# Конструктивная математика: обзор достижений, недостатков и уроков. Часть II[1]

Н. Н. Непейвода

**abstract.** Lessons of soviet constructivism and their relations to practice are displayed here.

*Keywords:* intuitionism, constructivism, realizability, computability, applications of constructivism

## 1 Введение

В данной работе рассматриваются принципы и достижения советского конструктивизма. Его недостатки и соотношения с другими конструктивными школами будут представлены в следующей части обзора.

## 2 Алгоритмы и реализуемость

С. К. Клини в 1941–1945 годах дал первую интерпретацию конструктивной теории в рамках классической (обратная интерпретация была дана еще раньше Гливенко): рекурсивную реализуемость, которая является конкретизацией реализуемости по Колмогорову, если наши функции — алгоритмы, а функции отождествляются с вычисляющими их программами. Рассмотрим конструкцию Клини подробнее, преобразовав ее таким образом, чтобы она как можно меньше зависела от конкретной формализации понятия алгоритма. Для этой цели дадим аксиоматическое описание понятия алгоритма (оно взято из книги [7]). Это позволяет работать над любым имеющимся множеством исходных программ и типов данных, что соответствует всему духу нашего изложения и требуется для приложений в информатике.

[1]Первая часть работы опубликована в 17 выпуске настоящего издания.

Основное свойство алгоритма — выполнимость на соответствующем устройстве, которым может быть и организационная система (так называемые бизнес-процессы). Тьюринг доказал, что общая модель исполнителя обладает любопытным свойством, которое вовсю используется в современной информатике: есть один исполнитель (*универсальный*), который в принципе может смоделировать поведение и результаты любого другого (если отвлечься от таких "мелочей", как затрачиваемое время и другие ресурсы). А именно:

**(Аксиома универсального алгоритма)** Имеется такой алгоритм $\Upsilon$, что для любого другого алгоритма $\varphi$ найдется его код $\lceil\varphi\rceil$, такой, что результаты вычисления $\varphi(x)$ и $(\Upsilon\ [\lceil\varphi\rceil, x])$ совпадают для любого $x$.

Например, таким исполнителем для программ на обычных языках является машина вместе с операционной системой и транслятором.

Возникает вопрос: а почему данное утверждение сразу не записано формально? В частности потому, что оно ограничивает набор возможных формализаций понятия алгоритма и отсекает некоторые из порою принимаемых по привычке вариантов.

Прежде всего, из анализа аксиомы универсального алгоритма видно, что неудобно брать в качестве данных сами объекты. Лучше пользоваться их структурами. Таким образом, лучше брать не универс $U$, а множество кортежей над ним $U^*$. Из технических соображений к $U$ целесообразно добавить два логических значения **ИСТИНА** и **ЛОЖЬ** (если их там не было).

Раз у нас есть множество кортежей, то нужно иметь элементарные операции над кортежами. Практика и теория совместно показали, что достаточно иметь одну двухместную операцию CONS (см. [3]), присоединяющую свой первый аргумент в качестве первого элемента списка ко второму элементу. Первый ее элемент произвольный, а второй — обязательно список. Кроме того, нужны несколько одноместных операций: две, значением которых являются произвольные кортежи либо объекты:

(TAIL $x$), удаляющая из кортежа его первый элемент;
(HEAD $x$), выделяющая первый элемент кортежа $x$, —

и предикаты:

| | |
|---|---|
| ATOM | Проверяет, является ли $x$ элементом $U$. |
| SIMPLE | Проверяет, что $x$ — кортеж из одного элемента. |
| TRUE | Проверяет, что $x$ — атом, равный ИСТИНА. |
| FALSE | Проверяет, что $x$ — атом, равный ЛОЖЬ. |
| NULL | Проверяет, что $x$ — пустой кортеж [ ]. |

Пустой кортеж обозначаем NIL[2]. Через эти операции выражение, стоящее в аксиоме об универсальном алгоритме, записывается следующим образом:

(1) $(\Upsilon\ (\mathsf{CONS}\ \lceil\varphi\rceil\ (\mathsf{CONS}\ x\ \mathsf{NIL})))$

Конечные кортежи с фиксированным числом элементов определяются следующим образом:

> Выражение $(\mathsf{CONS}\ t_1\ \cdots\ (\mathsf{CONS}\ t_n\ \mathsf{NIL})\cdots)$ обозначается $[t_1, \ldots, t_n]$.

Термы, построенные из заданных известных алгоритмов и предикатов при помощи перечисленных выше операций и имеющие лишь одну свободную переменную $x$, называются *простейшими композициями* алгоритмов. В компьютерах программа-транслятор — универсальный алгоритм в смысле нашей аксиомы, а алгоритмический язык — кодирование алгоритмов. Из рассмотрения данного частного случая кодирования видно, что некоторые простые операции над программами тоже должны быть алгоритмичны, а именно:

**(Аксиома композиций)** Если имеется простейшая композиция переменных алгоритмов $\varphi_1,\ \ldots,\ \varphi_n$

$$T\lfloor\varphi_1,\ \ldots,\ \varphi_n\rfloor,$$

то имеется алгоритм, выдающий для кортежа кодов $[\lceil\varphi_1\rceil,\ \ldots,\ \lceil\varphi_n\rceil]$ код $\lceil T\lfloor\varphi_1,\ \ldots,\ \varphi_n\rfloor\rceil$.

Такие кодирования алгоритмов являются *главными вычислимыми нумерациями* (см. [4]).

ЛЕММА 1. *Из двух принятых выше аксиом следует невозможность построить всюду определенный универсальный алгоритм.*

[2]В отличие от языка Lisp он атомом не считается.

**Доказательство.** Рассмотрим выражение

$$(\psi\ x) \stackrel{\triangle}{=} [(\Upsilon\ \ [x,x])\,,\mathsf{NULL}]\,.$$

Это — простейшая композиция, и она является алгоритмом. Пусть $\pi$ — ее код. Тогда $(\Upsilon\ [\pi,\pi])$ есть по определению универсального алгоритма $(\psi\ \pi)$. По определению $\psi$, последнее выражение записывается как $([\Upsilon\ [\pi,\pi]),\mathsf{NULL}]$. Таким образом, мы получили, что кортеж $\Upsilon([\pi,\pi])$ не меняется после добавления еще одного элемента, чего не может быть.

Следовательно, значение $(\Upsilon\ [\pi,\pi])$ не определено. Q.E.D.

Видно, как тесно связаны элементы формализации. Приняв две естественные аксиомы вычислимости, нужно рассматривать частично определенные функции. Таким образом, слишком рано заменив слова естественного языка на “точные” математические термины, заходим в тупик.

Заметим, что результат о неопределенности не запрещает недетерминированных алгоритмов, которые при одних и тех же начальных данных могут давать разные результаты. Поэтому мы везде старались говорить осторожно: дает тот же результат. Утверждение, что алгоритм $\mu$ на входных данных $a$ может давать результат $b$, обозначается

$$(\mu\ a) \to b.$$

Равенство используется лишь в том случае, если результат имеется и только один.

Таким образом, точная формальная запись свойств универсальной функции:

$$(2)\quad (\Upsilon\ \ (\mathsf{CONS}\ \ulcorner\varphi\urcorner\ \ (\mathsf{CONS}\ x\ \mathsf{NIL}))) \to y \iff (\varphi\ x) \to y.$$

Но даже формально безупречная формулировка содержательно может подвести, поскольку часто важен не только конечный результат, но и процесс его вычисления.

Последний принцип дает возможность определять процедуры рекурсивно, сами через себя. Чтобы максимально ясно выразить его, воспользуемся идеей Дж. Маккарти. Заметим, что простейшие композиции, в которых последним применялся предикат,

можно сами рассматривать как предикаты, поскольку любое их возможное значение может быть только логическим. Такие предикаты назовем *простейшими алгоритмическими*. Определим условные термы следующим образом:

ОПРЕДЕЛЕНИЕ 1. Если $t$ — простейшая композиция, то $t$ — условный терм.

Если $P$ — простейший алгоритмический предикат, а $t$ и $u$ — условные термы сигнатуры $\sigma$, то

$$\mathsf{if}\ P\ \mathsf{then}\ t\ \mathsf{else}\ u\ \mathsf{fi}$$

условный терм той же сигнатуры. Его значения определяются следующим образом:

$$\begin{array}{l}\mathsf{if}\ P\ \mathsf{then}\ t\ \mathsf{else}\ u\ \mathsf{fi} \to b \\ (P\&t \to b) \vee (\neg P\&r \to b).\end{array} \quad \Longleftrightarrow$$

Таким образом, если условие не определено, то условный терм тоже не определен, а вот если одно из выражений не определено, то условный терм может быть определен.

**(Аксиома о рекурсивном определении)** Если все функции сигнатуры $\sigma$ проинтерпретированы как алгоритмы, $\psi$ не входит в $\sigma$, а $t\lfloor\psi, x\rfloor$ — условный терм со свободной переменной $x$ в сигнатуре $\sigma$, пополненной $\psi$, то имеется алгоритм $\psi$, такой, что

$$(\psi\ a) \to b \iff t\lfloor\psi, a\rfloor \to b.$$

Это определение записывается в форме

$$(\psi\ x) \leftarrow t\lfloor\psi, x\rfloor.$$

ПРИМЕР 1. Рекурсивное определение функции, выделяющей последний элемент кортежа, может быть задано следующим образом:

```
(LAST x) ←
 if (ATOM x) then NIL else
  if (NULL x) then NIL else
   if (NULL (TAIL x)) then (HEAD x) else
    (LAST (TAIL x))
 fi fi fi
```

Не уточняется, какое значение принимают исходные функции на неподходящих аргументах (скажем, функция TAIL, если ей подать атом или пустой кортеж). Пример 1 показывает, что можно просто определять новые алгоритмы так, чтобы сомнительные случаи исключались заранее (оставались в другой альтернативе условного терма). Так что если нужна всюду определенность элементарных композиций, то эти функции можно доопределить произвольным образом.

ПРИМЕР 2. Поскольку доказано, что не все алгоритмы определены на всех значениях, естественно ожидать, что главная операция рекурсивного определения приводит к неопределенным функциям. И в самом деле, рассмотрим

$$(\mathsf{ERR}\ x) \leftarrow (\mathsf{CONS}\ \mathsf{NIL}\ (\mathsf{ERR}\ \ x)).$$

Вычисление данного алгоритма не может привести ни к какому результату ни при каком значении $x$. В самом деле, результат должен быть кортежем, поскольку последней применяется функция CONS. Но ни один список не остается сам собой после добавления в него еще одного элемента.

Заметим, что при стандартной семантике рекурсивных процедур не определен гораздо более простой алгоритм:

$$(\mathsf{ERR1}\ x) \leftarrow (\mathsf{ERR1}\ x).$$

Логически привести к противоречию предположение о существовании значения не удается, и можно представить себе семантику алгоритмического языка, раскрывающего такие определения в тождественную функцию (тоже подходящую в качестве решения такого рекурсивного уравнения).

Из неопределенности вычислимых функций (функций, определяемых алгоритмами) следует, что не для всякого предиката множество его истинности можно задать всюду определенным вычислимым предикатом. Появляются следующие определения:

ОПРЕДЕЛЕНИЕ 2. Множество называется *разрешимым*, если имеется всюду определенный алгоритм, вычисляющий его характеристическую функцию (таким образом, данный алгоритм должен всегда давать логическое значение и только одно; такой алгоритм называется *вычислимым предикатом*). Множе-

ство называется *перечислимым*, если имеется однозначный алгоритм, дающий значение ИСТИНА тогда и только тогда, когда элемент принадлежит данному множеству[3].

Например, множество троек $\langle A, a, b\rangle$, где $A$ — код алгоритма, таких, что алгоритм с кодом $A$ дает значение $b$ на входных данных $a$, является перечислимым, но не разрешимым.

Заметим, что дополнения некоторых перечислимых множеств могут быть неперечислимы.

Рассмотрим еще одну аксиому вычислимости, которая отражает, что алгоритмы выполняются шаг за шагом и выдают решение через конечное число шагов. Соответственно, привлекая компьютерные аналогии, вспоминаем, что некоторые программы не переводятся машиной на свой язык, а исполняются при помощи специальной программы-интерпретатора. Существование интерпретатора — более сильное свойство, чем существование универсальной функции, и может быть записано следующим образом. Пусть PAIR — вычислимый предикат, проверяющий, является ли объект кортежем из двух элементов (он легко строится).

**(Аксиома перечислимости)** Имеются такая функция (*универсальный интерпретатор*) UI и предикат UT (*пошаговый тестер*), такие,что

1. $(\mathsf{UI}x) \to y \equiv (\mathsf{TRUE}\ (\mathsf{UT}\ x\ y)$;
2. $(\mathsf{UI}x) \to y \supset (\mathsf{PAIR}\ x) \wedge (\mathsf{PAIR}\ y)$;
3. Для любого алгоритма $\varphi$ $(\varphi\ a) \to b$ тогда и только тогда, когда имеется кортеж $C$, первым членом которого является пара $[\ulcorner\varphi\urcorner, a]$, все последующие члены получаются из предыдущего применением функции UI:

$$(\mathsf{UI}\ [\ulcorner\varpi_i\urcorner, a_i]) \to [\ulcorner\varpi_{i+1}\urcorner, a_{i+1}],$$

а последний имеет вид $[\mathsf{NIL}, b]$.

Используя аксиому перечислимости, можно построить вычислимый предикат $\Psi$, проверяющий, что данный кортеж является протоколом применения универсального интерпретатора, и на последнем шаге получаем окончательный результат.

Если принять аксиому перечислимости, то, в частности, для натуральных чисел любое множество, такое, что и оно само и

[3]Но алгоритм может не давать никакого значения, если элемент множеству не принадлежит.

его дополнение являются перечислимыми, разрешимо. Правда, для действительных чисел и других сложных (не кодируемых достаточно просто натуральными числами) объектов ситуация может быть несколько сложнее.

Порою нужна не разрешимость, а более слабое понятие: *отделимость*.

ОПРЕДЕЛЕНИЕ 3. Множество $Z$ *отделяет* два непересекающихся множества $X$ и $Y$, если $X \subset Z$, $Y \subset \bar{Z}$.

Таким образом, отделимость — понятие относительное. Она зависит от рассматриваемой системы множеств и универса.

ЛЕММА 2. **(Теорема о неотделимости)** *Имеются два перечислимых множества $X$ и $Y$, таких, что нет разрешимого множества $Z$, отделяющего их.*

**Идея доказательства.** В качестве таких двух множеств можно взять

$$\{[x,y] \mid (\Upsilon\ [x,y]) \to 0\},$$
$$\{[x,y] \mid (\Upsilon\ [x,y]) \to 1\}.$$

Если предикат равенства вычислим, то любые два одноэлементных множества отделимы. Но, скажем, для действительных чисел естественно, в частности, рассматривать вычислимость на базисе некоторых стандартных непрерывных аналитических функций ($\cdot$, $+$, $-$, $\backslash$, exp, sin, cos, ln...) и предиката $<$, определенного как частичный, чтобы сохранить непрерывность: если $x = y$, то значение $x < y$ не определено. Поэтому естественно появляется понятие отделимых элементов: два элемента отделимы, если есть вычислимый предикат, истинный на одном из них и ложный на другом.

Имеется просто формулируемая и общая теорема, показывающая, что для свойств вычислимых функций нельзя почти никогда надеяться на разрешимость. Детерминированный алгоритм $\varphi$ называется *экстенсиональным* по первому аргументу, если для всех $A$ и $B$, таких, что функции, ими вычисляемые, одинаковы, имеем $\varphi([A]{*}X) \to y \iff \varphi([B]{*}X) \to y$. Таким образом, он на одинаковых функциях дает одинаковые результаты.

ЛЕММА 3. **(Теорема Успенского–Райса)** *Пусть предикат равенства разрешим. Тогда, если экстенсиональный алгоритм*

*всюду определен, то он не зависит от первого аргумента, т. е.*

$$\forall x, y\,(x, y \text{ — } \textit{коды алгоритмов} \supset \\ \forall z, u\,((\varphi\ [x] * z) \to u \iff (\varphi\ [y] * z) \to u)).$$

**Доказательство.** Рассуждаем от противного. Пусть есть такое $z$, что $\psi = \lambda x\,(\varphi\ [x] * z)$ не является тождественной функцией.

Пусть ERROR — нигде не определенная функция, а $e$ — код одного из алгоритмов, ее вычисляющих. Пусть $(\varphi\ [e] * z) \to a$. Тогда, по экстенсиональности $\varphi$, для любого кода $e_0$ нигде не определенной функции $(\varphi\ [e_0] * z) \to a$. Поскольку $\psi$ не является тождественной, найдется такой код $d$, соответствующий некоторой вычислимой функции $\chi$, что $(\varphi\ [d] * z) \to b$. Теперь построим следующее определение:

$$(3)\quad (\xi\ f\ x\ y) \leftarrow \mathsf{if}\ (\mathsf{ATOM}\ (\Upsilon\ f\ x))\ \mathsf{then}\ (\chi\ y)\ \mathsf{else}\ (\chi\ y)\ \mathsf{fi}.$$

Эта функция нигде не определена, если $(f\ x)$ не определено, в противном случае, при фиксированных $f$, $x$ дает $\chi$. Таким образом, проверяя, чему равно $(\varphi\ \lambda y\,(\xi\ f\ x\ y))$, мы могли бы проверить, применима ли функция к аргументу, что является неразрешимой проблемой. Q.E.D.

И наконец, рассмотрим систему алгоритмов, которая обычно представляется в книгах по теории вычислимости. Она может быть описана следующим образом.

Имеется бесконечно много атомов — натуральные числа. Истину отождествляем с 1, ложь — с 0. Имеются две исходные функции и один предикат, определенные на атомах:

$(\mathsf{S}\ n)$, дающая по $n$ $n+1$,
$(\mathsf{Pd}\ n)$, дающая по $n > 0$ $n-1$, а для 0 — NULL;
$(\mathsf{Z}\ n)$, предикат, проверяющий, равен ли его аргумент 0.

Такие алгоритмы, конечно же, детерминированы, поскольку детерминированы исходные функции, и называются *рекурсивными функциями*. Соответственно, множества, разрешимые при помощи рекурсивных функций, называются *рекурсивно разрешимыми*, а перечислимые с их помощью — *рекурсивно перечислимыми*.

ПРИМЕР 3. Построим алгоритм, осуществляющий сложение двух натуральных чисел:

$$(\mathsf{PLUS}\ m\ n) \leftarrow \mathsf{if}\ (\mathsf{Z}\ n)\ \mathsf{then}\ m\ \mathsf{else}\ (\mathsf{PLUS}\ m\ (\mathsf{Pd}\ n))\ \mathsf{fi}.$$

Другой способ определения алгоритмов дан в определении оператора Mu, строящего минимальное значение $n$, при котором $(\phi\ [m, n]) = 0$.

$$(\mathsf{Muk}\ \phi\ \ m\ k) \leftarrow if\ (\mathsf{Z}\ (\phi\ m\ k))\ \mathsf{then}\ k\ \mathsf{else}\ (\mathsf{Muk}\ \phi\ m\ (\mathsf{S}\ k))\ \mathsf{fi};$$
$$(\mathsf{Mu}\ \phi\ m) \leftarrow (\mathsf{Muk}\ \phi\ m\ 0).$$

Здесь сначала вводится вспомогательный алгоритм, а сама рекурсия идет в другом направлении, чем обычно. Такая операция часто обозначается *квантором минимизации*, введенным Д. Гильбертом:

$$\mu n\,(P\ n),$$

где $P$ — алгоритмический предикат.

Есть экспериментальный факт, степень подтвержденности которого в настоящий момент неизмеримо выше, чем у любого естественнонаучного принципа: все изобретенные детерминированные алгоритмы выражаются как рекурсивные схемы над примененным базисом исходных функций. Как и принято в естественных науках, это экспериментальное наблюдение возведено в ранг общего утверждения, но в отличие от, скажем, законов Ньютона скромно называется *тезис Черча* (может быть, правильнее было бы добавить сюда еще и Тьюринга):

> Любой содержательный замкнутый в себе (не использующий источников ввода, отличных от заранее заданного аргумента) алгоритм выражается в любой из принятых в настоящее время в математике формализаций понятия алгоритма (в частности, как рекурсивная схема).

Из этого тезиса следует строгое утверждение, которое доказано для всех используемых в математике понятий алгоритма:

все они эквивалентны в том смысле, что алгоритмически определимые функции над минимальным базисом понятий совпадают[4]. Но еще более важно то, что для всех явно выделенных в математике способов построения алгоритмов найдено их представление, согласующееся с тезисом Черча[5].

**Внимание!** Тезис Черча работает лишь в том случае, если мы рассматриваем вычислительные устройства сами по себе, без физических датчиков для ввода данных и без взаимодействия с человеком. Эту оговорку практически никто никогда не делает.

Любое частное определение алгоритмов задает пространство в смысле примера 7 из первой части статьи, в котором построено чум-пространство программно определимых функций, поскольку мы принимаем аксиому перечислимости.

Из аксиомы перечислимости и аксиомы о рекурсивном определении следует аксиома об универсальном алгоритме.

Теперь можно определить реализуемость по Клини формул произвольной теории, над объектами которой введено понятие вычислимости, такое, что универсальный интерпретатор определим.

ОПРЕДЕЛЕНИЕ 4. Определим операцию построения по формуле теории формулы, означающей ее реализацию $®\lfloor A, \varsigma \rfloor$), где $\varsigma$ — терм. В определении предполагается, что переменные $\zeta$, $\zeta_1$ и так далее не входят в исходные формулы.

---

[4]Тут есть две ловушки. Во-первых, совпадение определимости не означает сопоставимости ресурсов, необходимых для вычисления данной конкретной функции различными понятиями алгоритма. Так что утверждение, что все можно вычислить на машине Тьюринга, является типичной для математиков демагогией, когда смешивают модальности ‘в принципе возможно’ и ‘практически возможно’. Во-вторых, если исходный базис произвольный, то даже определимость начинает различаться. Рекурсивные схемы — мощнейший в смысле относительной определимости аппарат теории алгоритмов.

[5]Есть одно исключение, найденное автором. Если имеются вращающиеся черные дыры, то в принципе можно организовать вычислительный процесс таким образом, чтобы получить ответ на неразрешимую задачу, но при этом нужно самому успешно совершить путешествие сквозь такую черную дыру в другую вселенную. Таким образом, в некотором смысле тезис Черча и общая теория относительности друг другу противоречат. Глубинные взаимосвязи научного знания гораздо сильнее, чем можно вообразить, наблюдая современную науку, разделенную на почти невзаимодействующие отрасли.

1. Если $A$ — элементарная формула, то

$$®\lfloor A, \zeta \rfloor = A\&(\mathsf{NULL}\ \zeta)$$

2. $$®\lfloor A\&B, \zeta \rfloor = \neg(\mathsf{ATOM}\ \zeta)\&\neg(\mathsf{NULL}\ (\mathsf{TAIL}\ \zeta))\&®\lfloor A, (\mathsf{HEAD}\ \zeta)\rfloor \&®\lfloor B, (\mathsf{HEAD}\ (\mathsf{TAIL}\ \zeta))\rfloor.$$

3. $$®\lfloor A \vee B, \zeta \rfloor = \neg(\mathsf{ATOM}\ \zeta)\&\neg(\mathsf{NULL}\ (\mathsf{TAIL}\ \zeta))\& ((\mathsf{NULL}\ (\mathsf{HEAD}\ \zeta)) \Rightarrow ®\lfloor A, (\mathsf{HEAD}\ (\mathsf{TAIL}\ \zeta))\rfloor)\& (\neg(\mathsf{NULL}\ (\mathsf{HEAD}\ \zeta)) \Rightarrow ®\lfloor B, (\mathsf{HEAD}\ (\mathsf{TAIL}\ \zeta))\rfloor).$$

4. $$®\lfloor A \Rightarrow B, \zeta \rfloor = \forall\zeta_1(®\lfloor A, \zeta_1 \rfloor \Rightarrow \exists C\,(\Psi\ (\mathsf{CONS}\ [\zeta_1, \zeta]\ C))\& ®\lfloor B, (\mathsf{LAST}\ (\mathsf{LAST}\ C))\rfloor)).$$

5. $$®\lfloor \neg A, \zeta \rfloor = (\mathsf{NULL}\ \zeta)\&\neg\exists\zeta_1\, ®\lfloor A, \zeta_1 \rfloor$$

6. $$®\lfloor \exists x\, A, \zeta \rfloor = \neg(\mathsf{ATOM}\ \zeta)\&\neg(\mathsf{NULL}\ (\mathsf{TAIL}\ \zeta)) \&®\lfloor A[x \mid (\mathsf{HEAD}\ \zeta)], (\mathsf{HEAD}\ (\mathsf{TAIL}\ \zeta))\rfloor.$$

7. $$®\lfloor \forall x\, A, \zeta \rfloor = \forall x\,\exists C\,(\Psi\ (\mathsf{CONS}\ [\zeta, x]\ C))\& ®\lfloor A(x), (\mathsf{LAST}\ (\mathsf{LAST}\ C))\rfloor)).$$

Заметим, что в определении реализуемости не понадобился даже предикат равенства или какой-то другой двухместный предикат. Пункты нашего определения переводят на язык абстрактных алгоритмов пункты определения Колмогорова. При этом алгоритм отождествляется с кодом его программы.

Рассмотрим, как в конструктивной теории с реализуемостью по Клини выражаются некоторые свойства функций. Прежде всего, следуя Клини, формально выразим тезис Черча.

$$\begin{array}{ll} & \forall x\,(\neg A(x) \Rightarrow \exists y\, B(x,y)) \Rightarrow \\ (4) & \quad \exists z \forall x\,(\neg A(x) \Rightarrow \\ & \qquad \exists u\,(\Upsilon\ [z,x]) \to u \& \forall u\,(\Upsilon\ [z,x]) \to u \Rightarrow B(x,u)) \end{array}$$

В большинстве теорий из тезиса Черча следуют, в частности, результаты, несовместимые с классической логикой. Например, если выразимо на языке нашей теории хоть одно неразрешимое свойство $U(x)$, то реализуема формула

$$\neg\forall x\,(U(x) \vee \neg U(x)). \tag{5}$$

Тем самым мы получаем конструктивную характеризацию классической логики: она полностью приемлема конструктивно, если в теории все свойства разрешимы. То есть теория должна быть полна. Поэтому, в частности, в элементарной геометрии или в алгебраической теории действительных чисел классическая логика как раз на месте. В полной теории она не может привести к «грязным» теоремам существования.

Реализуемость, понимаемая классически, влечет в случае перечислимых типов данных важный логический принцип:

**Принцип конструктивного подбора**

$$\forall x\,(A(x) \vee \neg A(x))\&\neg\neg\exists x\,A(x) \Rightarrow \exists x\,A(x). \tag{6}$$

Этот принцип часто называют *принцип Маркова* и обозначают **PM**.

Содержательно принцип конструктивного подбора означает важный практический вывод: если у нас есть способ проверки корректности решения и алгоритм перебора всех возможных значений типа данных, то можно спокойно применять для обоснования корректности задачи поиска классическую логику. Если в принципе процесс должен закончиться, мы просто запустим программу и дождемся ее остановки.

Новые выразительные возможности открываются и в плане выражения неспособности что-то сделать алгоритмически. Например, теорема Успенского–Райса о невозможности эффективного распознавания свойств алгоритмов может быть переформулирована для абстрактных типов (например, функций, множеств...) следующим образом:

$$\begin{aligned}&\forall X : \mathsf{Function}\,\exists b : \mathsf{Bool}\ A(X,b) \Rightarrow\\ &\quad\exists b : \mathsf{Bool}\,\forall X : \mathsf{Function}\ A(X,b)\end{aligned} \tag{7}$$

(формулировка открыта Троэлстра).

## 3 Советский конструктивизм

А. А. Марков (1947) на базе теории алгоритмов предложил вариант конструктивизма, базирующийся на традиционном естественнонаучном материализме и рационализме. Первоначально он использовал в качестве основы конструктивного понимания математических суждений реализуемость по Клини. Но он, так же, как и Брауэр, поставил задачу конструктивной переработки не какой-то отдельной теории, а всей математики, как она тогда понималась.

По рассказам самого Андрея Андреевича, переход к конструктивизму стимулировали его работы во время Второй мировой войны. Она в значительной степени решилась соревнованием математиков, расшифровывавших коды противников и изобретавших надежные коды для своих войск. А. А. Марков работал именно на этом сверхсекретном фронте и на своем опыте убедился, насколько трудно отличить математические результаты, дающие метод их использования, от тех, которые остаются лишь идеальными. Так что Марков решил придать математике более прикладной характер. И путь к этому он увидел в брауэровском подходе.

Поскольку по своему мировоззрению А. А. был убежденным естественнонаучным материалистом, философские основания подхода Брауэра были для него неприемлемы. Но, поскольку он был действительно свободомыслящим, а не из тех, которые допускают для других свободу мыслить так же, как они, и падают в обморок (или начинают говорить непристойности) при одном упоминании имени Бога, он четко и объективно проанализировал со своей точки зрения достижения Брауэра и всегда воздавал ему должное как отцу-основателю нового направления.

Сам Марков выделил три основные абстракции, на которых, с точки зрения естественнонаучного рационализма, базируется математика.

**Абстракция отождествления**. Она сформулирована косвенным образом Г. Лейбницем в его определении равенства: «Равные объекты — обладающие одинаковыми свойствами». Если понимать это с точки зрения информатики, то, поскольку предметов с совершенно одинаковыми свойствами просто нет,

нам остается переформулировать абстракцию отождествления следующим образом:

**«Мы отказываемся рассматривать в математической формализации некоторые свойства, различающие предметы, и тем самым отождествляем их».**

**Абстракция потенциальной бесконечности. Считается выполнимым любое конечное число действий, построение сколь угодно сложных конечных объектов. Абстрагируемся от ограниченности времени, памяти и других ресурсов.**

**Абстракция актуальной бесконечности. Бесконечные процессы и бесконечные совокупности можно рассматривать как завершенные объекты и свободно оперировать с ними.**

Таким образом, А. А. Марков заменил гильбертовское понятие «идеальных объектов» и брауэровские «идеальные умственные построения» внешне безобидными с точки зрения примитивного материализма абстракциями от реально существующих объектов. Это было необходимо сделать в те времена, когда распоясавшиеся «свободомыслящие» набрасывались на любое место, где им чудился призрак идеализма или Бога[6]. Надо заметить, что, как всегда, когда человек вынужден как можно точнее и выразительнее перевести понятия на другой язык, Марков выявил новые важные стороны математических идеализаций и упорядочил их в единую систему. Эта система идеализаций в дальнейшем послужила толчком к открытию логик реальной осуществимости.

Основные предположения советского конструктивного направления в математике можно выразить следующим образом:

1. Математические объекты, рассматриваемые в конструктивизме, являются конструктивными объектами: результатами конечных комбинаций простейших действий над четко различимыми исходными сущностями. В конкретной реализации конструктивными объектами почти всегда были слова в конечном алфавите. Но порою рассматривались и

[6]Из советского уголовного кодекса было выброшено понятие «идеальной совокупности преступлений», т. е. нескольких преступлений, совершаемых одним и тем же действием, поскольку оно пахло идеализмом.

другие типы конструктивных объектов (например, конечные графы). В этом отношении советская конструктивная математика четко поставила вопрос о представлении данных, исключительно важный для информатики и полностью игнорировавшийся и классической, и интуиционистской теорией.

2. Все действия являются алгоритмами. Алгоритмы задаются своими программами. Алгоритмы рассматривались лишь над фиксированным базисом конструктивных объектов и лишь над стандартным базисом исходных действий. В этом отношении советская конструктивная математика явилась шагом назад по сравнению с классической для потребностей информатики, где оказались наиболее применимы высокоуровневые алгебраические и топологические структуры.

3. Некоторые конструктивные объекты могут отождествляться и называться равными. Здесь впервые было систематически исследовано влияние множественных представлений одного и того же объекта на алгоритмы работы с ним. Это кладезь идей для современной информатики, практически не затронутый ею.

4. Не предполагалось, что все проблемы решены, и что в областях, имеющих дело с актуальной бесконечностью, они могут быть все решены. Но по поводу любой конкретной точно поставленной проблемы, касающейся конструктивных объектов, конструктивисты были уверены, что при желании она может быть решена человечеством. Таким образом, советский конструктивизм в максимально возможной степени сохранял веру в познаваемость мира и в прогресс. Незнание никогда не рассматривалось как нечто большее, чем временный статус. Принцип «*ignoramus et ignorabimus*» столь же был чужд для советских конструктивистов, как и почти для всех естественнонаучных рационалистов.

5. Конструктивизм полностью принимал абстракции отождествления и потенциальной осуществимости и безогово-

рочно отвергал для себя абстракцию актуальной бесконечности.

6. Конструктивизм позиционировал себя как альтернативу классической математике, а не как ее «могильщика». Классическая математика должна была лишь согласиться на равных разговаривать с конструктивистами и мирно сосуществовать с ними, стремясь к взаимопониманию и взаимообогащению. Но этого большинство «классиков» как раз и не могло допустить. «Практические» математики, к реальной практике никакого отношения не имевшие и интересовавшиеся лишь доказательством теорем, отождествили точность с привычной им манерой изложения и оказались нетерпимыми к малейшим отступлениям от нее.

Надо сказать, что советский конструктивизм оказался мощным психологическим орудием вовлечения активно работающих математиков в конструктивную парадигму. По видимости он не противоречил традиционному «точному» мышлению, но постепенно перестраивал ум на новые рельсы не менее радикально, чем интуиционизм. Многие ученики Маркова и его учеников, даже внешне ушедшие в совсем другие области, говорили о том, что метод конструктивного мышления им сильно помогает на практике, хотя прямо результаты конструктивизма они применить не могут.

В последнее время метод конструктивного мышления (правда, уже в другой обстановке и на другом уровне) успешно применен для подготовки специалистов-информатиков высокого уровня.

А. А. Марков заметил, что из принятых им посылок вытекает целесообразность принятия принципа конструктивного подбора. Но этот принцип оказался более фундаментальным, чем простое концептуальное следствие из методологических предположений. Это установил Н. А. Шанин, разработав алгоритм конструктивного понимания математических суждений (*конструктивная расшифровка*). Опишем его.

Прежде всего, в практических конструктивных теориях естественно предполагать по крайней мере возможность снятия двойного отрицания с элементарных формул. В противном слу-

чае внутри них спрятаны скрытые объекты и построения, что кажется крайне нежелательным. На самом деле почти всегда элементарные формулы разрешимы.

*В алгоритме конструктивной расшифровки предполагается, что с элементарных формул можно снимать двойное отрицание.*

Далее, поскольку по определению универсального интерпретатора предикат $\Psi$ разрешим, по принципу конструктивного подбора с формулы $\exists x(\Psi\ \alpha\ x)$ можно снимать двойное отрицание. Таким образом, с утверждения, что алгоритм дает результат при конкретных исходных данных, можно снимать двойное отрицание. Осмысленность алгоритмического терма в конструктивизме обозначается $!t$.

Формулы, составленные из элементарных и $!t$ без помощи связок $\vee$ и $\exists$, называются *устойчивыми*[7]. С устойчивых формул можно снимать двойное отрицание (а, значит, и навешивать его с сохранением эквивалентности). Следовательно, их реализации можно считать тривиальными. Для устойчивых формул применима в полном объеме классическая логика, в которой исключена связка $\vee$ и квантор $\exists$ (что не ограничивает общности, поскольку данные связки выразимы через остальные). Поэтому устойчивые формулы называем еще *классическими*.

Формулы, реализации которых нетривиальны, Шанин назвал *содержащими конструктивную задачу* или *проблемными*. Его алгоритм конструктивной расшифровки выявляет эту задачу и делит формулу на две части: конструктивное построение и его обоснование.

ОПРЕДЕЛЕНИЕ 5. Строим по каждой формуле $A$, не являющейся классической, формулу $\mathbf{Ш}\lfloor A\rfloor$, определяемую при помощи вспомогательной классической формулы $\mathbf{Ш}_0\lfloor A, x\rfloor$. Для большинства неклассических формул выполнено

$$\mathbf{Ш}\lfloor A\rfloor = \exists x\ \mathbf{Ш}_0\lfloor A, x\rfloor.$$

[7] Сам Н. А. Шанин и советские конструктивисты называли их “нормальными”. Мы систематически изгоняем традиционные в математике абсолютно бессодержательные названия типа “нормальный”. Практика показала, что их употребление в диалоге со специалистами из других предметных областей дезориентирует обе стороны.

В оставшихся же случаях результирующая формула оказывается классической, и тем самым наша формула эквивалентна классической[8].

1. $\mathbf{Ш}\lfloor A\rfloor = A$ для классических формул.
2. Если $A$ — проблемная формула, а $B$ — классическая, то
$$\mathbf{Ш}\lfloor A \Rightarrow B\rfloor = \forall x\,(\mathbf{Ш}_0\lfloor A, x\rfloor \Rightarrow B).$$
3. Если $A$ — проблемная формула, то $\mathbf{Ш}\lfloor\neg A\rfloor = \forall x\,\neg\,\mathbf{Ш}_0\lfloor A, x\rfloor$.
4. Если оба члена дизъюнкции классические, то
$$\mathbf{Ш}_0\lfloor A \vee B, x\rfloor = ((\mathsf{TRUE}\ x) \Rightarrow A)\&((\mathsf{FALSE}\ x) \Rightarrow B)\& \neg(\neg(\mathsf{TRUE}\ x)\&\neg(\mathsf{FALSE}\ x)).$$
$$\mathbf{Ш}\lfloor A \vee B\rfloor = \exists x\ \mathbf{Ш}_0\lfloor A \vee B, x\rfloor.$$
5. Если $A$ — проблемная формула, а $B$ — классическая, то
$$\mathbf{Ш}_0\lfloor A\&B, x\rfloor = \mathbf{Ш}_0\lfloor A, x\rfloor\&B,$$
$$\mathbf{Ш}\lfloor A\&B\rfloor = \exists x\ \mathbf{Ш}_0\lfloor A\&B, x\rfloor.$$
6. Если $B$ — проблемная формула, а $A$ — классическая, то
$$\mathbf{Ш}_0\lfloor A \Rightarrow B, x\rfloor = (A \Rightarrow !(\Upsilon\ [x, \mathsf{NIL}])\&\,\mathbf{Ш}_0\lfloor B, (\Upsilon\ [x, \mathsf{NIL}])\rfloor,$$
$$\mathbf{Ш}\lfloor A \Rightarrow B\rfloor = \exists x\ \mathbf{Ш}_0\lfloor A \Rightarrow B, x\rfloor.$$
7. Если $A$ — проблемная формула, а $B$ — классическая, то
$$\mathbf{Ш}_0\lfloor A \vee B, x\rfloor = ((\mathsf{TRUE}\ (\mathsf{HEAD}\ x)) \Rightarrow \mathbf{Ш}_0\lfloor A, (\mathsf{HEAD}\ (\mathsf{TAIL}\ x))\rfloor)\& ((\mathsf{FALSE}\ (\mathsf{HEAD}\ x)) \Rightarrow B)\& \neg(\neg(\mathsf{TRUE}\ (\mathsf{HEAD}\ x))\&\neg(\mathsf{FALSE}\ (\mathsf{HEAD}\ x))).$$
$$\mathbf{Ш}\lfloor A \vee B\rfloor = \exists x\ \mathbf{Ш}_0\lfloor A \vee B, x\rfloor.$$

[8] Порою сильно отличающейся от исходной.

8. Если оба члена конъюнкции проблемные, то

$$\mathbf{Ш}_0\lfloor A\&B, x\rfloor = \mathbf{Ш}_0\lfloor A, (\mathsf{HEAD}\ x)\rfloor \& \mathbf{Ш}_0\lfloor B, (\mathsf{HEAD}\ (\mathsf{TAIL}\ x))\rfloor,$$

$$\mathbf{Ш}\lfloor A\&B\rfloor = \exists x\ \mathbf{Ш}_0\lfloor A\&B, x\rfloor.$$

9. Если оба члена дизъюнкции проблемные, то

$$\begin{aligned}\mathbf{Ш}_0\lfloor A \vee B, x\rfloor = ((&\mathsf{TRUE}\ (\mathsf{HEAD}\ x)) \Rightarrow \\ &\mathbf{Ш}_0\lfloor A, (\mathsf{HEAD}\ (\mathsf{TAIL}\ x))\rfloor)\& \\ ((&\mathsf{FALSE}\ (\mathsf{HEAD}\ x)) \Rightarrow \mathbf{Ш}_0\lfloor B, (\mathsf{HEAD}\ (\mathsf{TAIL}\ x))\rfloor)\& \\ &\neg(\neg(\mathsf{TRUE}\ (\mathsf{HEAD}\ x))\&\neg(\mathsf{FALSE}\ (\mathsf{HEAD}\ x))).\end{aligned}$$

$$\mathbf{Ш}\lfloor A \vee B\rfloor = \exists x\ \mathbf{Ш}_0\lfloor A \vee B, x\rfloor.$$

10. Если оба члена импликации проблемные, то

$$\begin{aligned}&\mathbf{Ш}_0\lfloor A \Rightarrow B, x\rfloor = \\ &\quad \forall z\,(\mathbf{Ш}_0\lfloor A \Rightarrow B, z\rfloor \Rightarrow !(\Upsilon\ [x, z])\& \mathbf{Ш}_0\lfloor B, (\Upsilon\ [x, z])\rfloor),\end{aligned}$$

$$\mathbf{Ш}\lfloor A \Rightarrow B\rfloor = \exists x\ \mathbf{Ш}_0\lfloor A \Rightarrow B, x\rfloor.$$

11. Если $A(z)$ — классическая, то

$$\mathbf{Ш}_0\lfloor \exists z\, A(z), x\rfloor = A(x),$$

$$\mathbf{Ш}\lfloor \exists z\, A(z)\rfloor = \exists x\ \mathbf{Ш}_0\lfloor \exists z\, A(z), x\rfloor.$$

12. Если $A(z)$ — проблемная, то

$$\mathbf{Ш}_0\lfloor \exists z\, A(z), x\rfloor = \mathbf{Ш}_0\lfloor A((\mathsf{HEAD}\ x)), (\mathsf{HEAD}\ (\mathsf{TAIL}\ x))\rfloor,$$

$$\mathbf{Ш}\lfloor \exists z\, A(z)\rfloor = \exists x\ \mathbf{Ш}_0\lfloor \exists z\, A(z), x\rfloor.$$

13. Если $A(z)$ — проблемная, то

$$\mathbf{Ш}_0\lfloor \forall z\, A(z), x\rfloor = \forall z\,(!(\Upsilon\ [x, z])\& \mathbf{Ш}_0\lfloor A(z), (\Upsilon\ [x, z])\rfloor),$$

$$\mathbf{Ш}\lfloor \forall z\, A(z)\rfloor = \exists x\ \mathbf{Ш}_0\lfloor \forall z\, A(z), x\rfloor.$$

В этом определении сокращены лишние части, которые получаются при прямом применении реализуемости по Клини. Как показал Клини, конструктивная расшифровка Шанина с классической точки зрения эквивалентна его реализуемости. Алгоритм Шанина, при большей громоздкости описания, выявляет некоторые важные стороны понимания суждений в советском конструктивизме. Многие из выявленных при создании этого алгоритма свойств важны для применения конструктивизма (не только советского) с целью анализа понятий информатики.

ПРИМЕР 4. **Сравнение конструктивной расшифровки и утверждения о реализуемости формулы.**

Рассмотрим формулу

(8) $\neg\forall x(A(x) \vee \neg A(x)),$

где $A(x)$ — полностью классическая неразрешимая формула (не содержащая даже ссылок на принцип конструктивного подбора) вида $\forall x(\mathsf{NULL}\ (\varphi\ x))$, где $\varphi$ — функция, всюду определенность и однозначность которой устанавливается самыми элементарными средствами (например, в арифметике примитивно-рекурсивная функция). Пусть эта функция определена термом языка, так что дальнейших сложностей нет. Тогда утверждение о реализуемости формулы (8) выглядит следующим образом:

$$\begin{aligned}(9)\quad &\exists x\,(\forall\zeta\,\neg(\forall z\,\neg(\mathsf{ATOM}\ \zeta)\&\neg(\mathsf{NULL}\ (\mathsf{TAIL}\ \zeta))\&\\ &((\mathsf{NULL}\ (\mathsf{HEAD}\ \zeta)) \Rightarrow \forall x_1\,(!(\Upsilon\ [(\mathsf{HEAD}\ (\mathsf{TAIL}\ \zeta)), x_1])\&\\ &\forall\zeta_2\,((\Upsilon\ [(\mathsf{HEAD}\ (\mathsf{TAIL}\ \zeta)), x_1]) \to \zeta_2 \Rightarrow (\mathsf{NULL}\ \zeta_2)\&A(z))\&\\ &(\neg(\mathsf{NULL}\ (\mathsf{HEAD}\ \zeta)) \Rightarrow\\ &\forall\zeta_3\,\neg\forall x_3\,(!(\Upsilon\ [\zeta_3, x_3])\&\forall\zeta_4\,((\Upsilon\ [\zeta_3, x_3]) \to \zeta_4 \Rightarrow (\mathsf{NULL}\ \zeta_4))\&\\ &(\mathsf{NULL}\ (\mathsf{HEAD}\ (\mathsf{TAIL}\ \zeta)))))\&(\mathsf{NULL}\ x)\&\neg A(z))\end{aligned}$$

Приведем шанинскую расшифровку той же формулы.

$$\begin{aligned}(10)\quad &\forall\zeta\,\neg\forall x\,(!(\Upsilon\ [\zeta, x])\&\\ &((\mathsf{TRUE}\ (\Upsilon\ [\zeta, x])) \Rightarrow A(x))\&((\mathsf{FALSE}\ (\Upsilon\ [\zeta, x])) \Rightarrow \neg A(x))).\end{aligned}$$

По шанинской формулировке легко усмотреть, что она утверждает отсутствие разрешающего алгоритма, а вот по клиниевской... Она загромождена множеством лишних тривиальных

построений. А если формула $A(x)$ сформулирована чуть посложнее, ситуация становится просто безнадежной.

Интересна дискуссия между Н.А. Шаниным и С.К. Клини по поводу соотношения их расшифровок. Клини утверждал, что Шанин ничего нового не сделал, а Шанин подметил то, что полностью игнорируется «практическими» математиками, но важно для информатиков. Его алгоритм четче по структуре, яснее по результату и, самое главное, идемпотентен: дважды применив его к формуле, мы ничего не изменяем по сравнению с первым применением. А реализация по Клини будет разбухать неограниченно.

Алгоритм конструктивной расшифровки можно переписать в абстрактные теории без понятия алгоритма, основываясь на типах данных. То, от чего он существенно зависит, было нами сформулировано выше. Поэтому сфера применимости шанинской расшифровки гораздо шире советского конструктивизма.

В этом алгоритме формула четко делится на построение объекта ( квантор $\exists$) и обоснование проделанного построения. Формула делится на конструктивную и дескриптивную части. Если построение уже осуществлено, то обосновывать его можно средствами классической математики. Правда, в дальнейшем Н.А. Шанину такой вывод (который он сам сделал) показался слишком оппортунистическим и он отказался от собственного алгоритма.

Но можно пойти и в другую сторону. В обосновании построения можно разрешить идеальные объекты, в том числе и актуальную бесконечность. Все равно обосновывать программы, опираясь лишь на те понятия, которые используются при их непосредственном исполнении, невозможно. Приходится вводить *призраки*, которые необходимы для понимания и обоснования программы, но как минимум бесполезны в ходе ее исполнения. Призраками могут быть в том числе и существенно нефинитные объекты (например, количество шагов процедуры можно оценивать ординалами, а для получения оценок точности вычислений применять нестандартные действительные числа).

Явное признание вычислимости алгоритмичностью приносит немало выгод. Рассмотрим в связи с этим пример Шпекера, строящий опровержение утверждения о сходимости возрастаю-

щей ограниченной сверху последовательности без помощи метода провокации, который подвержен критике.

ПРИМЕР 5. В любом абстрактном понятии алгоритма можно смоделировать натуральные числа как кортежи вида

$$[\mathsf{NIL}, \ldots, \mathsf{NIL}].$$

Понятие натурального числа разрешимо.

$$(\mathsf{NAT}\ x) \leftarrow \textbf{if}\ (\mathsf{NULL}\ x)\ \textbf{then true else}\ (\mathsf{NULL}\ (\mathsf{HEAD}\ x))\&(\mathsf{NAT}\ (\mathsf{TAIL}\ x))\ \textbf{fi}$$

Равенство натуральных чисел разрешимо.

$$(\mathsf{NATEQ}\ x\ y) \leftarrow (\mathsf{NAT}\ x)\&(\mathsf{NAT}\ y)\&\textbf{if}\ (\mathsf{NULL}\ x)\ \textbf{then}\ (\mathsf{NULL}\ y)\ \textbf{else}\ (\mathsf{NATEQ}\ (\mathsf{TAIL}\ x)\ \mathsf{TAIL}\ y))\ \textbf{fi}$$

Поэтому можно свободно пользоваться последовательностями объектов (конечно же, алгоритмическими). Для любого понятия алгоритма существует счетно-перечислимое (т. е. являющееся множеством значений последовательности) неразрешимое множество. Поскольку алгоритмически можно определить лишь счетное множество множеств, а множество всех множеств натуральных чисел сложнее (несчетно)[9], есть неразрешимое множество натуральных чисел, а, значит, не все счетно-перечислимые множества натуральных чисел разрешимы. Возьмем неразрешимое счетно-перечислимое множество натуральных чисел $X$. Тогда по определению счетной перечислимости имеется алгоритмическая последовательность натуральных чисел, порождающая все элементы этого множества. По разрешимости равенства натуральных чисел ее можно модифицировать (просто исключая повторяющиеся элементы) таким образом, что она будет перечсислять его без повторений. Поскольку множество неразрешимо, она будет перечислять его элементы в некотором неизвестном порядке. Такую последовательность обозначим $\alpha$. Построим ряд

[9]Мы отказываемся употреблять давно уже ставшее невежественным чтение мощности бесконечных множеств как «числа элементов в них». Это одна из мер сложности описания бесконечного множества.

$$(11)\ \mathrm{Sp} = \sum_{i=0}^{\infty} \frac{1}{2^{(\alpha\ i)}}$$

Если бы было можно построить последовательность стягивающихся сегментов, сходящуюся к его сумме, был бы построен разрешающий предикат для множества $X$, просто вычисляя Sp с точностью $\frac{1}{2^{n+2}}$, чтобы проверить принадлежность $n \in X$.

## Литература


[1] *Ершов Ю. Л.* Теория нумераций. М.: Наука, 1977. 416 с.

[2] *Митчелл Дж.* Основания языков программирования. М.; Ижевск: РХД, 2010. 720 с.

[3] *Городняя Л. В.* Основы функционального программирования. М: ИНТУИТ, 2004. 270 с.

[4] *Ершов Ю. Л.* Теория нумераций. М.: Наука, 1977. 416 с.

[5] *Колмогоров А. Н.* Zur deutung der intuitionistischen Logik // Math. Z., **35** (1932), 58–65.

[6] *Непейвода А. Н.* О сюръективной импликации в реверсивной логике. // Смирновские чтения **5**, (2009). С. 72–74.

[7] *Непейвода Н. Н.* Прикладная логика. Новосибирск: НГУПресс, 2000. 448 с.

[8] *Непейвода Н. Н.* О прикладных теориях с суперинтуиционистскими логиками // Логические исследования. Вып. 7. С. 61–71. М.: Наука, 2000.

[9] *Непейвода Н. Н., Скопин И. Н.* Основания программирования. М.; Ижевск, 2004.

[10] *Непейвода Н. Н.* Конструктивная математика: обзор достижений, недостатков и уроков. Часть I // Логические исследования. Вып. 17. М.; СПб, 2011. С. 191–239.

[11] *Шанин Н. А.* О конструктивном понимании математических суждений. Тр. МИАН СССР, **52**, 1958. С. 226–311.

[12] *Шурыгин В. А.* Основы конструктивного математического анализа. М.: URSS, 2009. 326 с.

[13] *Brouwer L.E.J.* Over de grondslagen der wiskunde. Thesis. Amsterdam, 1907.

[14] *Brouwer L.E.J.* De onbetrouwbaarheid der logische principes // Tijdschrift voor wijsbegeerte, **2**, 1908.

[15] *Brouwer L.E.J.* Intuitionisme en formalisme. Groningen, 1912.

[16] *Brouwer L.E.J.* Besitzt jede reele Zahl eine dezimalbruchentwicklung?// Proc. Acad. Amsterdam, **23**. P. 949–954.

[17] *Brouwer L. E. J.* Richtlijnen der intuitionistische wiskunde // Proc. Acad. Amsterdam, **50**, 339. 1947.

[18] *Glivenko V.* Sur la logique de M.Brouwer // Academie Royale de Belgique. Bulletins de la classe des sciences. Ser. 5, **14**, 1928. P. 225–228.

[19] *Glivenko V.* Sur quelques points de la logique de M.Brouwer // Academie Royale de Belgique. Bulletins de la classe des sciences. Ser. 5, **15**, 1929. P.183–188.

[20] *Kleene S.C.* Introduction in metamathematics. New-York: 1952 (русский перевод *С.К. Клини.* Введение в метаматематику. М., 1957).

[21] *Kreisel G., Lacombe D.* Ensembles récursivement measurables et ensembles récursivement ouvert les fermes // Compt. rend Acad. si. Paris **245**, № 14 (1957). P. 1106–1109.

[22] *Kreisel, G., Troelstra A. S.* Formal systems for some branches of intuitionistic analysis // Ann Math Log, **1**:229–387. 1970.
[23] *Martin-Löf P.* Notes on constructive mathematics. Almqvist & Wiskell, Stockholm, 1970.
[24] *Troelstra A. S.* History of Constructivism in the Twentieth Century // University of Amsterdam, ITLI Prepublication Series ML-91-05, 1991.
[25] *Troelstra A. S.* From Constructivism to Computer Science // Theoretical Computer Science **211**, 233–252. 1999.