# Введение в математическую логику

## Лекция 2

### 1. Вычислимые функции. Алгоритмы

На прошлой лекции мы говорили о том, что математическая логика отвечает на два важных вопроса: что такое доказуемое математическое утверждение и что такое вычислимая математическая функция. Что такое доказуемость, в полном объёме сказано не было, но было предложено общее интуитивное понятие исчисления. В частности, было указано на общий класс исчислений, задаваемых грамматиками.

Теперь дадим определение вычислимой функции.

**Определение 1.** Функция называется *вычислимой*, если она вычисляется некоторым *алгоритмом*.

Значит, нам нужно понимать, что такое алгоритм.

Алгоритмы сейчас проходят в школе, но и в школе, и сейчас мы не будем давать математического определения того, что такое алгоритм. Его, по-видимому, и нельзя дать, потому что алгоритм, как и исчисление, относится к понятиям из реального мира, из реальной деятельности человека, даже если она производится изображениями и именами математических объектов.

Но, тем не менее, про алгоритм можно сказать следующее: у алгоритма имеется *описание*, или *программа*. Имеется некоторый способ, как по исходному данному начинается процесс выполнения алгоритма, или вычисления. Выполнение состоит в том, что возникает некоторая последовательность состояний алгоритма, причём каждое следующее состояние в этой последовательности однозначно определяется предыдущим в соответствии с описанием. Но иногда алгоритм заявляет, что следующего состояния нет и он свою работу закончил. Тогда последовательность заканчивается. Из последнего члена этой последовательности получается результат работы.

То есть, мы по исходному данному получаем результат (после выполнения последовательности шагов), и таким образом задаётся функция, которая эти исходные данные преобразует в результаты.

Такая функция и называется вычислимой функцией. Как правило, мы будем считать, что и аргументы, и результаты — это слова в алфавите $\{0, 1\}$.

Обратим внимание, что мы всё время работаем со словами. Исходное данное — слово, результатом является слово, и выполнение — это последовательность слов. Это связано с тем, что самые разные объекты, с которыми имеет дело математическая логика, могут кодироваться словами. В том числе и конечные последовательности тоже легко кодировать словами, о чём мы ещё будем говорить.

Множество всех слов в алфавите $\{0, 1\}$ — счётно, то есть оно может быть поставлено во взаимно однозначное соответствие множеству всех натуральных чисел. Вот пример такого соответствия:

$$
\begin{aligned}
0 &\longleftrightarrow \Lambda \\
1 &\longleftrightarrow 0 \\
2 &\longleftrightarrow 1 \\
3 &\longleftrightarrow 00 \\
4 &\longleftrightarrow 01 \\
5 &\longleftrightarrow 10 \\
6 &\longleftrightarrow 11 \\
7 &\longleftrightarrow 000 \\
&\dots
\end{aligned}
$$

Мы «пересчитываем» двоичные слова в порядке увеличения их длины, а слова одинаковой длины выстраиваем в *лексикографическом* порядке.

Возвращаемся к объяснению понятия алгоритма.

Заметим следующее важное обстоятельство. Про последовательность состояний алгоритма, о которой шла речь (то есть — вычисление), заранее неизвестно, является ли она конечной. Вычисление может оказаться бесконечным.

Иногда, работая за компьютером, вы видите, что он работает очень уж долго, зависает, и ничего не меняется, если вы нажимаете обычные клавиши. Но на самом деле, внутри него что-то происходит. Это видно хотя бы из того, что если вы нажимаете комбинацию клавиш $Ctrl, Alt + Del$, то появляется меню с прерываниями. Так что компьютер работает, но, видимо, собирается работать бесконечно долго. Если вы оставите его включённым в сеть и не будете нажимать $Ctrl, Alt + Del$, то он что-то внутри себя будет считать, хотя и не будет реагировать на обычное нажатие клавиш. Это — пример бесконечной работы алгоритма, и легко придумать простые описания алгоритмов, которые или всегда (что неинтересно), или на некоторых исходных данных будут работать бесконечно долго.

На самом деле, это не очень удобно, и возникает вопрос о том, нельзя ли каким-то образом перестроить алгоритм, вычисляющий данную функцию, чтобы всегда, когда у функции есть значение, он давал бы это значение, а в других случаях говорил, что значения не будет, то есть тоже давал бы значение, но ещё одно, дополнительное значение «не ждите ответа».

Рассмотрение потенциальной возможности бесконечно долгой работы алгоритма — это очень важный вопрос, и мы к нему ещё несколько раз будем возвращаться. Пока просто отметим, что функция, задаваемая алгоритмом, — это не обязательно всюду определённая функция, а частичная функция и интересно было бы понять, можно ли доопределить эту функцию до всюду определённой.

Одна и та же вычислимая функция может вычисляться разными алгоритмами, и даже очень разными. Например, один алгоритм может работать очень долго, а другой — быстро и т. д.

**Теорема 1.** Пусть функции $f$ и $g$ — вычислимы, и пусть функция $h$ является их *суперпозицией*, или *композицией* $(h = g \circ f)$. Тогда функция $h$ тоже вычислима.

**Доказательство.** Рассмотрим следующий алгоритм вычисления функции $h$. К исходному данному $x$ применим алгоритм, вычисляющий функцию $f$, а к результату этого вычисления применим алгоритм, вычисляющий функцию $g$.

Естественно, что если для какого-то $x$ функция $f$ будет неопределена, то есть алгоритм, вычисляющий функцию $f$, не закончит работу, то и алгоритм, вычисляющий функцию $h$, тоже не закончит работу, и функция $h$ тоже будет неопределена.

Или же, может оказаться, что для какого-то $x$ алгоритм, вычисляющий функцию $f$, работу закончит, будет получен результат $f(x)$, но на входном данном $f(x)$ бесконечно долго будет работать алгоритм, вычисляющий функцию $g$. То есть, окажется, что функция $g$ неопределена на $f(x)$. В этом случае алгоритм, вычисляющий функцию $h$, тоже не закончит работу, и окажется, что функция $h$ неопределена на аргументе $x$, что тоже совершенно естественно.

Теорема 1 доказана. Эта теорема легко обобщается также и на функции нескольких аргументов.

## 2. Кодирование упорядоченной пары слов с помощью одного слова

Заметим, что до сих пор речь шла о функциях с одним исходным данным, то есть, с одним аргументом. Но если аргументов несколько, например, мы хотим вычислять какую-то функцию от аргументов $x_1$, $x_2$, то можно считать, что мы хотим вычислять функцию от одного объекта, являющегося парой $\langle x_1, x_2 \rangle$. Эта пара является как бы кодом для двух слов $x_1$ и $x_2$, и тоже двоичное слово.

Можем ли мы пары двоичных слов тоже кодировать двоичным словом? Как это можно было бы сделать?

На первый взгляд, не получается. Если мы просто будем писать одно слово, а рядом другое без запятой (запятую нам нельзя писать), то как мы поймём, где кончается первое слово и начинается второе? Можно на нечётных местах записывать символы первого слова, а на чётных местах — символы второго слова. Но если слова имеют разную длину, то, начиная с какого-то места придётся записывать подряд символы более длинного слова, и как потом мы сможем разделить такую запись на два слова? Нам нужно было бы знать длину хотя бы одного из этих слов.

Но всё-таки, способ кодирования существует. Пары двоичных слов можно выписать в виде таблицы. А дальше можно эту таблицу обходить каким-то разумным образом. Можно обходить по квадратам, а можно обходить по диагоналям. Идея здесь такая же, как при установлении взаимно однозначного соответствия между натуральными числами и парами натуральных чисел.

В таблице 1 показано, как обходом по диагоналям устанавливается взаимно однозначное соответствие между натуральными числами и парами натуральных чисел. В таблице 2 таким же способом устанавливается взаимно однозначное соответствие между словами и парами слов.

Интересно, насколько длиннее код пары $\langle x_1, x_2 \rangle$, чем сумма длин $x_1$ и $x_2$? Хотелось бы, чтобы код был покороче. Возникает вопрос, возможно ли такое кодирование, при котором длина кода пары слов в точности равна сумме длин кодов слов? А если такое кодирование невозможно, то какое возможно самое маленькое увеличение длины? То есть, какое возможно самое плотное, экономное кодирование пар?

## 3. Перечислимые множества

Следующее важное понятие для математической логики и её применений — это перечислимые множества. Опять-таки мы говорим о множествах слов в алфавите $\{0, 1\}$, хотя легко дать определение и в других ситуациях.

**Определение 2.** Множество называется *перечислимым*, если оно является множеством значений какой-нибудь вычислимой функции.

| | 0 | 1 | 2 | 3 | 4 | … |
|---|---|---|---|---|---|---|
| | 0 | 1 | 3 | 6 | 10 | |
| 0 | $\langle 0,0\rangle$ | $\langle 0,1\rangle$ | $\langle 0,2\rangle$ | $\langle 0,3\rangle$ | $\langle 0,4\rangle$ | … |
| | 2 | 4 | 7 | 11 | 16 | |
| 1 | $\langle 1,0\rangle$ | $\langle 1,1\rangle$ | $\langle 1,2\rangle$ | $\langle 1,3\rangle$ | $\langle 1,4\rangle$ | … |
| | 5 | 8 | 12 | 17 | 23 | |
| 2 | $\langle 2,0\rangle$ | $\langle 2,1\rangle$ | $\langle 2,2\rangle$ | $\langle 2,3\rangle$ | $\langle 2,4\rangle$ | … |
| | 9 | 13 | 18 | 24 | 31 | |
| 3 | $\langle 3,0\rangle$ | $\langle 3,1\rangle$ | $\langle 3,2\rangle$ | $\langle 3,3\rangle$ | $\langle 3,4\rangle$ | … |
| | 14 | 19 | 25 | 32 | 40 | |
| 4 | $\langle 4,0\rangle$ | $\langle 4,1\rangle$ | $\langle 4,2\rangle$ | $\langle 4,3\rangle$ | $\langle 4,4\rangle$ | … |
| … | … | … | … | … | … | … |

В бесконечной таблице выписаны пары натуральных чисел. Первый элемент пары равен номеру строки таблицы, второй элемент пары — номеру столбца.

При обходе этой таблицы «по диагоналям» (по возрастанию суммы элементов пары, а при одинаковой сумме — по возрастанию первого элемента пары) устанавливается взаимно однозначное соответствие между натуральными числами и парами натуральных чисел:

$0 \longleftrightarrow \langle 0,0\rangle$
$1 \longleftrightarrow \langle 0,1\rangle$
$2 \longleftrightarrow \langle 1,0\rangle$
$3 \longleftrightarrow \langle 0,2\rangle$
$4 \longleftrightarrow \langle 1,1\rangle$
$5 \longleftrightarrow \langle 2,0\rangle$
$6 \longleftrightarrow \langle 0,3\rangle$
$7 \longleftrightarrow \langle 1,2\rangle$
$8 \longleftrightarrow \langle 2,1\rangle$
$9 \longleftrightarrow \langle 3,0\rangle$
$10 \longleftrightarrow \langle 0,4\rangle$
$11 \longleftrightarrow \langle 1,3\rangle$
$12 \longleftrightarrow \langle 2,2\rangle$
$13 \longleftrightarrow \langle 3,1\rangle$
$14 \longleftrightarrow \langle 4,0\rangle$
$15 \longleftrightarrow \langle 0,5\rangle$
$16 \longleftrightarrow \langle 1,4\rangle$
$17 \longleftrightarrow \langle 2,3\rangle$
$18 \longleftrightarrow \langle 3,2\rangle$
…

**Таблица 1.** Взаимно однозначное соответствие между натуральными числами и парами натуральных чисел. Паре $\langle a,b\rangle$ соответствует натуральное число $n=(a+b)(a+b+1)/2+a$.

| | Λ | 0 | 1 | 00 | 01 | … |
|---|---|---|---|---|---|---|
| | Λ | 0 | 00 | 11 | 011 | |
| Λ | $\langle \Lambda,\Lambda\rangle$ | $\langle \Lambda,0\rangle$ | $\langle \Lambda,1\rangle$ | $\langle \Lambda,00\rangle$ | $\langle \Lambda,01\rangle$ | … |
| | 1 | 01 | 000 | 100 | 0001 | |
| 0 | $\langle 0,\Lambda\rangle$ | $\langle 0,0\rangle$ | $\langle 0,1\rangle$ | $\langle 0,00\rangle$ | $\langle 0,01\rangle$ | … |
| | 10 | 001 | 101 | 0010 | 1000 | |
| 1 | $\langle 1,\Lambda\rangle$ | $\langle 1,0\rangle$ | $\langle 1,1\rangle$ | $\langle 1,00\rangle$ | $\langle 1,01\rangle$ | … |
| | 010 | 110 | 0011 | 1001 | 00000 | |
| 00 | $\langle 00,\Lambda\rangle$ | $\langle 00,0\rangle$ | $\langle 00,1\rangle$ | $\langle 00,00\rangle$ | $\langle 00,01\rangle$ | … |
| | 111 | 0100 | 1010 | 00001 | 01001 | |
| 01 | $\langle 01,\Lambda\rangle$ | $\langle 01,0\rangle$ | $\langle 01,1\rangle$ | $\langle 01,00\rangle$ | $\langle 01,01\rangle$ | … |
| … | … | … | … | … | … | … |

В бесконечной таблице выписаны пары слов. Первый элемент пары соответствует строке таблицы, второй элемент пары — столбцу.

При обходе этой таблицы «по диагоналям» устанавливается взаимно однозначное соответствие между словами и парами слов:

$\Lambda \longleftrightarrow \langle \Lambda,\Lambda\rangle$
$0 \longleftrightarrow \langle \Lambda,0\rangle$
$1 \longleftrightarrow \langle 0,\Lambda\rangle$
$00 \longleftrightarrow \langle \Lambda,1\rangle$
$01 \longleftrightarrow \langle 0,0\rangle$
$10 \longleftrightarrow \langle 1,\Lambda\rangle$
$11 \longleftrightarrow \langle \Lambda,00\rangle$
$000 \longleftrightarrow \langle 0,1\rangle$
$001 \longleftrightarrow \langle 1,0\rangle$
$010 \longleftrightarrow \langle 00,\Lambda\rangle$
$011 \longleftrightarrow \langle \Lambda,01\rangle$
$100 \longleftrightarrow \langle 0,00\rangle$
$101 \longleftrightarrow \langle 1,1\rangle$
$110 \longleftrightarrow \langle 00,0\rangle$
$111 \longleftrightarrow \langle 01,\Lambda\rangle$
$0000 \longleftrightarrow \langle \Lambda,10\rangle$
$0001 \longleftrightarrow \langle 0,01\rangle$
$0010 \longleftrightarrow \langle 1,00\rangle$
$0011 \longleftrightarrow \langle 00,1\rangle$
…

**Таблица 2.** Взаимно однозначное соответствие между словами и парами слов.

Имеются очень сложные проблемы, относящиеся к перечислимым множествам, но есть и довольно простые факты.

Посмотрим на свойства перечислимых множеств, которые в математике принято называть свойствами *замкнутости*. А именно, пусть множество $A$ и множество $B$ — перечислимы. Что можно сказать об их объединении $A \cup B$, пересечении $A \cap B$ и о дополнении $\overline{A}$ к множеству $A$?

**Теорема 2.** Объединение $A \cup B$ перечислимых множеств $A$ и $B$ перечислимо.

**Доказательство.** Так как множество $A$ перечислимо, то, по определению перечислимого множества, существует такая вычислимая функция $f$, что если мы будем подставлять в неё всевозможные исходные данные, то в тех случаях, когда алгоритм, её вычисляющий, будет заканчивать работу, мы будем получать элементы множества $A$, только их, и причём все элементы множества $A$ сможем таким способом получить.

Аналогично, существует вычислимая функция $g$, которая будет давать нам все элементы множества $B$ при подстановке в неё всевозможных слов.

Нам нужно построить вычислимую функцию $h$ для объединения двух множеств.

Алгоритм вычисления функции $h$, получив на вход слово, смотрит, на какой символ заканчивается это слово. Если слово заканчивается нулем, то ноль отбрасывается и на укороченном слове запускается алгоритм вычисления функции $f$.

Если же слово заканчивается единицей, то алгоритм вычисления функции $h$ отбрасывает эту единицу и к тому, что получилось, применяет $g$. Заметим, наконец, что на пустом слове построенная функция $h$ не определена.

Несложно доказывается и следующая теорема:

**Теорема 3.** Пересечение $A \cap B$ перечислимых множеств $A$ и $B$ перечислимо.

А как обстоит дело с дополнением? Можно ли, имея функцию $f$ для $A$, построить какую-то функцию $h$, у которой множество значений будет дополнением к множеству $A$?

Как нам выяснить, что какой-то элемент, какое-то слово, не лежит в $A$? Всё что у нас есть — это только алгоритм для вычисления элементов самого $A$. Мы можем запускать этот алгоритм (последовательно или параллельно) на различных входных данных, получать (в тех случаях, когда алгоритм останавливается) элементы множества $A$. Но как нам устроить (и можно ли так устроить), чтобы алгоритм печатал элементы не из $A$? Здесь есть трудность, и к этому вопросу мы ещё вернёмся.

Замечательным фактом, относящимся к понятию перечислимого множества, является следующее утверждение:

**Теорема 4.** Породимость эквивалентна перечислимости.

**Доказательство.**

1) Пусть множество $A$ перечислимо. Тогда процесс порождения множества $A$ можно описать следующим образом. Используя процесс, который порождает вообще все слова, порождаем какое-нибудь совершенно произвольное слово. К этому слову применяем алгоритм, вычисляющий функцию, множеством значений которой является множество $A$. Если алгоритм заканчивает работу, то результат его работы и объявляем результатом порождения.

2) Пусть множество $A$ породимо. Тогда множество $A$ порождается некоторым исчислением. Можно определить понятие вывода в этом исчислении как последовательности слов, в которой каждое слово выводится за один шаг из каких-то предыдущих. Все последовательности слов закодированы словами. Алгоритм, вычисляющий функцию, множеством значений которой является множество $A$, можно описать следующим образом. Получив на вход некоторое слово, алгоритм проверяет, является ли это слово кодом вывода. Если является, то последнее слово в этом выводе (результат вывода) алгоритм выдаёт в качестве результата своей работы и останавливается. Если же входное слово не является кодом вывода, то алгоритм зацикливается и ничего не выдаёт.

Класс породимых множеств можно охарактеризовать по-разному. Два определения — исходное определение породимости и определение перечислимости у нас уже были.

Перечислимое множество можно определить и как область определения вычислимой функции. Также можно утверждать, что всякое перечислимое множество или пусто, или есть область значений всюду определенной вычислимой функции. Наконец, заметим, что вычислимость функции эквивалентна перечислимости ее графика (множества пар).

### 4. Разрешимые множества

Следующее важное понятие — понятие разрешимого множества. Чтобы его ввести, нужно вспомнить, что такое *характеристическая функция* множества. Характеристическая функция множества равна единице на всех элементах из множества и нулю на всех элементах не из множества. Но это общее определение, пока здесь ничего не говорилось о вычислимости.

**Определение 3.** Множество называется *разрешимым*, если его характеристическая функция вычислима.

Для класса разрешимых множеств тоже естественно рассмотреть вопрос о замкнутости этого класса относительно операций объединения, пересечения и дополнения.

**Теорема 5.** Объединение $A \cup B$ разрешимых множеств $A$ и $B$ разрешимо.

**Доказательство.** Пусть $f$, $g$ и $h$ — характеристические функции множеств $A$, $B$ и $A \cup B$ соответственно. Нужно доказать, что если существуют алгоритмы, вычисляющие функции $f$ и $g$, то функция $h$ — тоже вычислима.

Действительно, алгоритм, вычисляющий функцию $h$, получив входное слово, запускает на этом входе алгоритмы вычисления функций $f$ и $g$. Если они оба дают в качестве результата 0, то и алгоритм, вычисляющий функцию $h$, тоже даёт 0. В остальных случаях (то есть если хотя бы один из алгоритмов выдал единицу) алгоритм, вычисляющий функцию $h$, выдаёт 1.

**Теорема 6.** Пересечение $A \cap B$ разрешимых множеств $A$ и $B$ разрешимо.

**Доказательство** аналогично доказательству предыдущей теоремы, только на этот раз алгоритм, вычисляющий характеристическую функцию множества $A \cap B$ выдаёт 1 тогда и только тогда, когда оба алгоритма (и для $A$, и для $B$) выдали 1. В остальных случаях алгоритм выдаёт 0.

**Теорема 7.** Дополнение $\overline{A}$ к разрешимому множеству $A$ разрешимо.

**Доказательство.** Алгоритм, вычисляющий характеристическую функцию множества $\overline{A}$, выдаёт 1, если алгоритм для $A$ выдал на том же входе 0, и выдаёт 0, если алгоритм для $A$ выдал 1.

Имеется простое и замечательное соответствие между понятиями разрешимости и перечислимости.

**Теорема 8.** Множество разрешимо тогда и только тогда, когда и оно, и его дополнение перечислимы.

**Доказательство.**

1) Если множество разрешимо, то сделать его множеством значений вычислимой функции нетрудно — надо вычислять на каждом аргументе характеристическую функцию множества, если получится единица, то выдать в качестве результата исходное данное, если ноль, то не выдавать ничего (работать бесконечно).

Чтобы дополнение разрешимого множества сделать множеством значений вычислимой функции, нужно поступать наоборот: если характеристическая функция равна единице, не выдавать ничего (работать бесконечно), а если нулю — выдавать в качестве результата исходное данное.

2) Если множество и его дополнение перечислимы, то это дает нам возможность справиться с проблемой бесконечно долгой работы. (Воспользуемся тем, что всякое перечислимое множество является областью определения некоторой вычислимой функции.) Получив аргумент, надо запустить на этом аргументе два алгоритма: один с областью определения, совпадающей с нашим множеством, другой с областью определения, совпадающей с его дополнением (можно, например, чередовать их шаги). Один (и только один) из этих алгоритмов обязательно закончит работу, тем самым мы узнаем, где лежит аргумент.

## 5. Алгоритмические проблемы

В математике довольно давно возникали проблемы построения алгоритмов. Некоторые алгоритмы изучают уже в начальной школе, например, алгоритм сложения столбиком, другие — немножко позднее, например, алгоритм решения квадратных уравнений или алгоритм деления одного многочлена на другой.

Алгоритмы хотелось бы иметь в разных ситуациях.

Я уже упоминал Давида Гильберта. Гильберт поставил 23 проблемы в 1900 году. Мы сейчас объясним, в чем состоит десятая из этих проблем.

Алгебраическое уравнение с несколькими переменными и с целыми коэффициентами может иметь или не иметь решение в целых числах. Гильберт поставил проблему отыскания общего способа — алгоритма, дающего ответ на этот вопрос для всех уравнений.

Усилиями ряда математиков, американских и, в конечном итоге, российского математика Юрия Владимировича Матиясевича, к семидесятому году прошлого века выяснилось, что такого алгоритма нет. То есть, нет общего способа, алгоритма, который по всякому уравнению говорил бы, есть ли у него решения в целых числах.

А о том, как выяснять, разрешимо ли уравнение в действительных числах, мы ещё поговорим в нашем курсе.

Проблема Гильберта упомянута нами в связи с общим понятием алгоритмической проблемы — это проблема построения алгоритма.

Причём в данном случае этот алгоритм не вычислительный, а разрешающий, алгоритм разрешения, выяснения того, верно что-нибудь или нет, верно, что есть решение в целых числах или неверно.

## 6. Алгоритмы Маркова («нормальные алгорифмы»). Тезис Чёрча

Когда мы занимались понятием исчисления, то в какой-то момент мы дали определение грамматики.

В то время, как понятие исчисления было общим и неопределяемым, понятие грамматики было формальным математическим понятием.

Похожая ситуация и с понятием алгоритма. Сейчас мы опишем специальный вид алгоритмов, который будет достаточно общим. Алгоритмы, которые мы рассматриваем, называются алгоритмами Маркова. Марков — это российский математик, ныне покойный, он был первым заведующим нашей кафедры математической логики и теории алгоритмов.

Мы будем считать, что исходные данные алгоритмов — это слова в алфавите $\{0, 1\}$.

**Определение 4.** *Алгоритм Маркова* $\Phi$ — это набор $\langle \Delta, \Pi \rangle$, где

$\Delta$ — алфавит алгоритма $\Phi$,

$\Pi$ — последовательность слов вида $u \longrightarrow v$ или $u \longrightarrow \bullet\, v$ (правил алгоритма $\Phi$), где $u$ и $v$ — слова в алфавите $\Delta$, причём мы считаем, что $\longrightarrow \notin \Delta$ и $\bullet \notin \Delta$.
Слово $u$ называется левой частью правила, $v$ — правой.
Правила, содержащие $\longrightarrow\bullet$, называются заключительными.

Выполнение алгоритма — это последовательность шагов. Применение алгоритма к слову $w$ — это построение последовательности слов, начинающейся с $w$ и такой, что каждое следующее слово получается из предыдущего слова $s$ следующим действием, состоящим из двух шагов:

1) выбрать в последовательности правил $\Pi$ первое правило, левая часть которого входит в $s$;

2) просматривая слово $s$ слева направо, заменить в слове $s$ левую часть правила на правую, как только это окажется возможным.

Если подходящего правила не удалось найти, то построение заканчивается и последнее построенное слово — это результат применения алгоритма $\Phi$ к слову $w$. Если выбранное правило — заключительное, то построение заканчивается после применения этого правила. После применения незаключительного правила построение последовательности слов продолжается применением к новому слову шага 1.

Если последовательность не заканчивается (оказывается бесконечной), то говорят, что алгоритм $\Phi$ не применим к слову $w$.

Традиционно, исторически, правила алгоритма Маркова пишутся по вертикали, как в самих книжках Маркова.

**Пример алгоритма Маркова.**

Алфавит алгоритма состоит из трёх символов: $\Delta = \{0, 1, |\}$.

Последовательность правил $\Pi$ (которую можно ещё также называть программой алгоритма) состоит из трёх правил:

1. $|0 \longrightarrow 0||$
2. $1 \longrightarrow 0|$
3. $0 \longrightarrow$

**Пример работы алгоритма Маркова.**

Если алгоритм, приведённый выше в качестве примера, применить к исходному данному 101, то получим следующую последовательность слов:

1) 101
2) 0|01
3) 00||1
4) 00||0|
5) 00|0|||
6) 000|||||
7) 00|||||
8) 0|||||
9) |||||

Здесь, оказывается, всё так хитро устроено, что тоже выполняется замечательное утверждение, которое называется в данном случае тезисом Чёрча.

**Тезис Черча:** всякая вычислимая функция вычислима алгоритмом Маркова.

Итак, какой бы сложный алгоритм мы ни придумали, использующий весьма сложные операции по работе с символами и т. д., можно построить алгоритм Маркова, который вычисляет ту же самую функцию, то есть на тех же самых исходных данных, если исходный алгоритм кончал работу и получал ответ, будет получать такой же ответ и заканчивать работу, а если исходный алгоритм работу не заканчивал и ответ не получал, то и алгоритм Маркова не будет его получать.

Здесь использовано не совсем стандартное слово — «тезис» — как и в прошлый раз, потому что это не теорема, которую можно доказать, а это утверждение, относящееся к человеческой практике и к предсказанию этой практики.

Мы предсказываем, что закон всемирного тяготения будет действовать и завтра. Так же и здесь. Мы просто предсказываем, что никаких новых вычислимых функций, которых нельзя было бы задать с помощью алгоритма Маркова, не существует и не будет существовать в будущем.