Математическая логика и теория алгоритмов

Лектор: А. Л. Семенов

Лекция 1

Оглавление

Введение. Основная задача математической логики и теории алгоритмов

Наша задача – построить систему математических определений и теорем, позволяющих математически исследовать следующие виды математической деятельности человека:

1. Доказательство теорем и определение понятий
2. Описание отношений между математическими объектами
3. Описание и применение алгоритмов

Конечно, принципиальную роль здесь играют математические языки. Мы будем постоянно к ним обращаться.

Занимаясь указанным исследованием, мы получим и результаты, относящиеся к:

1. Множествам, которые можно описать в каких-то формальных системах
2. Множествам доказуемых формул
3. Множествам истинных формул
4. Множествам математических структур, в которых истинны формулы из заданного множества
5. Классов всех отношений, задаваемых в том или ином языке
6. Классов функций, которые вычисляются алгоритмами
7. Существованию алгоритма, выясняющего доказуемость формул

   и т. д.

   Кроме того, как показывает история развития информационных и коммуникационных технологий, все это имеет к ним непосредственное отношение. При совсем общем взгляде можно сказать, что математическая

логика дает фундамент для теоретической математики, а теория алгоритмов – для вычислительной практики (использования компьютеров). Более детальное рассмотрение показывает, что многие достижения математической логики нашли приложения в разработке и применении информационных технологий, а алгоритмические рассмотрения существенны в различных разделах чистой математики.

Важными моментами в становлении современной математической логики сыграли:

- Готлоб Фреге (8.11.1848, — 26.07.1925) – анализ математических языков и смыслов, логические исчисления
- Георг Кантор (3.03.1845 — 6.01.1918) – построение теории множеств – фундамента современной математики
- Давид Гильберт (23.01.1862 — 14.01.1943) – представление о математике, как математически формализуемой деятельности, осознание важности исследований в рассматриваемой нами области для математики, представление программы таких исследований – Программы Гильберта.

  Особую важность для математиков имело выступление Д. Гильберта на II Международном конгрессе математиков. (Париж, 1900). Там он сформулировал 23 т. н. Проблемы Гильберта – важнейших для математики того времени и для развития математики в XX в. Проблемы I, II, X из списка Гильберта относятся к предмету математической логики и теории алгоритмов.

- Эрнст Цермело (27.7.1871 □ 21.5.1953) – формулировка математики как формальной системы – теории множеств.
- Аксель Туэ (19.2.1863 □ 7.3.1922) – введение системы математических понятий, относящихся к исчислениям
- Эмиль Пост (11.02.1897 — 21.04.1954) – представление об общей природе исчислений и универсальной модели для множеств, породимых исчислениями,
- Курт Гедель (28.04.1906 – 14.01.1978) реализация возможного и выявление невозможного в Программе Гильберта.

Общее понятие алгоритма и варианты универсальной модели, описывающей все вычислимые функции, сформировалось к середине 1930-ых гг. в работах Эмиля Поста, Алана Тьюринга (23.06.1912 – 7.06.1954), Курта Геделя.

Гедель и Тьюринг – единственные математики, вошедшие в список ста наиболее влиятельных людей 20-го века по мнению американского журнала Time.

Из семи математических Проблем тысячелетия первая также относится к нашему предмету (ее не было среди Проблем Гильберта). В курсе будут обсуждаться все эти проблемы и что с ними произошло.

Метафорическим отражением научных достижений, о которых мы говорим, можно считать роман Германа Гессе Игра в бисер (Das Glasperlenspiel. 1943). Вот цитата оттуда:

Ты пишешь на листе, и смысл, означен
И закреплен блужданьями пера,
Для сведущего до конца прозрачен:
На правилах покоится игра.
Алфавит. Собственные сочинения Иозефа Кнехта, перевод С. Аверинцева

Базовые понятия

Мы считаем известными базовые математические понятия, например, понятие множества, обычных операций на множествах (объединение, пересечение, прямое произведение, проекция и т. п.), функции и т. п.

В частности, отношения (многоместные) на множестве – это подмножества декартовой степени этого множества. Функции могут рассматриваться как специальные виды отношений.

Мы также используем:

Логические значения: символы И, Л, или символы 0, 1.

Логические операции: **&** (и, конъюнкция), ∨ (или, дизъюнкция), ¬ (не, отрицание) применяются к символам 0 (И) и 1 (Л) и результат их применения описан следующей таблицей:

A	B	¬A	A&B	A∨B
0	0	1	0	0
0	1	1	0	1
1	0	0	0	1
1	1	0	1	1

Среди функций мы выделяем свойства – функции, принимающие только значения И и Л (0 и 1). Всякое свойство задает отношение – множество элементов, на которых ее значение – И. Всюду определенные свойства называются также характеристическими функциями.

Цепочка - конечная последовательность, в частности, последовательность может быть и пустой – Л. Для цепочек мы будем использовать обозначение $< a_0,\ldots a_n >/$

Длина цепочки – это число ее элементов.

Алфавит - это конечное множество символов

Слово (в данном алфавите) – цепочка, элементы которой взяты из алфавита. При записи слов запятые и угловые скобки опускаются.

Ансамбль конструктивных объектов – более общее понятие, чем множество всех слов в данном алфавите, например, можно дать определение ансамбля размеченных графов.

Мы будем ограничиваться словами и говорить об ансамбле слов в данном алфавите, имея в виду множество всех слов в этом алфавите. Говоря «ансамбль слов» мы будем подразумевать «ансамбль слов в данном алфавите». Алфавитом часто будет 01 – множество из двух символов: 0 и 1.

Нас будут интересовать функции, аргументы которых – это слова в каком-то алфавите. Значениями функций будут тоже слова, возможно, в другом алфавите. (Как уже было сказано, можно рассматривать и другие ансамбли конструктивных объектов, но мы этого делать не будем.)

Цепочку слов можно считать тоже словом в другом алфавите, например, в алфавите, расширенном символом «,» (если раньше в алфавите запятой не было) добавляя его после каждого элемента цепочки. Тогда пустая цепочка - это пустое слово, а цепочка из одного пустого слова, это слово из одного символа ,.

Кодирование

Можно цепочки слов кодировать словами в исходном алфавите.

Пример:

Код цепочки слов в алфавите 01 можно получить так:

- Нужно удвоить каждую букву 0 или 1, а запятую заменить на 01

Таким образом, мы задали инъективную функцию из ансамбле слов а алфавите трех символов 01, в ансамбль слов в алфавите 01, разные слова получили разные коды. Длина кода оказалась двое больше суммы длин слов исходной цепочки, считая еще и запятые.

Задача, к решению которой мы вернемся: *можно ли кодировать покороче?*

Использование кодирования позволяет нам вместо многоместных функций (функций от нескольких аргументов) и отношений использовать одноместные, но примененные к кодам цепочек. Например, вместо трехместной функции можно рассматривать одноместную функцию, применяемую к словам в том же алфавите, являющимся кодам цепочек длины три.

Принципиально важным для нас является то, что рассматриваемые функции могут иметь значение (быть определенными) не на всем ансамбле слов.

Описания. Действия и проверки

Особую важность для нас имеют функции, имеющие описание (в подходящем языке) и позволяющее (хотя бы в принципе), по аргументу (исходному данному) функции всегда найти значение (результат применения) функции.

Действие – исходное понятие, не имеющее точного математического определения через другие понятия, обладающее следующими свойствами. Действие:

- описано на понятном человеку языке, может осуществляться и человеком и каким-то (реальным или абстрактным) устройством,
- можно применить к любому исходному данному из фиксированного ансамбля исходных данных (у нас это ансамбль слов), при этом ясно, что всегда получается

результат применения – элемент (возможно, другого) фиксированного ансамбля (в нашем случае – слов).

Кодирование – пример действия.

Трудность выполнения действия (человеком или устройством) может быть большой, например, если велик объект, по отношению к которому действие нужно применить. Понятно, как возвести 2 в натуральную степень, но если показатель степени – это число из миллиарда знаков, то получение результата может столкнуться с техническими трудностями и потребовать много времени, хотя в принципе, как это делать – понятно, и мы справедливо считаем, что действие возведения в степень всегда заканчивается. Более того, даже при небольшом размере исходных данных требуемое действие может быть практически невыполнимым: известный пример – Ханойская башня – головоломка, где понятное задание при вполне обозримом исходном данном требует времени, соизмеримого со временем существования Вселенной.

Как писал Эмиль Борель в 1912 году: «Я намеренно оставляю в стороне большую или меньшую практическую длительность; суть здесь та, что каждая из этих операций осуществима в конечное время при помощи достоверного и недвусмысленного метода»

Проверка – это действие, всегда завершающееся результатом И (1) или Л (0).

Таким образом, действие задает всюду (то есть, на всем рассматриваемом ансамбле) определенную функцию, проверка – всюду определенное свойство – характеристическую функцию.

Исчисления. Породимые множества

Исчисление – это пара из двух проверок:

<правило создания, правило окончания>.

В этом определении мы считаем, что проверка «правило создания» применяется к коду цепочки слов, а правило окончания – к слову.

Пусть задано некоторое исчисление. Тогда множество создаваемых (этим исчислением) объектов определяется так:

- Если для цепочки объектов $a_0,\ldots a_n$ – выполнено (истинно) правило создания и все элементы этой цепочки, кроме последнего – создаваемы, то и последний элемент создаваем.

В частности, если в цепочке есть только один элемент и для нее выполнено правило создания, то этот элемент называют начальным объектом. Конечно, если таких объектов у данного исчисления нет, то множество создаваемых им объектов пусто.

Множество объектов, порождаемых данным исчислением, состоит из всех создаваемых объектов, для которых выполнено правило окончания (этого исчисления).

Заметим, что правило окончания может быть истинным не только для создаваемых объектов и что начальные объекты не обязаны ему удовлетворять.

Связанное с понятием исчисления и производными от него понятиями интуитивное представление таково. В процессе построения (создания) какой-то формальной системы, например, математической теории, человек исходит из какого-то запаса *начальных объектов*, в случае теории – аксиом, и строит новые, комбинируя уже полученные. Возможность получения нового, желательного объекта (возможно, подсказываемого человеку неформализованной интуицией) регулируется *правилом создания*. Можно воспользоваться аналогией с какой-то игрой, например, шахматной: правило создания (то есть – все описание шахматной игры) устанавливает, какие позиции из каких могут быть получены за один ход. При этом некоторые из создаваемых по ходу дела

объектов имеют вспомогательный характер, их, например, вовсе нельзя считать теоремами (или даже – утверждениями) теории. Однако теоремы теории легко выделяются некоторой формальной проверкой – *правилом окончания*.

Беря всевозможные исчисления, получаем класс породимых множеств. Далее мы будем считать, что все породимые множества – это множества слов в алфавите 01, хотя, например, начальные объекты могут быть словами в более обширном алфавите.

Замечания.

1. Можно считать, что правило создания – это отношение между конечными подмножествами ансамбля. Действительно, отношение создаваемости не зависит от порядка объектов, кроме последнего; создаваемость конечного множества – это просто создаваемость всех его элементов.

2. Определение создаваемости есть некоторый вариант определения *замыкания* отношения п.2. При этом мы ограничили рассматриваемые отношения проверками.

3. Если для $a_0,\dots a_n$ – выполнено (истинно) правило создания, можно говорить что a_n *создается*, или *выводится из* $a_0,\dots a_{n-1}$

4. Можно говорить о *выводе* объекта а, как о последовательности объектов, каждый из которых выводится из подмножества предшествующих ему в выводе. Объект создаваем тогда и только тогда, когда у него имеется вывод.

Теоремы замкнутости для исчислений

Теоремы замкнутости для исчислений. Объединение, пересечение и проекция породимых множеств породимы.

Доказательство. Рассмотрим, например, случай пересечения. Пусть множество А порождается исчислением <Правило создания А, Правило окончания А>, а множество Б порождается

исчислением <Правило создания Б, Правило окончания Б>. Можно было бы создавать все, что создается по правилу А и все, что создается по правилу Б, а потом отобрать то, что породилось и по тому и по другому правилу. Здесь, однако, требуется аккуратность. Мы, конечно, не можем просто использовать и то и другое правило вперемешку.

Нам нужно как-то пометить все объекты создаваемые по правилу А и все объекты создаваемые по правилу Б. Проще всего это сделать, создавая пары вида <А,х> <Б, у>. Это можно сделать, просто приписав ко всем элементам каждой цепочки, входящей в Правила создания А и Б, в качестве дополнительной первой компоненты символы А и Б, соответственно. Для этого можно расширить алфавит двумя буквами А и Б и запятой. Полученные правила уже можно безопасно объединить.

Далее нам понадобится расширить полученное (объединенное) правило еще средством, удаляющим, когда нужно, пометки, то есть тройками вида <<А,х>, <Б,х>,х>. Таким образом мы обеспечиваем создание тех и только тех слов в основном (не расширенном) алфавите, которые создаются и в исчислении А и в исчислении Б.

Что же мы можем взять в качестве правила окончания? Конечно – конъюнкцию Правил окончания А и Б.

Заметим, что среди формулировок теорем замкнутости для исчислений нет теоремы о породимости дополнения. К этому имеются весьма серьезные причины, которые мы обсудим в дальнейшем. Сейчас отметим два обстоятельства:

1. В самом определении говорится, как создавать новые объекты, и никак не видно, как можно установить не-создаваемость.
2. Математик, пытаясь понять, что что-то нельзя доказать, часто формулирует отрицание этого утверждения и пытается доказать его. Иногда он берет даже более сильное

негативное утверждение, например, конструирует конкретный объект, и пытается доказать что это – контрпимер для предполагаемой теоремы. Всегда ли можно доказать или утверждение или его отрицание? Вот один из вопросов, на которые математическая логика пытается дать ответ.

3. Конечно, для некоторых исчислений сама постановка вопроса об отрицании бессмысленна и тем менее понятно, как устанавливать не-выводимость

Операции над функциями и их описаниями

Выше мы говорили о том, что действия и проверки задаются своими описаниями. Описаниями можно и более сложные функции, в частности, получаемые некоторыми операциями из действий и проверок или уже описанных функций. Далее мы будем определять операцию над функциями одновременно с объяснением того, как мы поступаем с описаниями функций, если функции заданы своими описаниями, однако определения операций имеют смысл и для функций безотносительно к их описаниям.

Ячейки

Действие у нас работает с объектом целиком. Описывая действия или какие-нибудь еще функции бывает удобно считать, что объект разделен на части, каждая из которых лежит в своей ячейке, ячейка имеет отдельное имя, и действие меняет иногда содержимое одной ячейки, иногда – другой. (Если наши объекты слова, то это можно делать, используя цепочки слов и кодируя их словами.) Значение имени (содержимое ячейки) можно изменить, в частности, с помощью действия присваивания. Это действие мы обозначаем ←. Кроме обрабатываемого объекта бывает удобно использовать еще метки из подходящего конечного алфавита. Их тоже можно хранить в ячейках с удобными именами.

График

Пусть Г – описание функции. Тогда Г – описание функции, которая перерабатывает объект x в пару <x, Г(x)>. Можно считать, что элементы этой пары лежат в ячейках с именами: «аргумент Г» и «значение Г». Г может сначала сделать из x пару <x,x> (скопировать x), а потом применить Г ко второму элементу пары.

Композиция (последовательное выполнение)

Пусть Г, Д – описание функций, тогда

- Г
- Д

тоже является описанием функции: сначала применяем функцию, описанную Г, потом к результату – функцию, описанную Д. Если Г, Д описывают действия, мы получаем описание действия.

Ветвление

Пусть П – описание свойства, Г описание функции, тогда

- Если П
  - Г

тоже является описанием функции. Если значение свойства П для объекта – это И, то применяем Г, если значение свойства – Л, то результат применения функции совпадает с исходным данным, если свойство для объекта не определено, то и описанная функция на этом объекте не определена.

При этом, если П – проверка, а Г – действие, то мы получаем описание действия. Описывая это действие более детально, мы можем сначала перейти от проверки П к действию П, то есть сохранить аргумент, он нам понадобиться, если П даст значение Л.

Повторение (итерация, цикл)

Пусть П – описание свойства, Г – описание функции, тогда

- Пока П
  - Г

тоже является описанием функции. Если на исходном данном свойство П – Л, то результат применения функции совпадает с исходным данным. Если значение П – И, то нужно примерить Г, снова проверить П, если П даст значение Л, то остановиться и т. д. Если в какой-то момент П или Г не дает значения, то и описываемая функция – не определена. Если П никогда не дает значения Л, то описываемая функция тоже не определена.

И здесь, описывая полученное действие более детально, мы сначала перейдем от проверки П к действию П, то есть сохраним аргумент. При каждом применении Г мы должны будем этот аргумент заменить на результат применения действия к нему, после этого к этому аргументу применить П и результат применения поместить в ячейку результат П и т.д.

Если П – описание проверки, а Г – описание действия, то операция повторения не обязательно дает описание действия: работа по этому описанию может на завершиться. Получаемая функция может быть не всюду определенной. Однако, если в этом случае мы дополнительно знаем, что работа всегда завершается, то мы имеем описание действия.

Логические операции

Чтобы определить, как выполняются логические операции в применении к свойствам (а не только к проверкам), нам нужно договориться о том, какое значение имеет комбинация двух свойств, когда одно из них – не определено. Один из естественных способов такой договоренности состоит в

следующем: если имеющихся логических значений элементов комбинации достаточно, чтобы однозначно указать значение комбинации, то мы его и берем за значение комбинации. Если имеющихся значений недостаточно для этого, то значения комбинации – нет. Другими словами, отрицание истины – ложь, лжи – истина, неопределенности – неопределенность (то есть, если где-то свойство не определено, то и его отрицание – не определено). Если в конъюнкции один из членов ложен, то и вся конъюнкция ложна, если оба члена истинны, то – истинна, в остальных случаях – не определена. Если в дизъюнкции один из членов – И, то дизъюнкция – И, если оба – Л, то и дизъюнкция – Л, в остальных случаях – не определена.

Если П и С – описания свойств, то П и С, П или С, не П – описания свойств. В простейшем случае, когда П и С – проверки, указанные комбинации свойств тоже являются проверками и их значение определяется приведенной выше таблицей в применении к значениям функций.

Алгоритмы. Вычислимые функции

Понятие алгоритма, как и понятие действия, также может считаться исходным и неопределяемым. Однако пытаясь установить некоторые общие факты, относительно функций, осуществляемых алгоритмами, мы сталкиваемся с необходимостью использовать некоторые свойства алгоритмов. Одним из способов фиксации этих свойств является следующее определение.

Алгоритм – описание вида

- А
- Пока Б
    - В
- Г

Где А, В, Г – описания действий, Б – описание проверки, они называются:

- А - начало
- Б - продолжение
- В - переработка
- Г - извлечение результата.

Мы считаем, что и исходные данные и результаты – это слова в некоторых алфавитах, обычно, в алфавитах 01. Промежуточные результаты – также слова в некотором – третьем, алфавите. Естественно считать, что третий алфавит содержит первый и второй. Таким образом, алгоритм задает некоторую функцию из множества всех слов в алфавите исходных данных во все слова в алфавите результатов. Мы говорим, что рассматриваемый алгоритм вычисляет эту функцию.

Интуитивные соображения, лежащие в основе этого определения, таковы. Работа по алгоритму состоит в многократном применении некоторой инструкции, описывающей действие переработки. При этом мы должны вовремя остановиться, это можно обеспечить за счет проверки продолжения (не исключена возможность, что остановиться придется сразу). Начало работы может потребовать некоторой подготовки исходного данного, «ввода» в «вычислительную среду», или «устройство», добавления к нему «служебной информации», это обеспечивается действием начала. Соответственно после завершения переработки может требоваться дополнительное действие извлечения результата, «вывода».

Заметим, что, поскольку в описании алгоритма участвует проверка, то при его выполнении естественно иметь две ячейки. В одной хранится перерабатываемый объект, а в другую помещается результат проверки.

Интуитивная «алгоритмичность» предлагаемого описания не вызывает сомнений. Вопрос в том, охватывает ли наше определение все мыслимые, в принципе возможные, алгоритмы. Это – очень важный вопрос и мы к нему еще вернемся.

Вычислимая функция – это функция, вычисляемая некоторым алгоритмом. Вычислимое свойство – это вычислимая функция со значениями только И или Л. Вычислимое свойство называется разрешимым, если его функция всюду определена.

Итак, вычислимая функция и вычислимое свойство могут быть не всюду определенными. Причина этому в том, что для каких-то исходных данных процесс применения алгоритма может не завершаться. Хотя каждый отдельный шаг из списка А – Г завершается, но проверка продолжения Б может никогда не дать ответа Л и действия переработки В будут все идти и идти…

Наши определения породимого множества и вычислимой функции (при данном алфавите) зависят от исходного запаса действий (в частности – проверок). Однако оказывается, что взяв в качестве такого запаса некоторые действия, про которые не возникает сомнения в их принципиальной осуществимости (и завершаемости за конечное время), мы получаем класс вычислимых функций, который не удается расширить при самых различных дальнейших модификациях запаса действий. Такой «небольшой» запас мы скоро предъявим. Аналогичная ситуация имеет место и с породимыми множествами. Более точно мы это сформулируем позднее.

Теоремы замкнутости для вычислимых функций

Теорема. Операции композиции, итерации, ветвления в применении к вычислимым функциям и вычислимым проверкам дают вычислимые. Операции логики в применениям к вычислимым свойствам дают вычислимые, к разрешимым свойствам дают разрешимые

Доказательство. Как видно из нашего определения алгоритма, все, что им перерабатывается, рассматривается как единый объект (например, вся разнообразная память компьютера со всем содержимым), к которому применяется все время (кроме начала и завершения) одно и то же действие. Проверка окончания – тоже одна. Теперь у нас появляется несколько

разных действий (в том числе – проверок) и надо их сочетания уложить в общую схему. Чтобы это сделать, мы будем исходить из интуитивных представлений о том, как организовал бы свою деятельность человек и использовать ячейки.

Композиция

Пусть у нас есть алгоритмы вычисления функций f и g. Как нам вычислять g(f(x))? Ясное дело, надо сначала вычислить f, а потом к результату применить g. Как эту простую идею вложить в нашу формальную схему? Иными словами, пусть алгоритм для f использует начало f, переработку f, проверку окончания f, извлечения результата f, и аналогично – алгоритм для g. Как сконструировать действие переработки и соответствующие проверки так, чтобы сначала шла переработка в соответствии с алгоритмом для f, а потом, в соответствии с алгоритмом для g?

Чтобы разделить два этапа деятельности строящегося алгоритма, введем метки первого и второго этапа, соответственно – символы f и g. Эти метки будем помещать в ячейку, которую назовем Этап.

В дальнейшем описании алгоритма и других, когда мы пишем, что применяется какое-то действие f или действие g, то мы имеем в виду, что оно применяется к соответствующей ячейке перерабатываемого объекта, а содержимое остальных ячеек не меняется. В данном случае f и g действуют на одну и ту же ячейку. Мы также будем считать, что в ячейку с именем Продолжение мы будем помещать значение проверки продолжения описываемого алгоритма, соответственно используются имя Продолжение f и т. д.

Отдельные компоненты строящегося описания мы выделяем заголовками (комментариями).

Итак, пусть вычислимая функция f задана алгоритмом:

Начало

- Начало f

Продолжение

- пока Продолжение f

Переработка

  - Переработка f

Извлечение результата

- Извлечение результата f

Вычислимая функция g задана алгоритмом

Начало

- Начало g

Продолжение

- пока Продолжение g

Переработка

  - Переработка g

Извлечение результата

- Извлечение результата g

Вот как можно описать композицию двух функций

Начало

- Начало f
- Этап ← f

Продолжение

- Если Этап = f

  - Продолжение ← Продолжение f

        - Если не Продолжение
            - Извлечение результата f
            - Начало g
            - Этап← g
    - Если Этап = g
        - Продолжение ← Продолжение g

Переработка

- Если Этап = g
    - Переработка g
- Если Этап = f
    - Переработка f

Извлечение результата

- Извлечение результата g

Ветвление (условный оператор)

Применим операцию ветвления к вычислимому свойству p и вычислимой функции f, заданным своими алгоритмами.

- Если p
    - f

В начале работы мы копируем исходное данное и получаем набор из трех одинаковых элементов. Первый элемент набора (объект) так и будет оставаться неизменным до конца (он нам понадобится на случай, если свойство p окажется ложным), второй элемент (объект f) будет использоваться для вычислений, относящихся к f, третий элемент (объект p) будет использоваться для вычислений, относящихся к p. Кроме того, нам понадобятся еще ячейка, которую мы назовем этап, ее значением будет

символ p или f в зависимости от того, вычисляем ли мы сейчас p или f. Значение проверки продолжения мы будем хранить в ячейке Продолжение.

Описывая работу каждого блока нам надо будут следить, чтобы в нем выполнялось действие или проверка. Детальное «программирование» мы предлагаем в качестве упражнения.

Дизъюнкция. p или q

Мы будем использовать ячейки: Объект p, Объект q, Значение p и Значение q, результат получится в ячейке Результат.

Начало

- Объект p ←Объект, Объект q← Объект
- Значение p ← Н
- Значение q ← Н
- Продолжение ←И

Проверка продолжения

- Если Продолжение p = Λ
  - Извлечение результата p
  - Значение p ← Результат p
- Если Продолжение q = Λ
  - Извлечение результата q
  - Значение q ← Результат q
- Если Значение p = И или Значение q = И
  - Результат ← И
  - Продолжение ←Λ
- Если Значение p = Λ и значение q = Λ

- Результат ← Λ
- Продолжение ←Λ

Действие переработки:

- Если Значение p = H
  - Переработка p
- Если значение q = H
  - Переработка q

Извлечение результата

- Результат

В случаях других логических операций можно действовать аналогично.

Перечислимые множества

Определение. Перечислимое множество – это множеством значений какой-то вычислимой функции.

Доказательство следующих теорем оставляется для упражнений. Как и раньше, его можно провести на интуитивно-содержательном или более формальном, «программном» уровен.

Теорема. Следующие свойства множества эквивалентны:

1. Оно – перечислимо
2. Оно – область определения какой-то вычислимой функции
3. Оно – породимо

Можно указать общие способы (алгоритмы) построения по любому из описаний 1 – 3 любого другого. Перечислимость любого породимого множества особенно замечательна!

При доказательстве нам может пригодиться тот факт, что ансамбль всех слов «перечислим» в следующем смысле: есть действвие S, для которой: множество Λ, S(Λ), S(S(Λ)),.. содержит все элементы ансамбля.

Замечания

Так же легко доказать, что перечислимое множество пусто или является множеством значений всюду определенной вычислимой функции. Однако общего способа, как, имея описание 1 – 3, понять, что множество пусто или построить описание требуемой всюду определенной функции нельзя! (Мы увидим это далее.)

Теорема. Функция вычислима ↔ ее график перечислим

Таким образом, класс вычислимых функций можно определить через класс породимых множеств.

Теорема. Множество разрешимо ↔ оно и его дополнение перечислимы