что **класс вычислительных функций шире класса всех элементарных функций.**

Обратим внимание на тот факт, что ψ(x, a) была задана по **индукции**, то есть вначале было определено начальное значение ψ(0, a) *базис*, а также указан *способ вычисления всех ее значений по предыдущим значениям с помощью вполне доступных операций*. Отметим, что определение по индукции может быть введено на любом упорядоченном множестве, где введены понятия «предыдущий» и «следующий».

1. **Алгоритмическая система рекурсивных функций. Полная система формального описания схем.**

Используя в качестве исходных функций элементарные функции, можно с помощью небольшого числа конструктивных приемов строить все более и более сложные арифметические функции.

**Полная система формального описания схем** для получения произвольных функций может быть представлена следующим образом.

1. φ(x) = S(x) = x’;

2. φ(x1, x2, ..., xn) = Cnq = q;

3. φ(x1, x2, ..., xn) = Xni = xi;

4. φ(x1, x2, ..., xn) = ψ(χ1(x1, x2, ..., xn), ..., χm(x1, x2, ..., xn));

5. ┌ φ(0) = q;

└ φ(x’) = ψ(x, φ(x)).

6. ┌ φ(0, x1, x2, ..., xn) = g(x1, x2, ..., xn);

└ φ(y’, x1, x2, ..., xn) = h(y, φ(y, x1, x2, ..., xn), x2, x3, ..., xn).

Схемы под номерами 1-4 задают первоначальные функции, играя роль аксиом, а схемы 5 и 6 можно рассматривать как правила вывода.

1. **Понятие примитивно-рекурсивной функции и ее связь с вычислимой функцией. Частично рекурсивные функции. μ-оператор.**

*Функция φ(x1, x2, ..., xn) называется* **примитивно рекурсивной***, если она может быть определена с помощью конечного числа применений схем 1-6↑.*

Сложение и произведение аргументов x и y есть примитивно-рекурсивная функция. К этому же классу относятся возведение в степень и многие другие операции.

*Класс вычислимых функций шире, чем класс примитивно-рекурсивных функций*. Это показали независимо друг от друга Р. Петер и В. Аккерман, что означает, что средства построения вычислимых функций нуждаются в расширении. Операторы рекурсии не дают желаемого замыкания класса всех вычислимых функций. Подходящим для этой цели оказался **µ-оператор**.

Функция называется **частично-рекурсивной**, если она построена из простейших функций константы Cnq, тождества Xni, следования S(x) и с помощью применения конечного числа операций суперпозиции, примитивной рекурсии и µ-оператора.

Определим операцию с применением **µ-оператора как «операцию наименьшего корня»**. Эта операция позволяет определить новую арифметическую функцию   
f(x1, x2, ..., xn) от n переменных с помощью ранее построенной арифметической функции g(x1, x2, ..., xn, y) от n + 1 переменной. Для x1 = a1, x2 = a2, ..., xn = an в качестве соответствующего значения f(a1, a2, ..., an) определяемой функции принимается наименьший целый неотрицательный корень y = β из уравнения g(a1, a2, ..., an, y) = 0.

{С помощью операции минимизации определяется новая арифметическая функция f(n) с помощью ранее построенной арифметической функции g(n+1).

Это реализуется следующим методом.

Для некоторого заданного набора значений x1 = a1, x2 = a2,…, xn = an в качестве значения функции f(x1,…,xn) = y принимается наименьший целый неотрицательный корень уравнения g(a1,…,an,y) = 0.

Таким образом, механизм вычисления функции f сводится к тому, что фиксируется набор аргументов и затем решается уравнения.

Данный процесс будет продолжаться бесконечно в следующих случаях:

* если значения f(x1,…,xn, xn-1, 0) в точке 0 для xn не определено;
* f(x1,…,xn-1, y) для y = 1,2,3,…,a-1) в точке y = a значение функции не определено.}

Частично-рекурсивная функция называется **общерекурсивной**, если она всюду определена.

Понятие частично-рекурсивной функции оказалось достаточной формализацией понятия вычислимой функции, что содержится в тезисе А. Черча

**Тезис**. *Всякая функция, вычислимая некоторым алгоритмом, частично рекурсивна.*

1. **Понятие логической схемы алгоритма. Процесс реализации. Распределение сдвигов. Понятие подчиненности. Оператора max.**

**Логической схемой алгоритма (ЛСА)** *называют конечную строку из символов A1, A2, ..., An, логических условий α1, α2, ..., αm и концов стрелок ↓1, ..., ↓m такую, что для каждого начала стрелки ↑i найдется один и только один конец стрелки с тем же индексом.*

Назовем символы операторов и логических условий (ЛУ) «**элементарным выражением**», и всякую строку, состоящую из таких выражений и концов стрелок, так что для каждого индекса i найдется не более одного начала и одного конца стрелки, «**выражением**».

Итак, ЛСА по вышеизложенному определению задает порядок выполнения операторов в зависимости от значений ЛУ.

Тогда процесс реализации ЛСА для произвольной последовательности ∆ может быть определен следующим образом:

* На первом шаге задаем значения переменных в виде ∆s1.
* На втором - отмечаем самое левое элементарное выражение схемы алгоритма U.
* После нескольких последовательно выполняемых шагов получаем: Ai1, Ai2, ..., Ai(L−1).
* Проанализируем элементарное выражение, следующего шага. Если это оператор АiL, то приписываем его к полученной строке справа и задаем значения набора переменных в виде ∆s(L+1), а затем переходим к следующему элементарному выражению. Если же это ЛУ αj (p1, ..., pm) ↑i, то действуем уже описанным способом, то есть при αj = 1 переходим к следующему элементарному выражению справа, иначе к ближайшему справа от конца ↓i.
* Процесс реализации ЛСА заканчивается, если выполнен заключительный оператор An.

**{Распределение сдвигов** *- сопоставленный каждому оператору набор логических переменных, которые могут изменить свое значение после выполнения этого оператора.*

Последовательность распределений сдвигов считается **допустимой**, когда наборы до и после выполнения оператора отличаются только на переменные из распределения сдвигов.}

Введем некоторые понятия и определения, предложенные в работах Ю.И. Янова.

Пусть задана ЛСА U и два множества Φ = {A1, A2, ..., An} и Ψ={p1, ..., pm}, принадлежащие U. Каждому оператору Ai (i=1, 2, …, n) ∈ A поставим в соответствие некоторое множество ЛУ Ψi ⊆ Ψ, которые могут изменять свои значения на обратные во время выполнения оператора Ai Ю.И. Янов назвал это соответствие «**распределением сдвигов**»: Ai − Ψi.

Среди всевозможных вариантов распределений сдвигов различают следующие.

* *Пустое*, когда оператору поставлено в соответствие пустое множество Ψi = λ.
* *Универсальное*, когда оператору поставлено в соответствие множество всех логических переменных Ψi = Ψ.
* *Нормальное*, когда Ψi ⊂ Ψ.

Говорят, что две ЛСА U1 и U2 «**равносильны**» при заданном распределении сдвигов (U1 = U2), если для всякой допустимой последовательности наборов значения этих ЛСА совпадают. Равносильное преобразование состоит в замене одних выражений другими и в изменении взаимного расположения выражений.

Будем говорить, что в ЛСА U(p1, ..., pm, A1, ..., An) элементарное выражение Di при заданном распределении сдвигов «**подчинено**» логической функции α(p1, ..., pm), и записывать этот факт как Di ≺ α(p1, ..., pm), если всякий набор ∆r, при котором должно выполняться Di, обращает функцию α в единицу.

Свойства функции β = **max**(Ψi)α таковы, что если

* α(∆) = 1, то β(∆) = β(∆’) = 1,
* α(∆)=0 и α(∆’) = 0, то β(∆) = β(∆’) = 0,
* α(∆) = 0 и α(∆') = 1, то β(∆) = β(∆’) = 0,

где ∆' - набор значений переменных p1, ..., pm, который отличается от набора ∆ значениями тех переменных, которые принадлежат Ψi. Отсюда следует, что А4, например, подчинен исходной функции А4 ≺ max(Ψ3) β = max(Ψ3)(max(Ψ2)α).

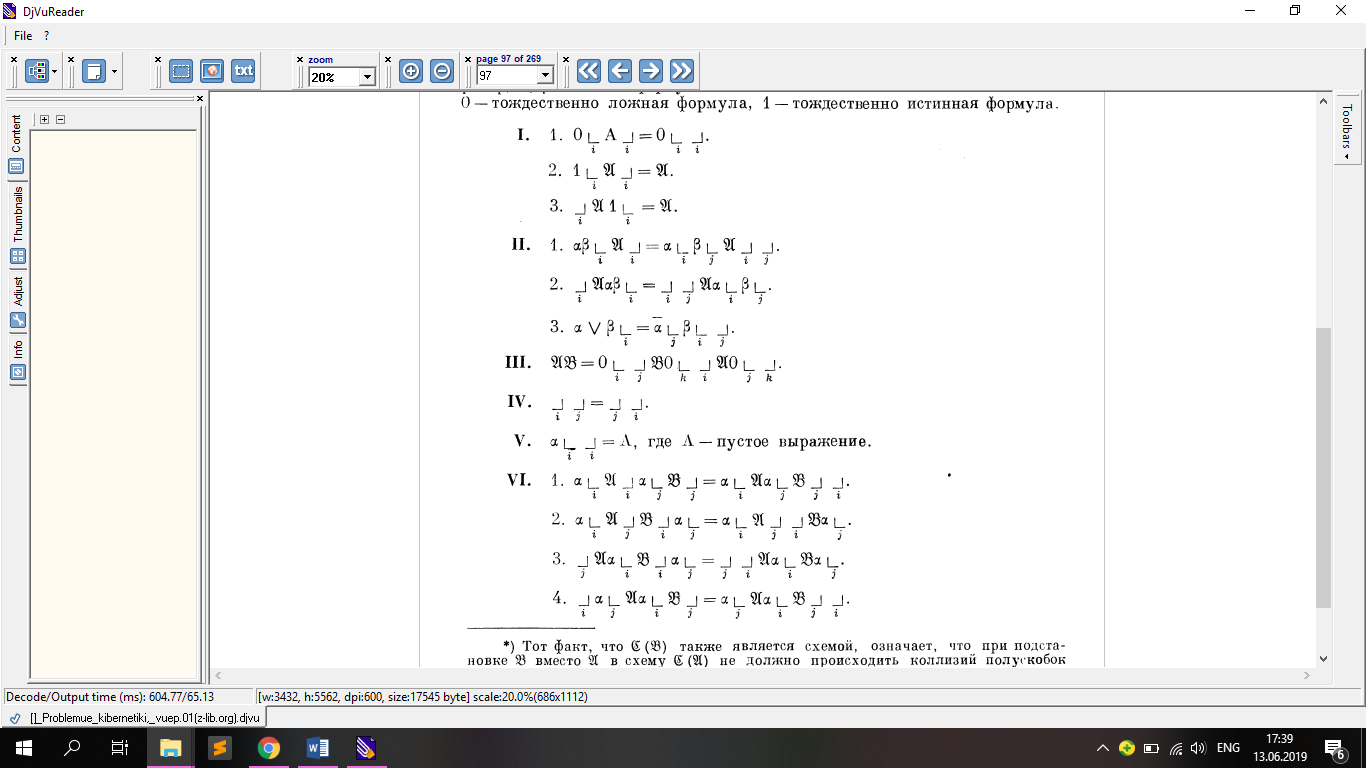
Назовем конъюнкцию логических переменных «**элементарной**», если в ней не содержится повторяющихся ЛУ. Чтобы по заданной логической функции найти ее max по всем наборам переменных из множества Ψi, следует.

* Представить α в ДНФ.
* В каждой элементарной конъюнкции все переменные, входящие в Ψ (в том числе и с отрицанием), заменить единицей.

1. **Полная система преобразований Янова. Отмеченные функции.**

Ю.И.Янов в своих работах предложил специальное ассоциативное исчисление, где формулы представлены как B = G, где B и G - выражения.

Во всякой ЛСА U(B) можно провести операцию подстановки B → G. Это дает возможность получить ЛСА с минимумом ЛУ, но путем перебора всевозможных ЛСА, равносильных заданной. Сложность подобных преобразований чрезвычайно высока, ввиду чего были предприняты усилия, направленные на обхождение указанной проблемы.



Если α, β, γ, ... логические функции от переменных p1, ..., pn, Аi ∈ A, где А - множество операторов А = {A1, ..., Am}, то запись вида αАi (или pjАi) назовем «**отмеченной** **функцией**» (отмеченной переменной).

В работах П.С. Новикова показано, что над логическими функциями, входящими в состав отмеченных, можно производить все преобразования алгебры логики.

Для них справедливы следующие правила, сокращенно называемые системой C1.

* (α ∨ β)Ai = αAi ∨ βAi;
* αβAi ∨ αγAj = α(βAi ∨ γAj );
* α = β → αAi = βAi.

Указанную систему правил совместно с полной системой аксиом алгебры логики следует рассматривать как систему правил тождественных преобразований отмеченных функций.

1. **Система формул перехода S1. Система скобочных формул S2. Система схемных формул S3.**

В работах В.Г. Лазарева и В.Ф. Дъяченко предлагается применять специальные формализмы для особой записи ЛСА, которая позволяет применять уже известные апробированные методы, развитые в теории релейноконтактных схем.

Запись вида «Аi ⇁ Аj» следует читать как «за Аi следует Аj». Очевидно, что после выполнения оператора Аi может следовать лишь один из операторов А1 или А2, или ..., поэтому множество функций αij должно обладать двумя свойствами.

* **Ортогональность** (˄=0);
* **Полнота** (˅=1).

Полное ортогональное множество функций называется «*нормальным*».

Конструктивный способ построения формул перехода заключается в следующем. Необходимо выписать из заданной ЛСА всевозможные последовательности логических переменных (конъюнкций), которые следуют за оператором Аi, после чего построить дизъюнкцию отмеченных логических функций. В общем виде сказанное может быть представлено как **система переходов S1**.

Две формулы перехода называются «*равносильными*», если их значения на данном наборе из любой допустимой последовательности совпадают.

В соответствии с введенной В.Ф. Дьяченко терминологией назовем приведенное по переменным выражение S1 «**скобочной» формулой перехода**, а все формулы перехода

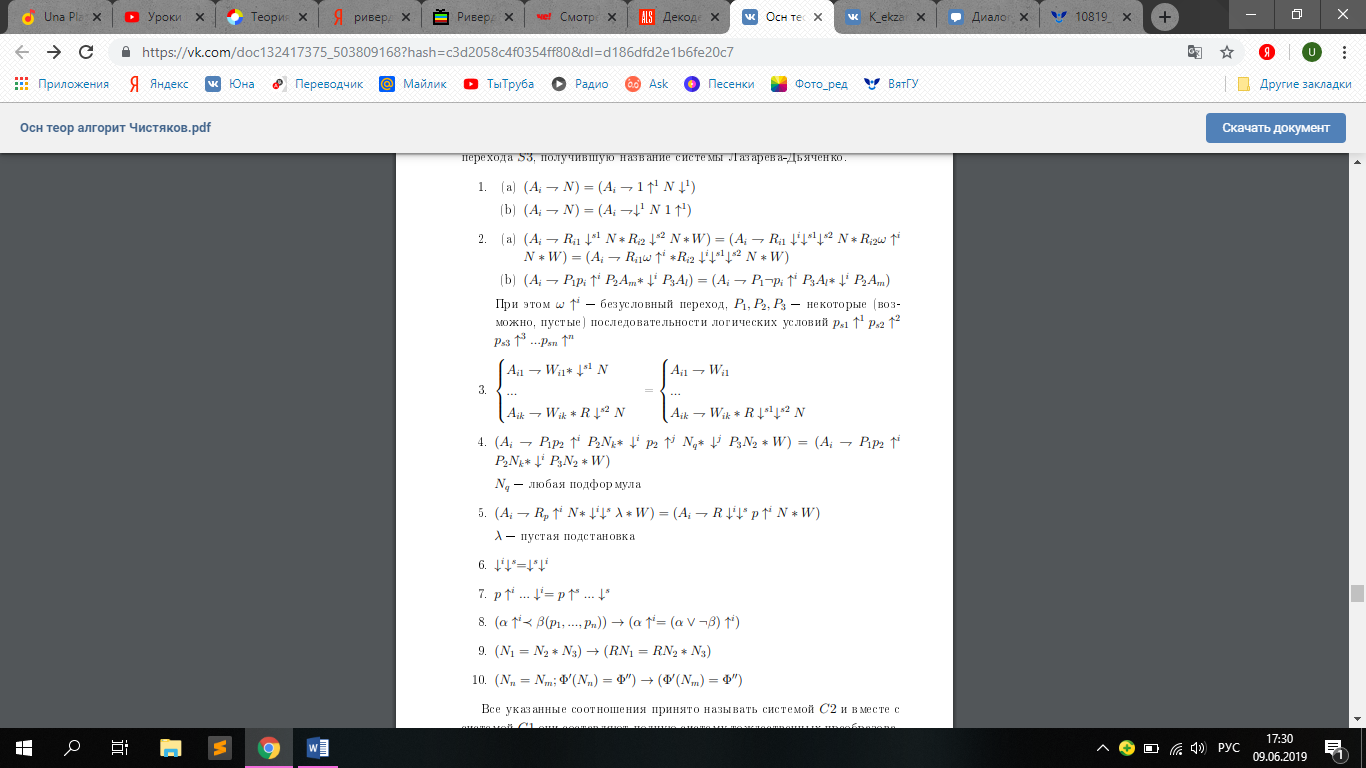
U - «системой скобочных формул перехода» **S2**.

Таким образом, любые ЛСА могут быть представлены системой формул перехода S1, а также системой скобочных формул перехода S2. Однако, существует и третья форма - система схемных формул перехода S3, получаемая из S2 введением в алфавит ЛСА символа разделительного знака «\*».

Для перехода от системы S2 к системе S3 следует воспользоваться следующим набором правил.

* Опустить все скобки формул перехода системы S2.
* Заменить знаки ѕ∨ї равносильными знаками «\*».
* Заменить все пары связанных переменных ps и ¬ps парами ps ↑s\*↓s

1. **Система преобразований Лазарева-Дьяченко. Переход от системы формул к логической схеме алгоритма. Оптимизация логической схемы алгоритма.**



{1-7 – 5 лабораторная}

После цепочки преобразований ЛСА предоставляется возможность произвести оптимизацию схемных формул перехода при помощи полной системы тождественных преобразований.

В качестве заключения следует выделить некоторые практические рекомендации вынесения за скобки логических переменных в формулах перехода.

* Начинать вынесение за скобки следует с той переменной, по которой приведено множество отмеченных элементарных конъюнкций.
* Если множество приведено по нескольким логическим переменным, то можно начинать с любой переменной.
* Если указанное множество отмечено по всем логическим переменным, то порядок безразличен при условии, что в формуле нет пары или более отмеченных элементарных конъюнкций с одинаковыми операторами.

1. **Графическая схема алгоритма. Формальное определение. Оптимизация на уровне ГСА. Пример.**

Как уже упоминалось ранее, граф-схемы алгоритмов были предложены Л.А. Калужниным.

**ГСА - конечный связный граф** G, удовлетворяющий следующим условиям.

* В каждом графе имеется два отмеченных узла - входной, из которого выходит лишь одна стрелка, и выходной, из которого не выходит ни одной стрелки.
* Из любого другого узла выходит либо одна (γ-узел), либо две (β-узел) стрелки.
* Стрелки, исходящие из β-узла, имеют отметки + и -.
* Имеются конечные множества функциональных элементов - множество преобразователей Q = {Qi} и множество распознавателей R = {Rj}, i = 1...n, j = 1...m.
* Каждому γ-узлу однозначно сопоставлен преобразователь Qi, а βузлу - распознаватель Rj, при этом некоторые преобразователи и распознаватели могут быть сопоставлены нескольким различным узлам.

1. **Матричная схема алгоритма. Формальное определение. Оптимизация на уровне МСА. Пример.**

Еще одним наглядным способом представления алгоритмов являются **матричные** **схемы (МСА),** зарекомендовавшие себя как удобный инструмент для анализа систем переходов.

**Матричной схемой алгоритма** *называется квадратная матрица, строки которой соответствуют операторам A0, A1, ..., An, а столбцы - операторам A1, A2, ..., An, Ak. При этом содержимое ячейки [i, j] соответствует составному логическому условию, определяющему возможность перехода от выполнения оператора Ai к выполнению оператора Aj.*

Определим процесс выполнения МСА U для последовательности наборов ∆.

* Задать значения переменных ∆i1 и определить значение функций A1j (j = 1, ..., n, k) в строке 1 матрицы (оператор A0); в связи с требованием ортогональности, найдется ровно одна обращающаяся в истину логическая функция; выполнить переход к j-ой строке матрицы (соответствующей определенному оператору Аj−1).
* Задать значения переменных ∆ij и определить значение истинной функции Aij = 1, при этом найдя строку (и оператор), к которому выполняется очередной переход.
* Процесс продолжается до тех пор, пока не выполнится оператор, символизирующий окончание алгоритма.

В результате будет получена строка Аi1Аi2Аi3...АikАi(k+1)..., являющаяся значением матричной схемы для последовательности наборов ∆.

1. **Объединение схем алгоритма. Определяющие конъюнкции. Кодирование схем.**

Нередко возникает необходимость в реализации устройства, поддерживающего выполнение сразу нескольких алгоритмов. Классическим примером подобной задачи является разработка арифметико-логического устройства (АЛУ) процессора ЭВМ. АЛУ должно иметь возможность осуществлять как стандартные арифметические (сложение, вычитание, умножение, деление, возведение в степень и так далее), так и элементарные логические операции (например, сравнение чисел). Каждой из перечисленных операций соответствует своя последовательность действий, а все устройство в целом должно обеспечивать выполнение каждого из алгоритмов.

Более формально, пусть имеются алгоритмы U1, U2, ..., Ul, заданные в форме ЛСА, и требуется получить некоторый минимальный по некоторому критерию алгоритм U, который, при определенных дополнительных условиях, мог бы быть равносилен любому из U1, U2, ..., Ul.

В роли дополнительных могут использоваться специальные логические условия r1, r2, ..., rk, применяемые в схеме U наряду с p1, p2, ..., pm таким образом, чтобы каждая из возможных их конъюнкций R1, R2, ..., R2 k соответствовала не более чем одному алгоритму из U1, U2, ..., Ul.

*Конъюнкции R1, R2, ..., Rk, сформированные из дополнительных условий r1, r2, ..., rk, сопоставленные алгоритмам U1, U2, ..., Ul, называются* **определяющими конъюнкциями**.

**Объединенной ЛСА U** *называется такая ЛСА, которая соответствует двум условиям*:

* Любой оператор Ai, входящий хотя бы в одну из объединяемых ЛСА U1, U2, ..., Ul, входит ровно один раз в ЛСА U.
* Если в ЛСА U (r1, r2, ..., rk, p1, p2, ..., pm) подставить значения определяющей конъюнкции Ri, то U трансформируется в равносильный, соответствующий Ri, алгоритм.

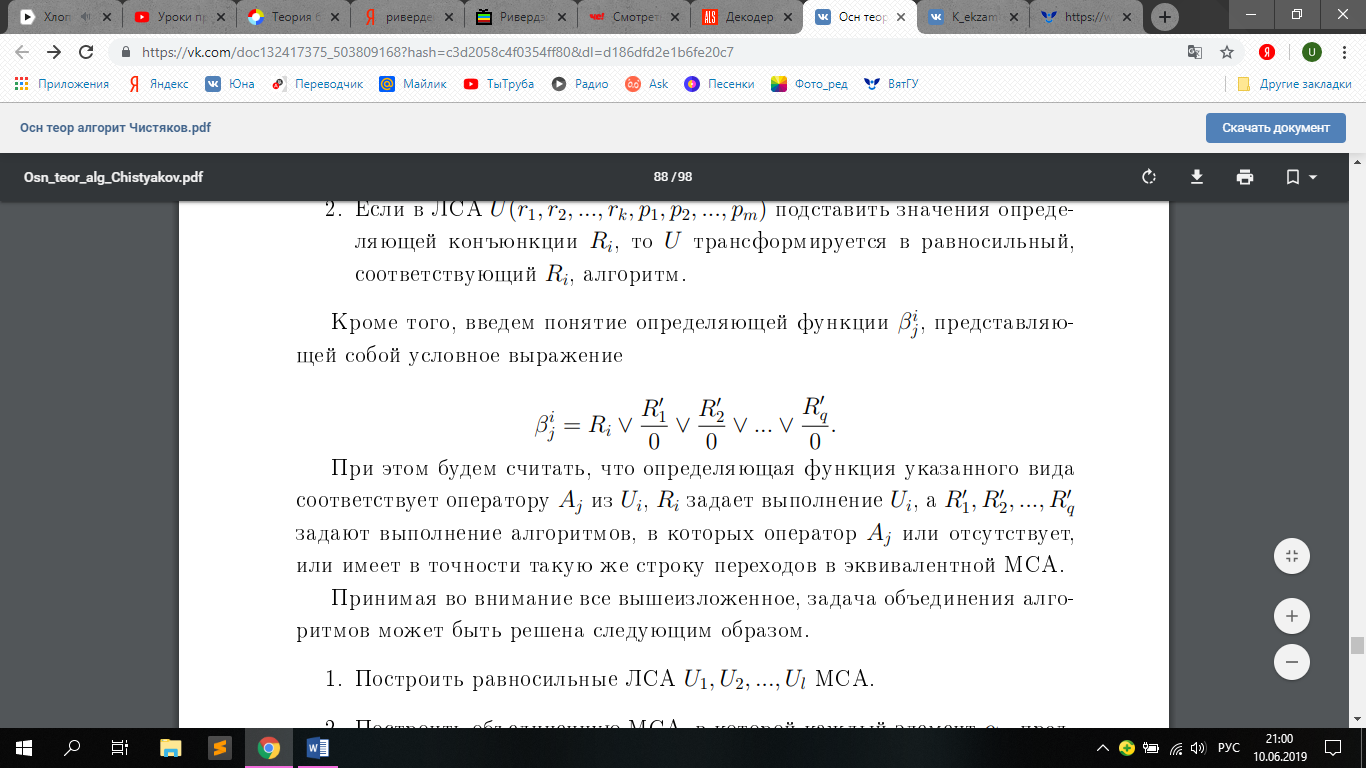
Принимая во внимание все вышеизложенное, задача объединения алгоритмов может быть решена следующим образом.

* Построить равносильные ЛСА U1, U2, ..., Ul МСА.
* Построить объединенную МСА.
* Построить недоопределенную систему формул перехода S1.
* Построить систему скобочных формул перехода S2, доопределив ее.
* Построить систему схемных формул перехода S3.
* Минимизировать S3 по заданному критерию посредством полной системы тождественных преобразований схемных формул.
* Преобразовать S3 в эквивалентную ЛСА U.

Итогом выполнения перечисленных действий будет получение логической схемы объединенного алгоритма U.

1. **Определяющие функции. Процесс доопределения.**

Кроме того, введем понятие определяющей функции βij, представляющей собой условное выражение



При этом будем считать, что определяющая функция указанного вида соответствует оператору Aj из Ui, Ri задает выполнение Ui, а R’1, R'2, ..., R'q задают выполнение алгоритмов, в которых оператор Aj или отсутствует, или имеет в точности такую же строку переходов в эквивалентной МСА.

1. **Алгоритмически неразрешимые проблемы. Примеры.**

Одним из важнейших свойств алгоритма является массовость, то есть способность решать не одну конкретную задачу, а целый класс однотипных задач.

Как показали исследования, существуют такие классы задач, для которых не существует единого универсального приема решения. Такие задачи в теории алгоритмов получили название **алгоритмически неразрешимых проблем**. *Заметим, что алгоритмическая неразрешимость не означает, что такие задачи нельзя решить. Речь идет о невозможности решения одним и тем же приемом.*

Одним из примеров алгоритмически неразрешимой проблемы является **проблема распознавания «самоприменимости».** Существуют самоприменимые и несамоприменимые алгоритмы. Например, тождественный алгоритм в любом алфавите А, содержащем две или более букв, является самоприменимым, способным перерабатывать любое входное слово р ∈ А в само себя. С другой стороны, нулевой алгоритм в алфавите А является несамоприменимым. Этот алгоритм задается схемой, содержащей единственную подстановку «→ y» (y ∈ A). По определению алгоритм неприменим ни к какому входному слову, а значит и к своему изображению.

*Проблема самоприменимости алгоритмов состоит в том, чтобы установить за конечное количество шагов по схеме любого алгоритма является ли он самоприменимым или нет.*

**Теорема**. *Проблема распознавания самоприменимости алгоритма является неразрешимой.*

Из этой теоремы вытекает алгоритмическая неразрешимость более широкой проблемы - **проблемы распознавания применимости**, заключающейся в необходимости установить применимость любой МТ к любой заданной конфигурации.

В терминах нормальных алгоритмов проблема самоприменимости может быть сформулирована следующим образом: *«Требуется найти единственный прием, позволяющий за конечное число шагов по схеме любого алгоритма A узнать, является ли A самоприменимым или нет».*

Природа противоречия, использованного в ходе рассуждений, имеет более глубокие корни и связана с парадоксом Рассела («множество всех множеств, не содержащее себя в качестве своего элемента).

Этот же метод «**от противного**» может быть использован при доказательстве неразрешимости *проблемы* *эквивалентности слов для ассоциативного исчисления*.

**Ассоциативным исчислением** *называется совокупность всех слов в некотором алфавите A с конечной системой подстановок.*

Чтобы задать ассоциативное исчисление достаточно указать алфавит A и систему подстановок. Подстановки бывают вида «p ↔ q» или «p → q», где p, q – слова в алфавите A. Неориентированная подстановка позволяет замену в прямом и обратном направлениях, ориентированная - лишь в прямом.

Если слово r может быть получено из слова s применением лишь одной подстановки, то говорят, что r и s смежные слова. *Если слово r может быть получено из слова s применением конечного числа подстановок, то говорят, что* **смежные слова образуют «дедуктивную цепочку от s к r»**. Слова s и r называются «**эквивалентными**» (s ∼ r), *если существует дедуктивная цепочка от s к r.*

Для отношения эквивалентности слов справедливо следующее.

* Рефлексивность: r ∼ r.
* Симметричность: r ∼ s ↔ s ∼ r.
* Транзитивность: r ∼ t, t ∼ s → r ∼ s.

*Проблема эквивалентности в ассоциативном исчислении состоит в необходимости определения для любых двух слов в данном ассоциативном исчислении эквивалентны они или нет.* Впервые, в 1914 г., данная проблема была сформулирована А. Туэ. Онпредложил некоторые алгоритмы распознавания эквивалентности слов длячастныхспециальных исчислений с повторениями сочетаний букв в словахзаданного алфавита Σ. Туэ назвал слово «*бесквадратным*», если оно несодержит сочетаний вида x2 = xx   
(или «*бескубным*», если оно не содержитсочетаний вида x3), где x ≠ λ.

Суть решаемой Туэ проблемы состояла в построении бесквадратных (бескубных) слов максимальной длины в Σ. После этого многие математики предпринимали усилия для построения такого общего алгоритма, который бы для любого ассоциативного исчисления и для любой пары слов в нем позволял бы установить, являются ли они эквивалентными. В 1947 г. Э. Пост и в 1964 г. А.А. Марков независимо друг от друга построили примеры ассоциативных исчислений, где проблема эквивалентности слов оказалась алгоритмически неразрешимой. Это позволило сделать вывод о том, что искомого алгоритма не существует в принципе. В 1955 г. П.С. Новиков доказал алгоритмическую неразрешимость проблемы распознавании эквивалентности слов для ассоциативного исчисления специального вида теории групп.

Следует отметить, что примеры, построенные для опровержения алгоритмической разрешимости проблемы эквивалентности слов в ассоциативных исчислениях, оказались очень сложными и громоздкими, включающими в себя сотни допустимых подстановок. Однако в 1959 г. Г. Цейкинсну удалось построить пример ассоциативного исчисления, имеющего всего семь допустимых подстановок, для которого проблема распознавания эквивалентности слов оказалась также алгоритмически неразрешимой.

Существует не один десяток примеров алгоритмически неразрешимых проблем. Одним из важнейших является **теорема Геделя о неполноте арифметической логики** (совокупности аксиом и правил вывода в элементарной теории чисел).

**Логика** *называется* **полной**, *если в рамках ее можно доказать истинность или ложность каждого утверждения.*

**Логика** *называется* **непротиворечивой**, *если она свободна от противоречий (например, в ней нельзя получить одновременно истинное утверждение А и ложное А).*

**Теорема**. *Каждая адекватная ω-непротиворечивая арифметическая логика неполна.*

Математический смысл теоремы в том, что в арифметической логике существуют истинные утверждения о целых положительных числах, которые нельзя вывести и доказать средствами этой логики. Кроме того, Гедель показал, что *невозможно доказать непротиворечивость арифметической логики теми методами, которые выразимы в данной логике.*

Оказалось, что такой арифметической логики нет и быть не может, а это значит, что существует бесконечное число проблем элементарной теории чисел, решение которых невозможно никаким аксиоматическим методом.

Одним из практических следствий теоремы Геделя является то, что алгебра Буля в алфавите A = {0, 1}, который есть подмножество B = {0, 1, 2, ..., 9} или А ⊆ В, может иметь также истинные выражения и формулы, которые невозможно вывести из системы тождественных соотношений. Этот результат напрямую касается вопросов проектирования систем обработки информации.

1. **Трудноразрешимые проблемы. Примеры.**

<http://www.mkurnosov.net/teaching/uploads/DSA/dsa-spring2014-lec12.pdf>

Каждая задача при решении имеет свою сложность. Все в алгоритмах оценивается во времени. Это время меряется в количестве операций. Для этого используются асимптотические оценки сложности. Оценка является асимптотической потому что зависит от объема входных данных. Эту зависимость можно описать функцией. Эта временная функция имеет свой некоторый рост.

**Трудноразрешимые проблемы** - такие задачи, для которых не существует алгоритмов, имеющих полиномиальную асимптотическую оценку. Полином n100500.

Хуже полинома идет Экспоненциальная оценка n!, 2n.

Задачи имеющие экспоненциальную оценку - трудноразрешимы так как для решения этих задач требуются большие временные затраты.

Примеры трудноразрешимых задач:

* **Задача Коммивояжера** (теория графов. Обойти все вершины графа, побывав в каждой только 1 раз)
* **Задача раскраски графа** (определить минимальное число цветов, в которое надо раскрасить граф, чтобы соседние вершины были разных цветов)
* **Поиск клики** (каждая вершина подграфа соединена с каждой вершиной этого подграфа)
* **Задача независимого множества** (найти максимальный подграф любые две вершины которого не были бы соединены)
* **Задача о вершинном покрытии**
* **Пятнашки**.
* **Задача выполнимости булевых формул**. Можно ли в формуле найти комбинацию переменных такую, при которой она обратилась бы в истину. Формулы записаны в конъюнктивной форме. Задача решается перебором переменных. Задача SAT

Знать такие проблемы надо чтобы знать, сводится задача к трудноразрешимой или нет.

1. **Трудноразрешимые проблемы теории графов.**

* **Задача о поиске клики**. (Число вершин графа/подграфа, каждая из которых связана с каждой. Пример такого графа, например, что-то типа треугольника. Каждая вершина соединена с каждой вершиной). Суть в поиске максимальной клики. Жизненный пример, формирование некоторой команды. Работа будет продуктивной если все будут друг с другом знакомы.
* **Задача коммивояжера**. Задача о поиске выгодного маршрута между графами, посещая каждую вершину один раз. Гамильтонов цикл
* **Задача о раскраске графа**. Раскрасить вершины графа в минимальное количество цветов так, чтобы не было двух одинаковых смежных цветов.
* **Вершинное покрытие**. Минимальное количество вершин, чтобы все ребра графа были соединены хотя бы с одной вершиной.

1. **Подходы к разрешению трудноразрешимых проблем.**

Примеры трудноразрешимых проблем и подходы к их решению:

* *Задача коммивояжера.*
  + **Полный перебор**. Действие довольно очевидное, но ввиду высокой сложности алгоритма не очень оптимальный
  + **Метод ближайшего соседа**. Эвристический метод решения задачи коммивояжера. Относится к категории жадных алгоритмов. Заключается в том, что следующий пункт для обхода выбирается ближайшим. Так как алгоритм является эвристическим, и основан на жадности, то не факт, что полученное нами решение будет оптимальным или вообще НЕ худшим.
  + **Метод имитации отжига**. Относится к стохастическим алгоритмам. По сути в этом подходе мы работаем со случайными наборами. Относится к алгоритмам Монте Карло.
    - Берем некое начальное значение
    - Генерируем новое
    - Если оно лучше, берем, если хуже, то считаем вероятность перехода в это состояние. Чем выше «температура», тем выше шанс перейти.
    - Дальше либо понижаем температуру, либо ищем новое кандидатское состояние.
  + **Муравьиный алгоритм**. Моделирует поведение муравьев в колонии. Муравьи способны находить кратчайший путь до пищи, оставляя следы феромонов. На каждом шаге муравей вероятностно выбирает путь. Так же каждое ребро графа имеет некую интенсивность феромонов. При проходе по ребру, он оставляет след, обратно пропорциональный длине ребра. Со временем муравьи могут найти приближенно оптимальное решение.
* *Задача раскраски графа*
  + Очевидно, можно **перебрать** все возможные комбинации
  + **Жадный алгоритм**. Вершины упорядочиваются. Каждой вершине последовательно присваивается наименьший доступный цвет, который не используется для окраски соседней, либо добавляется новый цвет.
  + **Эвристические алгоритмы**. Алгоритм решения задачи, правильность которого для всех случаев не доказана, но он дает хорошее решение в данном случае. Эвристика не дает лучшего решения, не гарантирует нахождения решения вообще и может дать неверное решение.
  + **Алгоритм локального поиска**. Это группа алгоритмов. В них поиск ведется только на основании текущего состояния, ранее пройденные не учитываются и не запоминаются. Метод поиска более оптимального решения в окрестности некоторого текущего решения.
  + **Метод ветвей и границ**. Заключается в разбиении рассматриваемого множества с отсевом решений, заведомо не содержащих оптимальных решений.

1. **Асимптотическая оценка сложности алгоритмов. Верхняя, средняя и нижняя оценки.**

Критерии анализа алгоритма:

* Количество переданных по сети данных
* Требование к аппаратному обеспечению
* Временные затраты
* Объем вспомогательных данных
* Размер входных данных

**Асимптотическая эффективность** – *порядок роста временных затрат в пределе*.

**Точная оценка** (**Ө(g(n))**) – для f(n) выражение f(n) = Ө(g(n)) означает, что существует некоторые С1, С2 и n0, что 0<=C1\*g(n)<=f(n)<=C2\*g(n) для всех n>=n0.

**Верхняя оценка** (**Օ(g(n))**) – для f(n) выражение f(n) = Օ(g(n)) означает, что существует некоторые С и n0, что 0< f(n)<=C\*g(n) для всех n>=n0.

**Нижняя оценка** (**Ω(g(n))**) – для f(n) выражение f(n) = Ω(g(n)) означает, что существует некоторые С и n0, что 0<=C\*g(n)<f(n) для всех n>=n0.

1. **Асимптотическая оценка сложности алгоритмов. Амортизированная оценка.**

**Амортизированная стоимость операции** – *это оценка сверху среднего времени выполнения операции в худшем случае (зависит от характера входных данных).*

1. **Оценка рекурсивных алгоритмов. Основная теорема.**

Основная теорема рассматривает следующие рекуррентные соотношения: T(n)=aT(nb)+f(n) где a≥1, b>1.

В применении к анализу алгоритмов константы и функции обозначают:

* n — размер задачи.
* a — количество подзадач в рекурсии.
* n/b — размер каждой подзадачи. (Предполагается, что все подзадачи на каждом этапе имеют одинаковый размер.)
* f (n) — оценка сложности работы, производимой алгоритмом вне рекурсивных вызовов. В неё также включается вычислительная стоимость деления на подзадачи и объединения результатов решения подзадач.

1. **Оценка рекурсивных алгоритмов. Метод деревьев рекурсии.**

В приведенном примере алгоритм рекурсивно разделяет исходную задачу размера n на несколько новых подзадач, каждая размером n/b. Такое разбиение может быть представлено в виде дерева вызовов, в котором каждый узел соответствует рекурсивному вызову, а ветви, исходящие из узла — последующим вызовам для подзадач. Каждый узел будет иметь a ветвей. Также в каждом узле производится объём работы, соответствующий размеру текущей подзадачи n (переданному в данный вызов функции). Общий объем работы алгоритма определяется как сумма всех работ в узлах данного дерева.

1. **Типовые алгоритмические идеи. Разделяй и властвуй. Meet-in-the-middle.**

**Разделяй и властвуй:**

*Задачу дробим на подзадачу при этом мы должны иметь способ решения подзадач и уметь объединять решения этих подзадач. Объединение этих решений должно быть более оптимально чем решение без разделения.*

Пример быстрая сортировка. В худшем случае n2 так как алгоритм сведется к пузырьку. В Среднем же у нас будет примерно n\*log(n).

**Meet-in-the-middle:**

Основная идея - в некоторых задачах более оптимальный поиск может быть *осуществлен, если искать с двух сторон.*

Например, две точки (геометрическая интерпретация). Можем ли мы, расширяя область поиска от одной точки дойти до второй. В таком случае можно искать от двух. Если идти с двух сторон мы сгенерируем две окружности, если между ними будет пересечение, значит мы можем достичь другой точки. Но в итоге сделаем меньше действий.

1. **Типовые алгоритмические идеи. Стохастические алгоритмы. Монте-Карло. Лас-Вегас.**

Иногда нас устраивает получение решения с некоторой вероятностью.

Вероятностные алгоритмы делятся на 2 группы:

* Лас Вегас. Устроены так, что мы можем не получить решения вовсе. («пальцем в небо»)
* Монте Карло. В данном алгоритме мы какое-то решение получим. Приближенное. (попиксельно проверяем)

1. **Типовые алгоритмические идеи. Жадные алгоритмы. Динамическое программирование.**

**Жадный алгоритм:**

*Жадный алгоритм - метод решения оптимизационных задач, основанный на том, что процесс принятия решения можно разбить на элементарные шаги, на каждом из которых принимается отдельное решение. Решение, принимаемое на каждом шаге должно быть оптимальным только на текущем шаге и должно приниматься без учета предыдущих или последующих решений.*

Жадность работает, например, на задаче о непрерывном рюкзаке. У нас допустим есть пески. Они своих масс и стоимостей. Мы вычисляем удельную стоимость каждого вещества и забиваем рюкзак только самым ценным.

Жадность не работает в размене монет. У нас скажем есть сумма 24 копейки. Размениваем 1, 5, 7 копеек. Жадный алгоритм выдаст 3 по 7 копеек и 3 по одной. Но правильное решение две по 7 и две по 5.

**Динамическое программирование:**

*Динамическое программирование. Идея похожа на жадность. Но в данном случае мы учитываем, что можем где-то проиграть, чтобы выиграть в конце.*

Пример - задача о дискретном рюкзаке. Мы конечно можем положить туда ОГРОМНЫЙ предмет с наибольшей удельной стоимостью, но зачем это делать, если можно положить ОГРОМНУЮ КУЧУ маленьких, которые по своей сумме превзойдут этот большой.

*Динамическое программирование заключается в разбиении исходной задачи на задачи меньшего размера, результаты которых мы сможем использовать в текущей.*

**Методы динамического программирования:**

* Метод программирования сверху, когда мы запоминаем результаты подзадач, которые могут встретиться в дальнейшем. Наверно имеется ввиду сбор всех решений, чтобы исключать повторения и не выполнять лишних просчетов.
* Программирование снизу. Заключается в переформулировании исходной задачи в виде рекурсивной последовательности более простых. Решаем простые и поднимаемся к более сложной.

1. **Типовые алгоритмические идеи. Эвристические алгоритмы. Предпросчет.**

**Эвристические алгоритмы:**

*Эвристический алгоритм - алгоритм решения задачи, основанный на некотором предположении. Он не является гарантированно точным или правильным, но его достаточно для решения поставленной задачи*.

Например, *правило Варнсдорфа*. Это правило для того, чтобы обойти все клетки шахматной доски конем, в каждой побывав один раз. Правило в том, что надо ходить в ту ячейку, из которой меньше всего продолжений.

**Предпросчет:**

*Просчитать все варианты на всех входных параметрах.*

Например, кубик Рубика.

1. **Типовые алгоритмические идеи. Приближенные алгоритмы. Структуры данных на основе хеш-функций.**

**Приближенное решение:**

*Приближенное решение - способ поиска не точного правильного решения, а близкого к правильному.*

Например, решение задачи может занимать довольно большое время. Но нам не нужно получать идеально верное решение. Нам может хватить неточного решения с некоторой погрешностью, но которое будет вычислено быстрее.

**Структуры данных на основе хеш-функций:**

*Хеш-функция* ***–*** *функция, осуществляющая преобразование массива входных данных произвольной длины в (выходную) битовую строку установленной длины.*

(Картинки→числа, сравнение не по пикселям а по числам).

Сопоставление объектам одной природы значения объектов другой природы называется *хешированием*.

Структуры данных на основе хэш-функций - различные хэш-таблицы, словари. Тогда по хэшу мы можем из таблицы получить нужное нам значение, если конечно не возникнет коллизии.

1. **Логика и исчисление высказываний. Формальное определение.**

*Исчисление относится к абстракции, логика привязано к конкретной ситуации.*

**Исчисление высказываний** *изучает сложные высказывания, образованные из простых, и их взаимоотношения. В отличие от логики предикатов, она не рассматривает внутреннюю структуру простых высказываний, она лишь учитывает, с помощью каких союзов и в каком порядке простые высказывания сочленяются в сложные.*

**Условия, определяющие исчисление:**

* Имеется **алфавит** исчисления, элементами которого называются символами. Конечная последовательность символов - слово. W - множество слов
* Задано подмножество - **множество выражений** исчисления. Это формулы.
* Выделено множество Ах входящее в W. Это **аксиомы**.
* Множество R = {R1, R2, …} - **правила вывода** исчисления

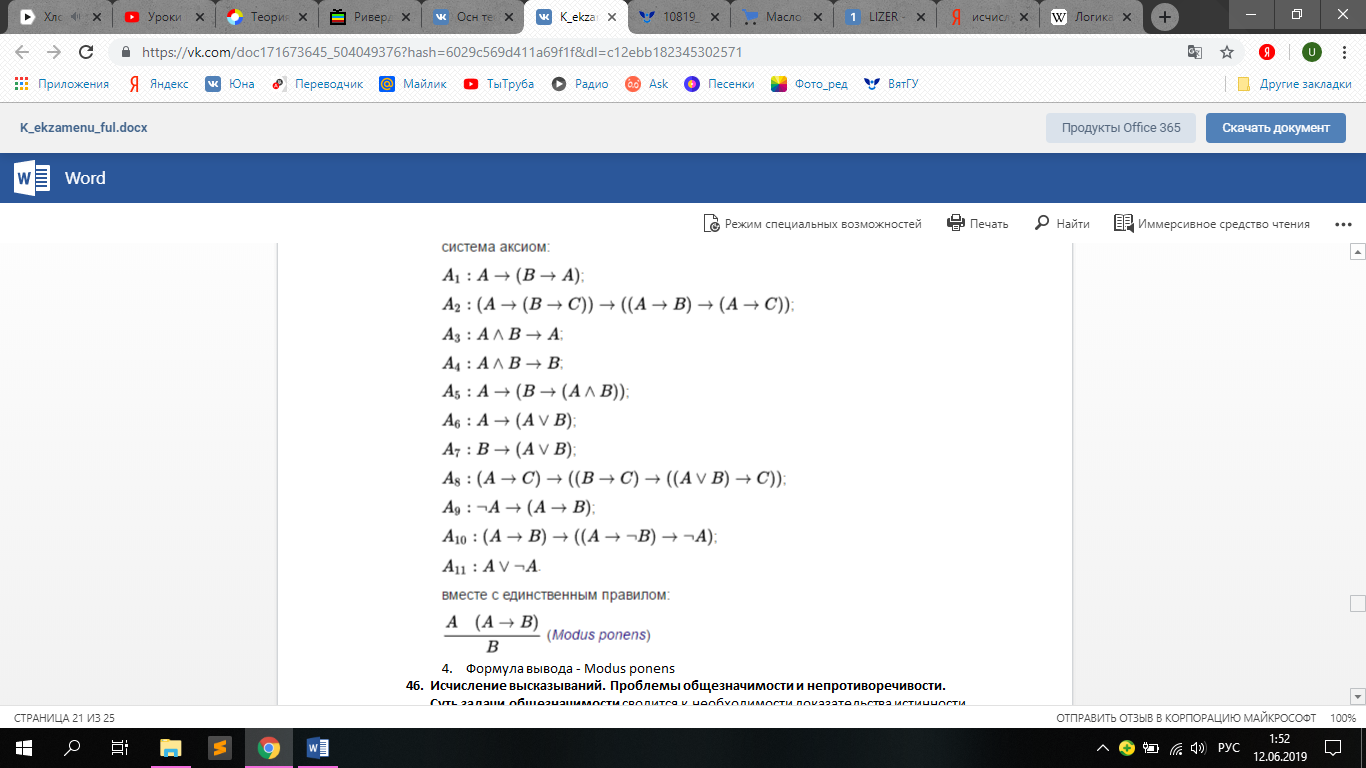
Исчисление высказываний статично. Позволяет описать статичные ситуации. Высказывания - исчисление состояний.

Базис позволяет определиться с алфавитом. Чтобы знать с какими знаками работаем.

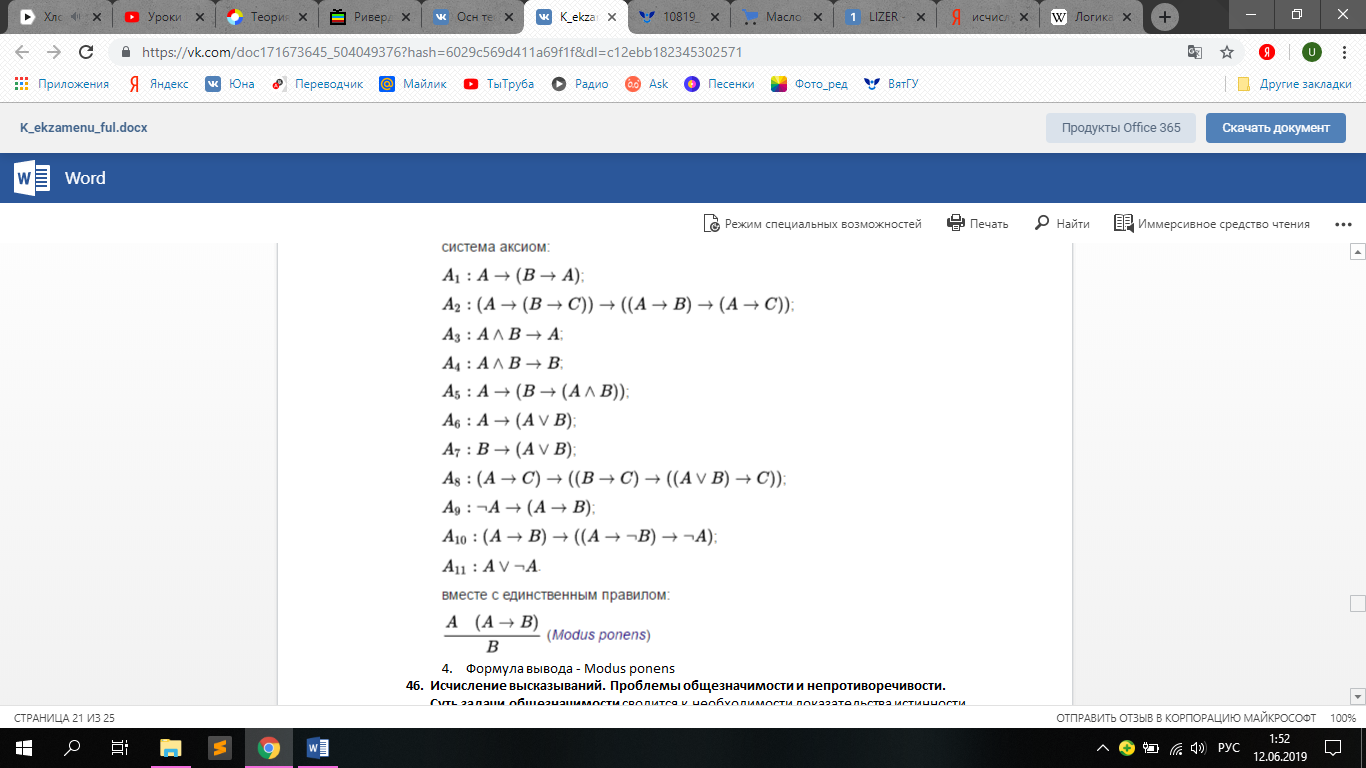
Переменные пропозициональные потому что они являются некоторыми утверждениями

**Исчисление Гильбертовского типа**.

* **Алфавит**. Пропозициональные переменные, логические знаки И, ИЛИ, НЕ, ИМПЛИКАЦИЯ, Строгая дизъюнкция, Эквивалентность.
* **Формулы**.
  + Отдельно стоящая пропозициональная переменная – формула
  + Если А - формула, то НЕ(А) - тоже формула
  + Если А и В произвольные формулы, то (A -> B), (A V B), (A˄B) – тоже формулы
* **Аксиомы**



* **Правило вывода**.



1. **Исчисление высказываний. Проблемы общезначимости и непротиворечивости.**

*Суть задачи* **общезначимости** *сводится к необходимости доказательства истинности некоторого заданного утверждения при любом наборе значений переменных*

*Задача* **непротиворечивости** *- необходимость установления наличия противоречия в наборе некоторых заданных логических утверждений*

1. **Проверка общезначимости в исчислении высказываний. Алгоритм Квайна. Пример.**

**Алгоритм Квайна** *позволяет определять общезначимость формулы. Суть алгоритма в построении дерева. Ребро помечается литерой. Литеры из одной вершины контрарны. И соответствуют одной переменной если находятся на одном расстоянии от корня.*

Перебираем все наборы переменных и, если на всех наборах истина - она *общезначима*. Если хотя бы где-то ложь - то не общезначима. Алгоритм Квайна позволяет сократить процесс перебора используя дерево.

1. **Проверка общезначимости в исчислении высказываний. Алгоритм редукции. Пример.**

**Алгоритм редукции** *применим в случаях, когда у нас много импликаций. Выполняется попытка нахождения значений переменных, при которых обращается в ноль. Импликация - ложь, когда посылка истина, а заключение ложь.*

1. **Задача логического вывода. Подходы к решению.**

**Задача логического вывода** *состоит в получении новых знаний из уже имеющихся знаний. Получение новых формул из аксиом и формул, полученных ранее.*

Подходы:

* **Метод резолюций**. Базовая операция - резолюция. Известно, что оба дизъюнкта истинны. Можно построить **резольвенту дизъюнктов**. При этом резольвента так же будет истинна.
* **Метод резолюций для хорновских дизъюнктов**
* **Метод деления дизъюнктов**

1. **Исчисление высказываний. Метод резолюции.**

*Базовая операция - резолюция*.

D1 = D1’ ˅ A D2 = D2’ ˅ ¬A

*res (D1, D2) = D1’ ˅ D2’*

**Теорема**. *Множество дизъюнктов противоречиво только тогда, когда существует резолютивный вывод, заканчивающийся нулем.*

Метод резолюций состоит в доказательстве целевого утверждения через его отрицание. Если доказали противоречивость от отрицания целевого, значит целевое истинно. При выводе используются различные стратегии. Стратегия вывода - последовательность построения резольвент.

1. **Исчисление высказываний. Метод резолюции для хорновских дизъюнктов.**

Данный метод резолюций применяется для Хорновских дизъюнктов.

**Хорновский дизъюнкт** - *дизъюнкт, содержащий не более одного положительного дизъюнкта.*

{¬A1 ˅ ¬A2 ˅ … ˅ ¬An ˅ B – точный;

¬A1 ˅ ¬A2 ˅ … ˅ ¬An – негативный;

B – унитарно позитивный;}

Строим таблицу. Берем унитарный позитивный дизъюнкт. Подставляем его в выражение, содержащее резольвентую пару. Продолжаем так делать пока не получим ноль. Если получили ноль - противоречие. Если ничего не получили, значит исходное множество непротиворечиво.

1. **Исчисление высказываний. Метод деления дизъюнктов.**

*Базовая операция - операция деления дизъюнктов (D % d).*

Результаты деления:

* **Единица**, если не имеют общих литералов
* **Ноль**, если D совпадает с d или является его частью
* **Остаток от деления**. Элементы d вычеркиваются из D

Семантический смысл. Допустим D истина. Доказываем истинность d. Если общих литералов не имеют, то мы ничего не можем сказать про истинность исходя из D. Если D совпадает с d или часть его. Тогда истина. Если есть пересечение, то d будет истинно, когда будет ложным остаток от деления так как он входит в d. А если вхождение будет истинным, то d - истина.

**Процедура деления дизъюнктов:**

{V=<M, d, q, M1, m>

M - исходные секвенции

d - то, что мы доказываем

q - признак окончания. 0 - вывод успешен, 1 - неудачно, G - требуется продолжение вывода

M1 - новое множество исходных секвенций

m - новое множество дизъюнктов выводимых секвенций}

Алгоритм выполнения процедуры:

* Все дизъюнкты исходной системы делим на выводимый. Формируем список остатков.
* Составляем секвенцию из конъюнкции этих остатков и анализируется. B1^B2->0
  + Если все остатки равны единице. То получили ложь. q=1 и переход к пятому шагу.
  + Если при делении хотя бы один даст нам ноль, то q=0. В противном случае преобразуем остатки через конъюнкцию в новый набор. Описан в шаге 3.
  + Если ничего не получили, то переходим к шагу 3
* Получаем дизъюнкцию конъюнкций. Устанавливаем q=G. Формируем новое множество выводимых из этих дизъюнкций
* Формируем M1 исходных секвенций. M1=M-M0 где М0 подмножество дизъюнктов, для которых были получены остатки не равные 1. Вычеркиваем все те, где было не равно 1.

**Метод деления дизъюнктов.**

Последовательное применение процедуры. На каждом шаге для каждого выводимого исходное делим на выводимый. Формируем b и так далее.

1. **Формальное определение исчисления предикатов первого порядка.**

**Предикат** – это логическая функция, задающая отношение между константами и переменными. Предикат возвращает либо «истину», либо «ложь».

**Исчисление предикатов первого порядка** – *формальное исчисление, допускающее высказывания относительно переменных, фиксированных функций и предикатов. Расширяет логику высказываний. В свою очередь является частным случаем логики высших порядков.*

Формально определим исчисление предикатов:

* **Алфавит**. Исходные символы исчисления
* Предметные **переменные** {х1, x2, …}
* **Предметные** **константы** {а1, a2, …}
* **Функциональные** **константы** (всякие плюсы минусы). Некая таблица подстановок.
* **Предикатные** **константы** (сравнения, отношения)
* **Символы** **исчисления**. Истина, ложь, дизъюнкция, конъюнкция, отрицание, следование
* **Кванторы**
* Формулы
  + Всякая предметная константа - терм - некоторое выражение, значение которого объект
  + всякая свободная переменная – терм
  + функциональная константа от термов – терм
  + то, что получаем из предыдущих пунктов
  + Если P - предикатная константа. То выражение Р (термы) - атомарная формула. Атомарная формула - отношение между термами.
  + Отдельно стоящая атомарная формула или константа - формула.
  + Если А и В формулы, то Not(A), (A\*B), (A V B) - формулы
* **Аксиомы**
* **Правило** **вывода**.

Свойства системы исчисления предикатов

* Исчисление предикатов первого порядка непротиворечиво, т.е. из аксиом нельзя доказать выводимость теоремы и не теоремы.
* Всякая теорема является общезначимой формулой.
* Всякая общезначимая формула является теоремой (см. разделы «Истинность формальной системы» и «Классы формул»).

1. **Преобразование выражений исчисления предикатов первого порядка в конъюнктивную нормальную форму. Пример.**

Для перевода в КНФ выносим кванторы в левую часть, после применяем правила булевой алгебры для перевода в КНФ.

1. **Сколемовское преобразование. Операции унификации.**

Выносим все кванторы. Убираем кванторы существования, заменяем переменные на термы от предшествующих переменных.

**Операция унификации**. Допустим у нас есть формулы. Надо проверить их на равенство. Для этого берем переменные и пытаемся изменить их значение, чтобы добиться равенства. Процедура обнаружения одинаковых литералов.

1. **Модальные логики. Области применения. Особенности. Примеры.**

**Модальная логика** - раздел логики, в котором изучаются логические связи модальных высказываний, те, которые включают модальность. Модальность - модальный оператор.

Проще говоря. Модальная логика - расширение исчисления высказываний.

Добавляются дополнительные операторы, позволяя расширить исчисление.

Модальности:

* **Алетическая** (необходимо, возможно, случайно)
* **Деонтические** (обязательно, разрешено, запрещено)
* **Аксиологические** (хорошо, плохо, нейтрально)
* **Темпоральная** (в будущем. в прошлом, сейчас)
* **Пространственные** (там, здесь, нигде)

Модальные операторы делают описания проще, позволяя описывать некоторые ситуации.

1. **Темпоральная логика линейного времени. Формальное определение. Модальные операторы.**

*Темпоральные логики используются для описания временного фактора в рассуждениях.*

Модальности:

* **Next (X)** - выражение истинно в текущем состоянии если p будет истинно в следующий момент времени
* **Future (F)** - выражение истинно сейчас, если в будущем p станет истинно
* **Globally (G)** - выражение истинно сейчас если в будущем p всегда будет истинно
* **Until (U)** - выражение pUq истинно в текущем состоянии если в будущем q станет истинным, а до тех пор истинным будет p
* **Weak until (W)** - выражение pWq истинно в текущем состоянии если когда-нибудь в будущем К станет истинно, а до тех пор p будет истинно, или q будет истинно всегда.
* **Release (R) -** pRq истинно в текущем состоянии если в будущем p станет истинно, а до того времени включительно истинно будет q или q всегда будет истинно

1. **Темпоральная логика ветвящегося времени. Формальное определение. Модальные операторы.**

Модальности:

{X, F, G, U}

+

* **Along all part (A)** - истинно в текущем состоянии если будет на всех путях от текущего состояния истинно будет фи.
* **Aong at least one part (E)** - если фи истинно хотя бы на одном пути.

Время представляется как ветвистая структура.

**Отличия логик.** Логика линейного времени логика путей. Логика ветвящегося времени - это логика состояний.