Вычислимость по Тьюрингу превосходит в принципиальном отношении вычислимость рекурсивных функций

Д. Л. Гуринович

03 августа 2022 года

УДК 510.51

Содержание

Аннотация

В данной работе продемонстрировано, что вычислимость по Тьюрингу превосходит вычислимость рекурсивных функций, вопреки принятому сейчас и «доказанному» мнению об их одинаковой вычислимости. Так же, намечен путь построения теории строк.

1 Вычисление длины числа-строки на ленте машины Тьюринга и отсутствие аналога в арифметике и среди рекурсивных функций для такой же вычислимости длины натурального числа

Если взять самую общеизвестную информацию о вычислимости – то есть, информацию из Википедии, то вот она:

Вычислимые функции — это множество функций вида, $f : N \to N$ которые могут быть реализованы на машине Тьюринга. Задачу вычисления функции f называют алгоритмически разрешимой или алгоритмически неразрешимой, в зависимости от того, возможен ли алгоритм, вычисляющий эту функцию. В качестве множества N обычно рассматривается множество B_* — множество слов в двоичном алфавите $B = \{0, 1\}$.

По мере развития теории вычислимости было сформулировано несколько определений, которые, как оказалось впоследствии, определяют одно и то же множество функций — множество вычислимых функций:

функции, реализуемые на машинах Тьюринга (Алан Тьюринг);

функции, реализуемые на нормальных алгоритмах Маркова (А. А. Марков);

функции, реализуемые на машине Поста;

частично рекурсивные функции (Курт Гёдель, Стивен Клини);

Конец цитаты

Нас будет интересовать вычислимость рекурсивных функций, потому что они «представимы» в арифметике и поэтому арифметика, вроде бы, является теорией, пригодной для исследования вычислимости. Однако мы убедимся, что это не так.

Пусть на ленте машины Тьюринга записано следующее число в шестнадцатеричном виде:

152a0ed7f

Назовём это число «входным аргументом».

Имеется простая программа, которая пересчитывает количество не пустых символов в приведённом сплошном блоке не пустых символов (во входном аргументе) и записывает на ленте МТ число – «длину» данного сплошного блока символов. Притом записывать может любым заранее оговоренным способом – в десятичном, шестнадцатеричном виде или в духе стандартной интерпретации.

Допустим, наша программа возвращает результат в десятичном виде. Тогда он будет следующим:

9

Теперь запишем то же число (входной аргумент) в десятичном виде:

5681245567

Результат работы нашей программы вычисления длины пусть снова будет записан в десятичном виде. И тогда он будет следующим:

10

Если же у нас программа вычисления длины возвращает результат в шестнадцатеричном виде, то он будет равен предыдущему числу, но записанный следующим образом:

A

Важный случай – стандартная интерпретация, когда ноль записывается как одна «счётная палочка», но мы будем использовать в качестве «счётной палочки» цифру ноль «0». Тогда число 1 записывается так: «00» и т.д. Тогда десять (в десятичной записи 10) будет записано следующим образом:

00000000000

А его длину программа вычисления длины строки пусть запишет в десятичном виде. И результат будет следующим:

11 – это число 11, а не число 1, записанное в «стандартной интерпретации» при помощи цифры «1».

Для математики в теории алгоритмов принципиальным вопросом является размер «входных» данных и время работы алгоритма с данными соответствующего размера. Длина числа, записанного в стандартной интерпретации, экспоненциально велика по сравнению с записью в позиционном (десятичном, например) виде. Поэтому нам необходимо работать с числом-строкой (назовём это так) – которое обладает и свойствами натурального числа (соответствуя аксиомам арифметики), и свойствами строки (обладая длиной).

Очевидно, что просто натуральное число не обладает одновременно обеими ипостасями – как число, и как строка. Это видно на приведённых примерах, где одно и то же натуральное число имеет разное значение своей длинны при разных способах своего представления на ленте МТ. А это значит, что в рамках рекурсивных функций и арифметики невозможно построить соответствие с числами-строками. И натуральному числу не соответствует (однозначным образом) длина записи этого числа. В то время как числа-строки элементарно передаются в качестве входных аргументов МТ и все их строковые характеристики могут быть поучены и использованы МТ. В том числе легко вычисляется их длина. А это значит, что вычислимость по Тьюрингу превосходит (на примере вычисления длины числа-строки) вычислимость рекурсивных функций.

Таким образом «доказательство» одинаковой вычислимость по Тьюрингу и рекурсивных функций является принципиально ошибочным.

Мнение об одинаковой вычислимости по Тьюрингу и рекурсивных функций застопорило развитие математики в невероятно быстро развивающейся и крайне важной для современного общества сфере ИТ-технологий на многие десятилетия – без малого на век. Притом ни математическое сообщество, ни уважаемые и скрупулёзные редакции математических журналов не заметили столь принципиальной и очевидной (при непредвзятом практическом взгляде) ошибки. Поэтому вопрос о том, как это произошло и в чём слабости математического сообщества (у любой системы есть свои слабости – не устранимые до конца) – заслуживает специального рассмотрения.

Если брать самое известное доказательство (см., например, Булос, Джеффри «Вычислимость и логика»), что вычислимая по Тьюрингу функция (алгоритм) может быть переписана в вычислимую рекурсивную функцию, то идея этого доказательства – следующая:

1. Лента МТ разбивается на 2 области: первая – это строка слева от «тележки» МТ, а вторая – это строка от символа под «тележкой» включительно и правее (нюансы, куда отнести положение символа под «тележкой» – к левому или правому «лучу» - не существенны);

2. Левый «луч» рассматривается как число A, соответствующее «левой строке», а правый «луч» рассматривается как число B, соответствующее «правой строке»;

3. Движение «тележки» или запись символа под «тележкой» являются некоторыми изменениями чисел A и/или B.

4. Так как эти изменения происходят в результате работы некоторого алгоритма, то каждый шаг можно формализовать и удаётся сделать это в рамках арифметики.

5. Итоговая формализованная в арифметике функция, которая обеспечивает выполнение всех шагов алгоритма, оказывается искомой рекурсивной функцией, представимой в арифметике.

Данное доказательство не учитывает, что записанные на ленте МТ данные имеют некоторый смысл, и этот смысл не обязательно совпадает с использованным в доказательстве способом рассмотреть их как числа A и B.

И правильный смысл в случае вычисления длины числа-строки оказывается совершенно необходимым. Притом он оказывается необходимым 2 раза – как минимум:

1. Для входного аргумента – чтобы правильно понимать, длину какого числа-строки вычисляет данная программа

2. Для результата – потому что он зависит от того, понимаем мы его как число в шестнадцатеричной записи, или десятичной, или как запись результата в «стандартной интерпретации», например.

Даже результат рекурсивной функции не обладает и не может обладать некоторыми свойствами данных на ленте машины Тьюринга – потому что на ленте МТ число имеет длину (размер) и данное свойство необходимо для работы в рамках теории алгоритмов, например. И при этом оно должно учитываться и в теории вычислимости, потому что длина числа-строки является вычислимым свойством и, таким образом – логика этого свойства должна быть отражена и использована в теории вычислимости.

Но как отличить запись

11

На ленте МТ в качестве числа «один» в стандартной интерпретации (когда в качестве «счётной палочки» используется цифра «1»), от той же записи в качестве числа «одиннадцать» в десятичном виде? Этот вопрос не нуждается в рассмотрении – потому что мы полагаем, что «каким-то» образом известно, какой смысл придаётся соответствующей записи и у нас есть способы, чтобы это узнать.

Математик просто исходит из того, что известно (как-то), с каким числом он имеет дело при данном формате записи. Вот и с числами-строками дело обстоит точно так же. И нет необходимости писать рядом с лентой МТ комментарий «Тут записано число в десятичном виде», например, чтобы математик понял, какое число тут записано.

И если для расчёта длины числа-строки используется один и тот же алгоритм МТ для одного

и того же числа при разных способах записи этого числа, то «перевод» этого алгоритма на язык рекурсивных функций даст разные функции для разных способов записи одного и того же числа. То есть, у нас делается тот выбор между рекурсивными функциями, который не может быть построен на логике отличия натуральных чисел в качестве входного аргумента, потому что натуральное число на входе в разбираемом случае – одно и то же.

Может «вычислимость» надо понимать как наличие функции, способной дать правильный результат для интересующего нас данного (одного) случая и данного (одного) аргумента? Но тогда вообще ничего искать не надо – у нас есть числа и среди них всегда есть правильный ответ о длине числа-строки. А функция, которая в качестве результата даёт «правильное число» всегда найдется – это функция-константа, которая возвращает «нужный» нам для данного случая результат, например. Но до такого выхолащивания понятия «вычислимость» мы не собираемся доходить, надеюсь.

Казалось бы, тип числа и само число можно объединить в «канторовской паре» – это тоже число. И, вроде бы, это путь свести вопрос вычислимости к рекурсивным функциям. Но, как мы увидим в следующе разделе, для этого (если такое вообще возможно) все равно пришлось бы добавлять такую логику (аксиомы), которые вышли бы за рамки арифметики, и мы использовали бы уже не рекурсивные функции с натуральными числами.

К тому же, есть такой частный (но очень важный) случай строк, который невозможно свести к канторовой паре или чему-то подобному «числовому». Этот частный случай мы рассмотрим ещё ниже. Поэтому и рассматривать вопрос о возможности записать логику строк (включая числа-строки) на языке арифметики (выйдя при этом за рамки логики арифметики) не имеет смысла, на мой взгляд.

I. Аксиомы Пеано

Что касается чисел-строк конкретного типа (!), то они подчиняются обычным свойствам арифметики, и тут мы имеем ещё аксиомы теории Пеано или её арифметического расширения. Запишем явным образом аксиомы теории Q, достаточной для представления рекурсивных функций, а затем и её расширение до теории Пеано. Но сразу расставим индексы типа в добавление к обычной записи. А позже разберём, почему это необходимо.

Дж. Булос, Р. Джеффри «Вычислимость и логика» Глава 14. Представимость в Q. Аксиомы теории Q, с добавлением индекса типа U.

(Q_1) $\forall_U i_U \forall_U j_U \, (i'_U = j'_U \Rightarrow i_U = j_U)$

(Q_2) $\forall_U i_U \, 0_U \neq i'_U$

(Q_3) $\forall_U i_U \, (i_U \neq 0_U \Rightarrow \exists_U j_U \, i_U = j'_U)$

(Q_4) $\forall_U i_U \, (i_U + 0_U = i_U)$

(Q_5) $\forall_U i_U \forall_U j_U \, (i_U + j'_U = (i_U + j_U)')$

(Q_6) $\forall_U i_U \, (i_U \times 0_U = 0_U)$

(Q_7) $\forall_U i_U \forall_U j_U \, (i_U \times (j'_U) = (i_U \times j_U) + i_U)$

Замечу, что для знака умножения использован символ «$\times$», потому что символ «$\cdot$» мы будем использовать для обозначения конкатенации.

Обозначение $\forall_U \; i_U$ означает, что в качестве возможных значений связанной переменной i_U выступают только все значения заданного типа. «Забегая вперед», отметим, что логика такого «типированного» квантора общности:

$\forall_U \; i_U \; \mathrm{A}(i_U)$

Эквивалентна следующей формуле:

$\forall \; i \, (\ominus_U = \mathrm{s}(i, 0) \Rightarrow \mathrm{A}(i))$

Дальше мы разберем и функцию взятие символа $\mathrm{s}(a, j)$ с произвольного места j строки a, и пустую строку заданного типа $\ominus_U$.

Аналогично, формула $\exists_U \; i_U \; \mathrm{A}(i_U)$ эквивалентна формуле $\exists \; i \, (\ominus_U = \mathrm{s}(i, 0) \wedge \mathrm{A}(i))$, что соответствует записи квантора существования $\exists \; i \ldots$ в виде $\neg \forall \; i \neg \ldots$. Действительно:

$\exists \; i \, (\ominus_U = \mathrm{s}(i, 0) \wedge \mathrm{A}(i))$ – эквивалентно:

$\neg \forall \; i \, \neg (\ominus_U = \mathrm{s}(i, 0) \wedge \mathrm{A}(i))$ – эквивалентно:

$\neg \forall \; i \, (\ominus_U = \mathrm{s}(i, 0) \Rightarrow \neg \, \mathrm{A}(i))$ – эквивалентно:

$\neg \forall_U \; i_U \neg \, \mathrm{A}(i_U)$ – эквивалентно:

$\exists_U \; i_U \; \mathrm{A}(i_U)$.

В той же книге: Глава 15 Неразрешимость, неопределимость и полнота. Добавление аксиом индукции к теории Q и переход к теории Пеано – «теории Z». Но тоже с добавлением индекса типа.

$\mathrm{A}(0_U) \wedge \forall_U i_U \, (\mathrm{A}(i_U) \Rightarrow \mathrm{A}(i'_U)) \Rightarrow \forall_U i_U \, (\mathrm{A}(i_U))$ – все формулы такого вида на языке теории строк, но с навешиванием на такие формулы типированного квантора общности $\forall_U i_U$.

Разумеется, для теории строк формула $\mathrm{A}(i_U)$ понимается как формула, которая может зависеть и от разных строковых аргументов, которые не являются числами-строками.

При этом аксиому Q_3 можно исключить, так как она выводится из аксиомы индукции и аксиомы Q_2.

Аксиомы Пеано тут – это аксиомы, которые выполнены для не просто чисел, но для строк со свойствами чисел определённого типа, на который указывает индекс U.

Для обозначения предметных переменных (и строк, и чисел-строк) будем использовать все переменные из прописных букв, кроме i, j, k, l, m, n – которые оставим для чисел-строк. И кроме переменной q, которую оставим для значений чисел-строк и пустой строки $\ominus$. Если для индекса типа строки используется число, то его тоже можно использовать как переменную для числа-строки. Обычно для индекса типа в аксиомах будем использовать индексы U, K (прописные). Конечно, эти индексы типа могут приписываться к переменной – когда тип строки для этой переменной известен.

Индекс типа числа-строки нам нужен для того, чтобы связать то, как числа-строки (и их аксиомы арифметики) данного типа записаны при помощи символов алфавита строки данного типа и операций конкатенации. Понятно, что десятичная запись будет отличаться от записи числа в стандартной интерпретации, например.

II. Связь чисел и строк

Теперь, сохраняя те же условности обозначения для строк, имеющих свойства чисел (для переменных со значением число-строка мы используем оговоренные ранее буквы) зададим связь между алфавитом для строк и числами.

Чтобы придать числам свойства строк, достаточно задать строку, которая будет 0 (нулём) и задать операцию добавления единицы (функцию следования или инкремент) в терминах строк. После этого остальные арифметические операции в терминах строк можно легко вывести на основании свойств строк и аксиом арифметики Пеано.

Вот нужная для «превращения» чисел ещё и в строки аксиома, которую можно при желании разбить на отдельные аксиомы – отдельная аксиома из каждого члена следующей конъюнкции:

$(i_U \leq U_U \Rightarrow i_U = \mathrm{e_U}(i_U))$
$\wedge(i_U < U_U \Rightarrow \mathrm{e_U}(i_U)' = \mathrm{e_U}(i'_U))$
$\wedge(0_U = U_U \Rightarrow 0'_U = 0_U \cdot \mathrm{e_U}(0_U))$
$\wedge(0_U < U_U \Rightarrow \mathrm{e_U}(U_U)' = (0'_U) \cdot \mathrm{e_U}(0_U))$
$\wedge((k_U \neq 0_U \wedge i_U < U_U) \Rightarrow (k_U \cdot \mathrm{e_U}(i_U))' = k_U \cdot \mathrm{e_U}(i'_U))$
$\wedge(k_U \neq 0_U \Rightarrow (k_U \cdot \mathrm{e_U}(U_U))' = (k'_U) \cdot \mathrm{e_U}(0_U))$

Пояснения:

1. Будем считать алфавит $\mathrm{e_U}(i_U)$ чисел-строк типа U состоящим из одних только цифр, притом ноль – первый элемент $\mathrm{e_U}(0_U)$, а самая большая цифра – последний элемент $\mathrm{e_U}(U_U)$.

2. Тип в рассматриваемой сейчас аксиоматизации соответствует $(U+1)$-ичной системе счисления для $U > 0$, и записи числа в стандартной интерпретации для $U = 0$. Например, для $U = 9$ получается аксиома для увеличения на 1 при десятичной записи числа.

3. Дальнейшие пункты можно пропустить, потому что после них будет пояснение на примерах, и там будет сказано о проблеме с выходом логики строк за рамки арифметики.

4. Первый член конъюнкции аксиомы $(i_U \leq U_U \Rightarrow i_U = \mathrm{e_U}(i_U))$ как раз говорит о том, что числа-строки типа U от нуля до числа, равного U, равны соответствующим цифрам (символам).

5. Второй член конъюнкции $(i_U < U_U \Rightarrow \mathrm{e_U}(i_U)' = \mathrm{e_U}(i'_U))$ аксиомы говорит о том, что увеличение на 1 числа, меньшего U (а такое число записано одной цифрой) даёт число, которое равно следующей цифре.

6. Третий член конъюнкции $(0_U = U_U \Rightarrow 0'_U = 0_U \cdot \mathrm{e_U}(0_U))$ аксиомы говорит о том, что увеличение на 1 числа, записанного в стандартной интерпретации ($0_U = U_U$) означает всего лишь приписывание ещё одной цифры.

7. Четвёртый член конъюнкции $(0_U < U_U \Rightarrow \mathrm{e_U}(U_U)' = (0'_U) \cdot \mathrm{e_U}(0_U))$ аксиомы говорит о том, что увеличение на 1 числа, равного максимальной цифре (если речь не о стандартной интерпретации – поэтому условие $0_U < U_U$), даёт число-строку «10», условно говоря.

8. Пятый член конъюнкции $((k_U \neq 0_U \wedge i_U < U_U) \Rightarrow (k_U \cdot \mathrm{e_U}(i_U))' = k_U \cdot \mathrm{e_U}(i'_U))$ аксиомы говорит о том, что увеличение на 1 числа, которое состоит из нескольких цифр и не начинается на ноль, и чья последняя цифра с номером i меньше U, означает то же число-строку, но последняя цифра будет с номером $i+1$ вместо i.

9. Шестой член конъюнкции $(k_U \neq 0_U \Rightarrow (k_U \cdot \mathrm{e_U}(U_U))' = (k'_U) \cdot \mathrm{e_U}(0_U))$ аксиомы говорит о том, что увеличение на 1 числа, которое состоит из нескольких цифр и не начинается на ноль, и чья последняя цифра с номером U – означает такое число-строку, где вся часть до последней цифры (не включая её) заменяется на эту же часть, увеличенную на 1, а последняя цифра будет с номером 0 вместо U.

Но разве записанное утверждение (аксиома) не может соответствовать логике арифметики без какого-либо специального добавления логики строк? Пусть у нас записано десятичное число-строка:

56789266

И пусть нам надо дописать к нему (в его конец) цифру 8. Мы можем сделать это и в арифметике, потому что:

$56789266 \cdot 8 = 567892668 = 56789266 \times 10 + 8$

То есть, конкатенация исходного числа и дополнительной цифры в данном случае – это просто умножение исходного числа на 10 и добавление дополнительной цифры как числа. А функция получения цифры по её номеру в алфавите десятичных цифр $\mathrm{e}_9(i_9)$ является в данном случае тождественной функцией при входном аргументе от 0 до 9. А большего нам, вроде, не требуется для десятичной записи.

Попытка сведения логики строк к арифметике – притом успешная, якобы – является корнем принципиальной ошибки в теории вычислимости. В отдельном приложении ниже представлен разбор того, как этой ошибке удавалось казаться истиной, и как она «доказывается». Лёгкая демонстрация выше содержит существенные неувязки при детальном разборе, но их удается устранить, притом раньше казалось, что удается устранить все «неувязки». А подробности изложены в Приложении 1.

Тут же сразу перейдём к сути ошибки о якобы «одинаковости» логики строк и арифметики, которая проясняется при сравнении формул:

$\mathrm{s}(15_9, 1_9) = 1_9$ и $\mathrm{s}(F_{15}, 1_9) = F_{15}$

Обе формулы – истинны, обе формулы сообщают о первом символе числа-строки. Притом число – одно и то же. Это число 15 (если записывать в десятичном виде). Но строки – разные. И результат работы функции – а это одна и та же функция! – получается разный. Разумеется, если бы мы рассматривали арифметическую функцию, то её результат для одних и тех же натуральных чисел в качестве «входных» аргументов был бы одинаковый. А с точки зрения арифметики мы имеем дело с одними и теми же числами. Значит, функция взятия символа из строки a с позиции i: $\mathrm{s}(a, i)$ – не является арифметической.

Таким образом, информация о типе должна быть частью свойств такого объекта, как число-строка. Иначе функция взятия символа с нужной позиции числа-строки не будет работать правильно. И это свойство (тип) не является арифметическим.

2 Арифметическая индукция не соответствует логике строк

Хотя рассуждение в конце предыдущего раздела и доказывает, что логика строк выходит за рамки арифметики, но важно доказать, что некоторые привычные математикам «законы» оказываются ложными для строк. В частности – аксиома арифметической индукции не может быть частью логики строк.

Да, в «модернизированном виде» аксиома индукции верна – для чисел-строк заданного типа, например. Но не для произвольных строк.

Запишем аксиому индукции из арифметики, которая приводилась выше, но тогда в неё были добавлены индексы типа. Теперь приведём её в обычном для арифметики виде:

$\mathrm{A}(0) \wedge \forall i\,(\mathrm{A}(i) \Rightarrow \mathrm{A}(i')) \Rightarrow \forall i\,(\mathrm{A}(i))$

Воспользуемся этой аксиомой индукции, чтобы «доказать», что длина числа-строки равна самому числу плюс один – начиная с числа-строки, равному 2. А число 2 – это число ноль, увеличенное на 1 два раза. Поэтому в качестве формулы $\mathrm{A}(i)$ будем рассматривать равенство:

$\mathrm{l}_9(i'') = (\mathrm{e}_9(0_9))\,i'''$

1. Левая часть равенства – функция, вычисляющая длину строки (в том числе длину числа-строки). Но мы в начале статьи увидели, что результат у подобной функции может иметь разный тип – потому что длина может быть представлена в десятичном, шестнадцатеричном виде, в стандартной интерпретации и т.д. Пусть в нашем случае результат будет иметь тип десятичного числа.

2. Для того, чтобы сравнить (на предмет равенства) длину числа-строки i'' с величиной числа i''' нам надо, чтобы у чисел-строк был один и тот же тип. И здесь нам необходимо прибегнуть к преобразованию типов. К той операции, с которой постоянно приходится считаться в программировании, но которой до сих пор не было в теории вычислимости из-за иллюзии «арифметичности» этой теории. Условимся, что для преобразования числа-строки i_K типа K к такому же числу i_U, записанному как число-строка типа U, мы будем использовать следующую операцию преобразования типа чисел-строк:

$(\mathrm{e}_U(0_U))\,i_K$

Операция преобразования типов имеет вид, аналогичный явному преобразованию типов в языке программирования Си. Только вместо имени типа, который пишется в языке Си в качестве левого операнда (который в круглых скобках), мы пишем первый (точнее, с номером ноль) элемент в алфавите нужного типа. И это всегда непустой символ.

Свойства этой операции преобразования чисел-строк «с сохранением числа» такие:

$0_U = (\mathrm{e}_U(0_U))\,0_K$

$i_U = (\mathrm{e}_U(0_U))\,j_K \Rightarrow i'_U = (\mathrm{e}_U(0_U))\,(j'_K)$

Мы можем преобразовывать не только числа-строки между собой. У строки может быть такой тип (это мы ещё обсудим), который не допускает чисел-строк. Но мы можем преобразовать его «с сохранением символов» в другой тип, который допускает числа-строки и использует алфавит с тем же количеством символов, что и исходный тип строки. Вот для такого преобразования (когда номера символов в алфавите не меняются, а тип числа-строки преобразуется) мы будем использовать в качестве левого операнда пустую строку (пустой символ) $\ominus_U$ того типа, к которому надо

преобразовать строку. Например:

$10_{15} = (\ominus_{15})\, 10_9$

Число, которое записано слева от знака равенства – это 16 (если записать его в десятичном виде), но записанное в шестнадцатеричном виде. А справа от знака равенства, где исходное (до преобразования) – число-строка 10 в десятичном виде. Разумеется, преобразование «с сохранением числа» будет таким:

$A_{15} = (e_{15}(0_{15}))\, 10_9$, Заметим, что A тут – это цифра шестнадцатеричной записи, соответствующая числу 10 десятичной записи.

Если у нужного нам типа U нет чисел-строк, то нет и числа 0_U, поэтому записать ошибочную формулу для преобразования к числам несуществующего типа при помощи выражения – $\mathrm{e_U}(0_U)$ – не получится. Тогда возможно только преобразование с сохранением номеров символов:

$10_U = (\ominus_U)\, 10_9$

Будем считать, что у десятичной цифры 0 (Ноль) номер в алфавите – тоже ноль. То есть:

$0_9 = \mathrm{e}_9(0_9)$

Тогда наш вариант формулы $\mathrm{A}(i)$:

$\mathrm{l}_9(i'') = (\mathrm{e}_9(0_9))\, i'''$

перепишем так:

$\mathrm{l}_9(i'') = (0_9)\, i'''$

В качестве начала индукции $\mathrm{A}(0)$ возьмём разобранную сейчас формулу с нулём, имеющем тип стандартной интерпретации, а сравнивать будем преобразованные к десятичному виду стороны равенства – для единообразия по всей формуле индукции:

$\mathrm{l}_9(0_0'') = (0_9)\, 0_0'''$

Напоминаю, что в приведенной выше аксиоме связи чисел и строк типами строк являются двоичная, троичная, . . . , десятичная и далее системы счисления с индексами типа, равными соответственно 1, 2, . . . , 9 и далее. А, кроме того, для индекса типа, равного нулю, типом числа-строки является стандартная интерпретация арифметики.

Тогда формула арифметической индукции для интересующего нас случая окажется следующей:

$\mathrm{l}_9(0_0'') = 3_9 \wedge \forall i\, (\mathrm{l}_9(i'') = (0_9)\, i''' \Rightarrow \mathrm{l}_9(i''') = (0_9)\, i'''') \Rightarrow \forall i\ \mathrm{l}_9(i'') = (0_9)\, i'''$

Сразу «расставим точки над i»: мы исследуем логику строк не формально на первом этапе, к которому относится данная статья. Мы ставим реальность выше теорий – что не характерно для математики, в которой работа происходит, практически всегда, с готовыми абстракциями. Но попытка использовать шаблонный для математиков метод работы (когда начинают с аксиом и подобных «абстракций»), чтобы начать работу со строками, привела к тяжелым ошибкам и десятилетиям неспособности не то, что преодолеть эти ошибки, но даже увидеть их. Поэтому будем делать так, как надо для данного случая, а не так, как привычно для работы математиков. Поэтому реальность и опора на практику будут пока что в приоритете перед теорией.

На практике у нас могут быть разные способы записи чисел при помощи строк. Не обязательно в аксиоме о связи чисел и строк рассматривать все варианты позиционной системы счисления, например. В Приложении 1 приведён иной вариант аксиомы связи чисел и строк, чем используе-

мый в данном разделе. И корректных вариантов аксиомы для связи чисел и строк – бесконечное количество. Мы доказываем ошибочность индукции для одного из таких вариантов аксиомы и для одного лишь способа применения индукции. Этого достаточно, чтобы доказать, что аксиома арифметической индукции не является истиной в рамках логики строк.

Итак, возвращаясь к интересующему нас случаю применения индукции – мы обнаруживаем, что посылка этой формулы является истинной. В самом деле:

$\mathrm{l}_9(0_0'') = 3_9$

Истинность этого равенства видна по факту того, что число 2 (Два в десятичном представлении тут), записанное в стандартной интерпретации имеет следующий вид, если использовать для его записи цифры ноль:

000

Непосредственно видим, что длина данного числа-строки равна 3 (Три в десятичном представлении тут).

На естественный «из общих соображений» вопрос, почему индукция для данного случае не «улучшается» в плане добавления квантора общности по «всем нулям» вместо конкретного нуля в равенстве $\mathrm{l}_9(0_0'') = 3_9$, напоминаю:

Мы не ищем в данном случае истинное утверждение, а демонстрируем ложность стандартного утверждения индукции на конкретном примере. Для записи индукции есть стандартная формула и её ошибочность хотя бы в одном случае будет означать (и означает), что перед нами – не арифметический случай, и арифметическая индукция в рассматриваемой теории не является ни истинным утверждением, ни – тем более – аксиомой.

Что же касается «улучшения индукции», то такие аксиомы у нас уже есть – для каждого конкретного типа числа-строки. Но все вместе они демонстрируют такую логику, которая оказывается несовместимой с «обычной» арифметической индукцией, что мы сейчас доказываем.

Теперь рассмотрим оставшуюся часть посылки в разбираемой нами индукции:

$\forall i\,(\mathrm{l}_9(i'') = (0_9)\,i''' \Rightarrow \mathrm{l}_9(i''') = (0_9)\,i'''')$

У переменной i могут быть значения самых разных типов. Мы исходим из того, что все типы перечислены в аксиоме связи чисел и строк. И при построении аксиоматики нам будет необходимо формализовать это соответствие типов в аксиому. Сейчас же, рассуждая не формально:

1. Для случая, когда типом числа-строки i является стандартная интерпретация (i_0) мы имеем истинную импликацию – с истинной посылкой и заключением. Потому что длина числа в стандартной интерпретации действительно на 1 больше величины этого числа – это мы видели выше.

2. Для случая, когда типом числа-строки i является позиционная запись $(i_U, U > 0)$ мы имеем истинную импликацию из-за ложности её посылки. В самом деле – для числа 0 его длина во всех позиционных системах счисления равна 1, как и для стандартной интерпретации. Но нас в разбираемом случае не интересует 0. А уже для числа 1 длина любой позиционной записи остается равной 1, что меньше, чем при стандартной интерпретации. И дальше при увеличении числа на 1 происходит (иногда) увеличение длины его позиционной записи (любой позиционной записи) на один – но не больше. А для стандартной интерпретации увеличение длины числа-строки на 1

происходит при каждом увеличении числа на 1. Поэтому длина позиционной записи числа-строки, начиная от числа 1, никогда не догонит длину стандартной интерпретации того же числа.

Поэтому для позиционной записи равенство $\mathrm{l}_9(i'') = (0_9)\, i'''$ не выполняется никогда.

А это значит, что все возможные импликации под квантором общности в разбираемом утверждении истинны, поэтому и само выражение с квантором общности – истинное:

$\forall i\, (\mathrm{l}_9(i'') = (0_9)\, i''' \Rightarrow \mathrm{l}_9(i''') = (0_9)\, i'''')$

Таким образом, оба члена конъюнкции в посылке разбираемого случая арифметической индукции – истинные. Значит – если исходить из истинности арифметической индукции – истинным должно быть и заключение этого случая индукции:

$\forall i\ \mathrm{l}_9(i'') = (0_9)\, i'''$

Но данное утверждение – ложное, потому что мы только что убедились, что для позиционной записи равенство $\mathrm{l}_9(i'') = (0_9)\, i'''$ не выполняется никогда, а типы чисел-строк, соответствующие позиционной записи, тоже являются значениями связанной квантором общности переменной i.

Таким образом, предположение об истинности арифметической индукции для строк является ошибочным.

И, следовательно, рассматриваемые в статье String theory теории строк в самой своей аксиоматике опирались на ложные утверждения. Не удивительно, что при работе с такими теориями были сделаны ошибочные выводы об «арифметичности» логики строк. Что «подтвердило» признанное ранее истинным (но ошибочное на самом деле) утверждение об одинаковой вычислимости по Тьюрингу и рекурсивных функций.

Кроме того, никакие попытки представить строки через «канторовские пары», в которых «упакованы» сразу число и его тип – тоже не позволят остаться в рамках арифметики при добавлении туда необходимой для строк логики. Потому что к таким «канторовским парам» (или их аналогам) окажется неприменимой арифметическая индукция, а это значит, что их логика вышла за рамки арифметики.

3 О языке теории строк, типах и аксиомах о типах

Понятно, что тип строки зависит не только от количества символов в алфавите, но и от того, какие числа-строки этому типу соответствуют. Например, в алфавите из 10 цифр от 0 до 9. Для одного типа строк символы 0 и 9 обозначают цифры 0 и 9 в десятичной записи числа-строки, в другом типе они обозначают цифры 9 и 0 соответственно.

Более того, в начале вычисления входной аргумент может не содержать информации о том, каким числам-строкам он соответствует. Поясню на практическом примере, как в программном тексте языка Си записываются числа в разных системах счисления:

0b101100101 – двоичная;

076205 – восьмеричная (начинается с ведущего нуля);

0x3FA7B33 – шестнадцатеричная.

Как видим тут тип числа-строки задаётся «префиксом» перед числом-строкой. Допустим, например, что входными аргументами программы являются 2 аргумента – префикс и строка, которая будет истолкована как число-строка того типа, который соответствует префиксу.

Ещё один вариант – бесконечный аргумент, состоящий из цифр и начинающийся не с цифры ноль. Если программа использует некоторую начальную часть такого аргумента и останавливается, то она «не в курсе», конечным был этот аргумент или же нет – и «можно» ли было считать его десятичной записью какого-то числа. Да и для некоторых строк вообще нет необходимости разбираться – какого они типа в плане соответствующих им чисел. Например, когда строка состоит не из одних цифр своего алфавита и не из признаков типа (признаками типа может быть префикс, например, как мы видели выше).

Есть ещё одна принципиальная причина, при наличии которой строка не может считаться числом. Но раскрывать тут детали не буду – так как они относятся к теории алгоритмов, а не вычислимости.

Однако я должен тут сказать, что первоначально идею дать убедительное (пусть и не строго формальное) доказательство невозможности индукции для строк высказал Сергей Витальевич Знаменский в нашей переписке о теории строк применительно к теории алгоритмов. Фактически такое предложение мало чем отличалось от готового доказательства при накопленных к тому моменту переписки идеях. И это означало, что уже тогда было доказано, что строка не может в общем случае считаться числом по причинам из теории алгоритмов, свидетельствующих о нарушении индукции – о чем и упомянуто в предыдущем абзаце.

Другое дело, что в острой (но этически приемлемой) дискуссии 14-15 февраля 2022 г. он возражал против запрета на индукцию в отношении строковых аргументов, которого придерживался я ещё до предложения доказать невозможность индукции в общем случае для строк. То есть, в какой-то степени он отошел от своей же позиции и предложенного доказательства нарушения индукции – хотя всё равно остался в определенной степени прав (как теперь понятно), потому что возможность или невозможность индукции (не арифметической) для строк зависит от типа строки – от того, имеются ли числа-строки для этого типа строк. В тот момент ещё не было понимания о наличии у строк такого свойства как тип, и получалось «противоречие» аналогичное вопросу

«волна или частица?» в квантовой механике до появления уравнения Шредингера.

При совместном творчестве нередко кто-то вносит свой вклад на одном этапе, а затем эта роль переходит другому и т.д. И в следующее полугодие после дискуссии 14-15 февраля я уже в одиночестве обнаруживал и исследовал типы строк и принципиальную ошибку в теории вычислимости. Но первое доказательство о нарушении индукции при работе со строками было не то, которое приведено в данной статье, а то, которое возникло в переписке с Сергеем Витальевичем по его инициативе.

Впрочем, возвратимся к типу строк без соответствующих им чисел-строк. Будем использовать для них обозначения типа, которые начинаются на S. И для них истинным будет утверждение, что никакая строка данного типа не равна нулю никакого типа. Например:

$a_{S9} \neq 0_U$

Разумеется, из-за отсутствия нуля нет и никаких чисел-строк данного типа. Но в процессе работы программы подобные строки могут быть конвертированы в строки иного типа, который допускает числа-строки. И на такое преобразование на практике зачастую потребуется время. Мы тут не рассматриваем алгоритмы, однако функция конвертации, соответствующая алгоритму преобразования типа, у нас в теории теперь есть.

Все символы в данной строке – одного и того же типа. Конечно, если мы «вытащили» символ и стали использовать его как-то отдельно для создания других строк и чисел-строк, то после всяких преобразований мы можем перейти к другому типу строки. Но тогда нам необходимо указать это явно при помощи оператора преобразования типа. А до тех пор, чтобы сохранять информацию об исходном типе (пусть неизвестно каком конкретно) – для производных «частей» строки можно писать индекс исходного типа при помощи имени переменной, содержащей исходную строку. Например:

$myvar = myvar_{myvar}$

$\ominus_{myvar} = \mathrm{s}(myvar, 0)$

$a_{myvar} = \mathrm{s}(myvar, 2)$

Если смотреть практически, то тип – это информация не «чисто» о значении, которое «содержится» в переменной, а это информация о самой переменной. Потому что тип есть там, где мы знаем, как нам интерпретировать значение – то есть, какие правила к нему применять. Например, на какой именно ленте МТ находятся эти данные и на каком этапе исполнения программы. А правила – они относятся к каждому элементу информации в «переменной», то есть – к каждому символу. И правила – это «вещь» целостная и поэтому они относятся к каждому символу в полном объёме и указываются, например, индексом типа.

Конечно, когда мы соединяем символы в одну строку, то тип нам нужен только в момент появления этой строки. Но, поскольку этот тип один на всех, то результат соединения символов с «нужным» типом у каждого символа в нужную строку того же типа ничем не отличается от соединения их сначала – без типа – и задание типа после – одного ко всем и каждому символу. Но задавать тип каждому «кирпичу» заранее – технически проще, чем иметь промежуточный этап, где тип «строительных конструкций» не задан.

Мы заранее знаем как минимум алфавит в части количества символов в нем для «строительных конструкций», и то, что все символы в итоге окажутся одного типа. А если не понятен сразу вопрос о типе числа-строки или даже о том, есть ли соответствующие этому типу числа-строки, то мы можем на промежуточном этапе построения строки дать символам тип «без чисел-строк», а когда, и, если необходимость в числе-строке возникнет – конвертировать строку к нужному типу U при помощи левого операнда $\ominus_U$.

Разумеется, если нам не требуется сохранять информацию о типе значения переменной в её имени, то мы можем давать переменной название без индекса типа, например:

$a = \mathrm{s}(myvar, 2)$

Необходимо пояснить подробнее, что за функция $\mathrm{s}(a, i)$ использована в последних формулах и почему числа в ней – без индекса типа.

Функция $\mathrm{s}(a, i)$ возвращает символ с i-го места строки a, притом непустые символы нумеруются, начиная с 1. А на нулевом месте всегда находится пустой символ (пустая строка) $\mathrm{s}(a, 0) = \ominus_U$, имеющая тот же тип U, который имеет вся строка.

Если на ненулевом месте встретиться пустая строка, то для всех оставшихся мест данная функция тоже возвращает пустую строку:

$i \neq 0 \wedge \mathrm{s}(a, i) = \mathrm{s}(a, 0) \wedge j > i \Rightarrow \mathrm{s}(a, j) = \mathrm{s}(a, 0)$, вместо $\mathrm{s}(a, 0)$ можно использовать и, конечно.

Получается, что длина строки $\mathrm{l_U}(a)$ (если результат функции возвращается в виде числа-строки типа U) равна номеру последнего не пустого символа. А если строка пустая, то нулю. И, можно договорится, что для бесконечной строки длина будет равна пустой строке того типа, который выбран для получения результата функции длины:

$(\forall i (i \neq 0 \Rightarrow \mathrm{s}(a, i) \neq \ominus_a)) \Rightarrow \mathrm{l_U}(a) = \ominus_U$

Но для такого определения (точнее сказать – для части определения) функции длины надо, чтобы было истинным утверждение, что пустая строка типа U (тип числа-строки результата) не равна никакому числу-строке типа U:

$\forall i_U \, (\mathrm{s}(a, 0) \neq i_U)$

Во всех практических реализациях строк данное условие выполнено. То есть – пустая строка никогда не считается никаким числом.

Как видим, довольно удобно, что на нулевом месте в строке находится пустая строка. Нам постоянно приходится её получать, а если бы нумерация непустых символов начиналась с нуля, то пришлось бы использовать более громоздкую конструкцию вроде $\mathrm{s}(\mathrm{s}(a, 0), 1)$. Не видно, зачем мириться с подобным усложнением.

Теперь разберем, почему мы используем для нумерации мест в функции $\mathrm{s}(a, i)$ аргумент i без указания типа.

Дело в том, что результат функции $\mathrm{s}(a, i)$ зависит от номера места i исключительно как от натурального числа, которое соответствует данному числу-строке i. Можно записать данную функцию эквивалентным образом – следующим, например:

$\mathrm{s}(a, (\mathrm{e}_9(0_9)) \, i_U)$

То есть, число-строка любого типа приводится к стандартному типу (десятичному в приведенном

примере) и его тип никак не влияет на результат функции $\mathrm{s}(a,i)$.

Очевидно, что для «чисто» числовых аргументов в функциях, результат которых не зависят от типа этих аргументов, верной оказывается арифметическая индукция. А вот для строковых аргументов (включая случай зависимости результата от типа числа-строки) арифметическая индукция уже не является истинной, что мы видели на примере $\mathrm{l}_9(i_U)$.

В качестве собственно типа (имени типа «отдельно» от символа) можно использовать пустую строку соответствующего типа. Конечно, имя тут не совсем «отдельно», но конструкция:

$\ominus_{myvar}$

Несет только одну «переменную часть» – индекс типа по имени исходной переменной.

Чтобы сравнивать типы – равны они или нет – достаточно сравнивать пустые строки соответствующих типов с индексами их типов.

Тип любой строки соответствует одному из типов, которые используются в записи аксиом (ни один тип ни одной строки не является «беспризорным»). И это при том, что ни одна теория не может «перечислить» все типы – ведь очевидно, что вопрос о корректности произвольного способа записи чисел при помощи символов не является разрешимым. Но, если в нашей теории используются только типы 1 и 2, например, тогда должно быть верным:

$\mathrm{s}(a,0) = \ominus_1 \vee \mathrm{s}(a,0) = \ominus_2$

Но у нас есть «алфавиты» — это способ классификаций тех правил, которые соответствуют разным переменным данной теории для строк, а обозначаются разные наборы таких правил при помощи индекса типа. Для получения символа номер j в алфавите с типом U есть функция:

$\mathrm{e_U}(j)$

Но, поскольку тип полностью определяется пустой строкой того же типа, то обе записи для этой функции с разных сторон следующего равенства будем считать тождественными:

$\mathrm{e_U}(j) = \mathrm{e}(\ominus_U, j)$

Тогда связь между переменными со строковыми значениями и алфавитами следующая:

$\forall a \exists \ominus_U \forall i \exists j \, (\mathrm{s}(a,i) = \mathrm{e}(\ominus_U, j) \wedge \mathrm{s}(a,0) = \ominus_U)$

Из последней формулы легко вывести «почти эквивалентный» ей короткий вариант, но с потерей наглядности в отношении индекса типа:

$\forall a \forall i \exists j \, (\mathrm{s}(a,i) = \mathrm{e}(\mathrm{s}(a,0), j))$

И данная формула – в отличие от предыдущей – не содержит информацию о том, что на нулевом месте значения переменной всегда находится пустая строка.

В отличие от функции $\mathrm{s}(a,i)$ функция $\mathrm{e}(\ominus_U, i)$ начинает нумерацию i не пустых символов с номера ноль. В данном случае мы следуем практике – все кодовые таблицы начинают нумерацию символов с нуля. И нет резонов отклоняться от этого стандарта и нумеровать не пустые символы с единицы, как в случае функции $\mathrm{s}(a,i)$. К тому же нумерация в $\mathrm{s}(a,i)$ не отходит от практики – в разных языках программирования в качестве начала нумерации символов в строке используются как 1, так и 0.

Поскольку алфавиты идут «в комплекте» с типированными аксиомами арифметики и правила (а это и есть тип) одинаковые для всех символов данного алфавита, то должно быть верным:

$\mathrm{e_U}(i) = \mathrm{e_K}(j) \Rightarrow \forall j\,(\mathrm{e_U}(j) = \mathrm{e_K}(j) \wedge j_U = j_K)$, если в первом алфавите есть символ, равный символу из второго алфавита, то эти алфавиты полностью одинаковы как по символам, так и по соответствующим этим алфавитам числам-строкам.

Разумеется, строки (и числа-строки) равны тогда и только тогда, когда попарно равны их символы, расположенные на одних и тех же местах своих строк. Поэтому равными могут быть только строки (и числа-строки) с равными алфавитами.

Для алфавитов должны быть добавлены аксиомы о том, что символы в алфавите данного типа не повторяются, из чего можно вывести, что для предыдущей формулы окажется верным:

$i = j$

Но тут есть и опасность противоречия – если ошибочно аксиоматизировать равенство таких символов, функционал алфавитов которых отличается.

Напоминаю, что когда в формулах числовые переменные указаны без индекса типа, то это означает, что результат зависит только от натуральных чисел, соответствующих этим числам-строкам. Поэтому и нужды в указании типа нет.

Условимся, что когда кончаются не пустые символы в алфавите, то функция $\mathrm{e_U}(i)$ возвращает для всех следующих номеров пустую строку того же типа. А количество непустых символов в алфавите будем обозначать функцией $\mathrm{E_K}(\ominus_U)$. И это же – номер первого пустого символа – ведь нумерация символов в алфавите у нас начинается с нуля. Напоминаю, что если результатом функции является число-строка, то необходимо указать зависимость от типа. Разумеется, этот тип может быть любым, допускающим числа-строки, потому что у нас есть операция конвертации чисел-строк одного произвольного типа к числам-строкам любого другого произвольного типа.

При выполнении перечисленных условий должно быть верным:

$i_9 \geq \mathrm{E_9}(\ominus_U) \Rightarrow \mathrm{e_U}(i_9) = \ominus_U$

А что такое равные алфавиты – одинаковое количество символов и одинаковые числа (если сравнивать их как натуральные числа), которые записаны символами с одинаковыми (в своих алфавитах) номерами?

Нет, не обязательно. Потому что помимо того, что к алфавиту с данным типом «привязаны» типированные аксиомы арифметики и привязана связь между данным алфавитом и типированными аксиомами арифметики (если для данного типа есть числа-строки) – возможны и другие «привязки». Например – графическое изображение данного символа. И утверждение, что каждый индекс типа создаёт уникальный набор правил – возможная аксиома.

Но возможна и такая аксиома, в соответствии с которой правила полностью заданы размером алфавита и одинаковой записью одинаковых чисел (натуральных) через числа-строки. Тогда будет верно:

$[\forall i_U \exists j_K \forall n \exists m\,(\mathrm{s}(i_U, n) = \mathrm{e_U}(m) \wedge \mathrm{s}(j_K, n) = \mathrm{e_U}(m) \wedge i_U = (\mathrm{e_U}(0_U))\,j_K)]$
$\wedge [\forall i_K \exists j_U \forall n \exists m\,(\mathrm{s}(i_U, n) = \mathrm{e_U}(m) \wedge \mathrm{s}(j_K, n) = \mathrm{e_U}(m) \wedge i_K = (\mathrm{e_K}(0_K))\,j_U)]$
$\wedge\, \mathrm{E_9}(\ominus_U) = \mathrm{E_9}(\ominus_K)$
$\Rightarrow$
$\mathrm{e_U}(0) = \mathrm{e_K}(0)$

Два первых «симметричных» члена конъюнкции предусмотрены на случай, если один алфавит имеет, а другой – не имеет соответствующие ему числа-строки.

Например, тип U - не имеет. Тогда первый конъюнктивный член оказывается истинным, потому что использует число i_U и это неявно означает импликацию (под квантором общности), в посылке которой стоит условие «если данная строка является числом- строкой». Если чисел-строк нет, то эта импликация всегда истинная.

А теперь смотрим на второй конъюнктивный член, считая, что тип K – имеет соответствующие ему числа строки. Тогда для него не найдется (не существует) соответствующего числа-строки типа U. И второй конъюнктивный член окажется ложным. А алфавиты окажутся не равными.

В заключении приведенной формулы стоит равенство символов номер ноль из обеих алфавитов, что означает равенство этих алфавитов, как мы видели выше.

До настоящего момента мы разбирали только такие пригодные для всех типов строк формулы, которые если и создают строки, то это строки из одного символа.

Конечно, есть формулы для связи строк с числами, но они не годятся для всех типов строк – не все символы используются в построении чисел-строк, к тому же есть такие типы строк, для которых чисел-строк того же типа не существует.

И если отвлечься от чисел-строк, то пока не было формул, которые позволяли бы доказать, что существуют строки более, чем из одного символа. А если и существуют, то сколько в них символов? Может быть, в них (строках, которые состоят больше, чем их одного символа) только чётное количество символов (не пустых)? Или только нечётное? Или существуют только строки с ещё более «затейливыми» условиями на свое существование?

Чтобы в теории заведомо существовали строки любого конечного размера и с любыми (не пустыми) символами соответствующего алфавита на любом месте в пределах длины строки, нам необходимо, чтобы было истинным следующее утверждение:

$\mathrm{l}_9(a) \neq \ominus_a \wedge \ominus_a = \ominus_b \Rightarrow$

$\Rightarrow (i_9 \leq \mathrm{l}_9(a) \Rightarrow \mathrm{s}(a \cdot b, i_9) = \mathrm{s}(a, i_9)) \wedge (i_9 > \mathrm{l}_9(a) \Rightarrow \mathrm{s}(a \cdot b, i_9) = \mathrm{s}(b, i_9 - \mathrm{l}_9(a)))$

На самом деле здесь достаточно аксиоматизировать конкатенацию произвольной конечной строки с одним символом справа (а можно и слева) – остальное же можно определить. Важно аксиоматизировать, что произвольные конечные строки есть и они могут быть получены из символов своего алфавита, последовательным присоединением к нарастающей «символ-за-символом» строке. А для этого хватит аксиомы для присоединения одного символа к произвольной конечной строке.

Необходима ли аксиома о наличии бесконечных строк и как её записывать, если необходима – вопрос. Или достаточно дать аксиому конкатенации с бесконечной первой строкой – логично считать результат такой конкатенации равной первой строке, на мой взгляд.

На практике мы никогда не имеем дела с актуальной бесконечностью при работе со строками – даже непрерывная трансляция в интернете, которую можно интерпретировать, как строку – используется нами в конечных пределах, и нам нет нужды считать, что эта трансляция будет идти вечно.

А раз мы используем любую строку в конечных пределах, то вопрос о её (бес)конечности не

важен с практической точки зрения. И работа функции $\mathrm{s}(a, i)$ не зависит ни от размера строки a за пределами i, ни от того, может ли строка a быть бесконечной.

Если нам все же необходимо обосновать наличие бесконечных строк, то, возможно, для этого можно использовать теорию множеств, которая специализируется по бесконечностям в том числе. Но для прикладных нужд хотелось бы иметь теорию без столь громоздкой «добавки» в неё, как теория множеств, на мой вкус – но я не настаиваю.

Во всяком случае, я дошел до предела своего нынешнего понимания логики строк, и пора заканчивать это изложение. Можно, конечно, делать бесконечное количество логических выводов из изложенного выше – но речь тут не о «вторичных» теоремах, а о «ядре» логики строк.

Прежде, чем идти дальше, хотелось бы получить реакции математиков и практиков ИТ (желательно) на изложенное выше и их возможные добавления к «ядру» логики строк.

Что касается формальной теории – то пока что в приоритете – опора на реальность, в которой практики почти век легко работают со строками без формальной теории. Реальность в данном случае не менее крепкая (даже более крепкая) опора, чем мир идей математики – который ещё не сформирован к тому же в отношении строк. Другое дело, что мир идей удобнее для математической работы, но – всему своё время.

Напоминаю, что основная цель данной работы – это выявление принципиальной ошибки в теории вычислимости. Вычислимость по Тьюрингу превосходит в принципиальном отношении вычислимость рекурсивных функций – вот что верно, а не «доказанное» ранее совпадение их возможностей в плане вычислимости. А то, что сказано сверх этого – «бонус», чтобы «заполнить пустоту», которая образовалась, когда логика арифметики оказалась не достаточной для описания строк.

И даже если вскоре появятся формальные теории для строк – а они появятся, наверно, то это не должно отменять в ближайшие десятилетия приоритет реальности над миром идей в отношении работы со строками. Вполне возможно, что будет обнаружено ещё что-то необходимое для «ядра» логики строк, что потребует расширения или даже пересмотра теории.

Своё нынешнее понимание логики строк я изложил выше. Не всё там записано в виде математических формул. Кое-что сказано словами, какие-то свойства отмечены косвенно – например, в рассуждении про нумерацию символов понятно, что длина строки из одного символа равна 1, о чем прямо не написано. В каком-то смысле это и есть неформальное изложение теории строк в моем нынешнем понимании. Но это не окончательный результат, а лишь первый – в цепи последующих «итераций».

4 Проблема была не в построении формальной теории, а в переносе части реальности в мир идей математики

В заключение выскажу мнение, почему математики так долго не могли увидеть «очевидного» отличия чисел-строк от натуральных чисел и даже «доказали» их, якобы, одинаковые свойства с точки зрения вычислимости.

Но для начала разберемся в том, что «открыла» эта статья и разве никто не понимал этого до неё? Десятки миллионов программистов на практике использовали строки и числа-строки, понимали и применяли их свойства в работе, зарабатывали в сумме сотни миллиардов долларов. Вроде бы, всё хорошо с их практическим пониманием...

Но преимущество теоретического понимания в том, что его можно передать в «готовом виде». То есть, такое понимание – более глубокое и – главное – рациональное, одинаковое для всех. Разве люди не понимали, что числа, которые используются компьютером – это не абстрактные натуральные числа – а такие объекты, которые кроме числовой информации обладают своим типом, записаны при помощи «символов» (то есть – это «строки») и обладают таким свойством, как количество упорядоченных (пронумерованных) символов, притом это количество разное для одного и того же натурального числа при разных типах представляющих его «строк»? Понимали, но – каждый по-своему. Так, что было невозможно передать это понимание в готовом виде. Опытные программисты показывали, как работать, и обучающийся у них человек понимал для себя сам – с чем он работает.

Однако, теоретическое понимание – это всегда некоторое абстрагирование от «лишних» деталей. А при практической реализации теоретических идей начинает на практике «работать» и то, что в исходном теоретическом знании не учитывалось.

И эти «побочные эффекты» могут иметь важное практическое значение и использоваться практиками – сверх теоретического знания. Но, допустим, все важные «побочные эффекты» перенесены из реальности в теорию, и все практически значимые эффекты стали теоретически предсказуемыми и доступными для преподавания (передаче ученикам в готовом виде). Тогда не возникает необходимости менять теорию. И даже если технологии меняются так, что в них не возникает «не теоретических» эффектов, имеющих важное практическое значение и используемых на практике – то необходимости менять теорию тоже не возникает.

Вот математические идеи – логика, натуральные числа с их свойствами, геометрия и т.п. – были перенесены в мир идей математики из реальности когда-то очень давно. И оказались настолько удачными абстракциями, что очень долго не требовали дополнительного переноса чего-то принципиально нового из реальности. А не принципиально новое удавалось найти в математике в качестве применения каких-то возможностей, которые уже есть в этом «мире идей».

Собственного мира идей математикам было достаточно, чтобы в его рамках конструировать необходимые «прикладникам» формулы, не испытывая необходимости в новых идеях, которые нельзя было бы создать на базе уже имеющихся у них. От математики и не требовали того, чего она не могла дать – «потребители» выражали желание что-то посчитать, например, а математики

считали – если им удавалось найти способ. И по мере того, как они придумывали отрицательные числа и корни из них, пределы и бесконечно малые с дифференцированием, сходящиеся ряды и т.д. – они давали все больше решений для разных случаев желаемого расчёта. Это математики переносили свои вновь обретенные в мире своих идей возможности в мир «обычных» людей, а обратного переноса не требовалось – «вот это математика может, а вот это сейчас не может – так она устроена и всё тут».

Назовем развитие на основе готового мира идей – «вертикальным прогрессом». Но есть и то, что рождается на стыке теорий. И тогда мы имеем «горизонтальный прогресс». При любом быстром прогрессе повышается вероятность выйти на такие практические приложения, где «побочные эффекты» (не учитываемые теорией) будут иметь существенное практическое значение, и это будет означать не полное соответствие теоретического знания современной практике.

Но вероятность быстрого «вертикального прогресса» сокращается со временем – самые «короткие» выводы из фундаментальных идей этого мира идей получены, а дальше сложность растёт от длины вывода экспоненциально или около того. А вот при «встрече» разных теорий и выработке способов их совместного использования – снова есть возможность пройтись по «низам» с короткими выводами – это горизонтальный прогресс, и он может быть быстрым в начальный период и какое-то время после.

Появление теорий в математике с 19 века и до периода перед 2-й мировой войной было вызвано объединением знаний и исследований посредством математических изданий и быстрого развития прочих глобальных информационных коммуникаций научной специализации. «Соединенные» знания надо было классифицировать и отбросить ошибочное – там, где были противоречия. Но это же объединение знаний вызвало в первую очередь не вертикальный, а горизонтальный прогресс. Ведь не одна математика соединилась глобальными коммуникациями, а вся информация – научная, экономическая, технологическая, политическая и т.д.

Результатом же периода быстрого горизонтального прогресса (который сейчас практически завершился) стало появление таких технологий, в которых имеются практически существенные, но не учтенные в мире идей математики свойства. Требовалось узнать размер «числа», время работы программы и вычислить другие очень важные для практики свойства данных и алгоритмов.

В мире идей математики не было ничего, что соответствует подобному «числу из компьютера». Натуральное число – это такая абстракция, которая проверена тысячами лет и никогда не требовала принести из реальности какие-то дополнительные свойства к тем, что у этого числа есть в математике. Со времен Евклида, видимо, математики не занимались «импортом» из реальности для своего мира идей. А Евклид совсем не вчера жил.

Поэтому не удивительно, что математики не поняли, что они чего-то не учитывают, когда они стали разбираться с вычислимостью по Тьюрингу и им надо было перенести не перенесённую ранее часть реальности в мир идей – но они уже давно забыли даже то, как такое можно понимать, потому что думать «надо» только в мире имеющихся у математики идей.

Кстати, прогресс, возникающий из-за переноса части реальности в мир идей, в котором соответствующего знания не было и его нельзя было получить средствами исходного мира идей – назовем

такой прогресс «прогрессом из реальности». В отличие от «прогресса из теории» - к которому относятся «вертикальный прогресс» и «горизонтальный прогресс».

Я и сам в вопросе вычислимости не критично поверил «доказательству» одинаковой вычислимости по Тьюрингу и рекурсивных функций с «оправданием» себя в духе: «Вот так математики понимают, что такое вычислимость – чтобы удалось сделать сравниваемые тут вычислимости одинаковыми».

Но я был не согласен, что числа-строки и натуральные числа могут быть «одинаково вычислимы» с точки зрения более тонких подходов теории алгоритмов – где учитывают время и размеры.

И там я нашел пример, где время работы алгоритма при работе со строкой заведомо отличается, как угодно сильно, от времени работы алгоритма с числом, соответствующим этой строке.

Надо было только записать «соответствующее» натуральное число на ленту Машины Тьюринга в его «стандартной интерпретации». Раз годится натуральное число, то не имеет значения, какую правильную интерпретацию для записи этого числа мы используем – в нем есть вся логика натуральных чисел. Желательно брать самую «примитивную» (но полностью соответствующую арифметике) интерпретацию, чтобы избежать посторонних (не арифметических) эффектов.

А теперь представим, что алгоритму надо проверить 1-й символ в полученной им строке и сообщить ответ – совпадает ли он с первым символом алфавита или нет. На практике для этого достаточно прочитать первый символ на ленте Машины Тьюринга. И время на работу алгоритма не будет зависеть от длины полученной алгоритмом строки. А вот если пытаться так же «частично» прочитать число в стандартной интерпретации, соответствующее данной строке?

Допустим, можно, прочитав N (фиксированное количество) «счётных палочек», выяснить, что соответствующая этому числу строка начинается на первый символ алфавита. Тогда это будет означать, что все числа от $N-1$ и до бесконечности начинаются на данный символ. А на все остальные строки остаётся только $N-1$ вариантов разных чисел, включая ноль. Но $N-1$ вариантов нельзя поставить в соответствие бесконечному количеству вариантов строк, начинающихся не на первый символ алфавита. Поэтому в рамках теории алгоритмов не может быть соответствия строк многосимвольного алфавита и натуральных чисел.

Идея красивая, на мой вкус, но «частичное чтение» означает ещё и время, которое «короткое», которое не зависит от длины полученной строки. А прежде, чем добраться до времени, надо построить теорию для строк с размерами, алфавитом и т.д. Я бы делал это ещё, наверно, десяток лет (или больше) в свободное от работы время. Но случился 2020 год с его карантинами и повезло познакомиться с профессиональным математиком Сергеем Витальевичем Знаменским, который был готов обсуждать исследования такого занудного любителя, как я.

Без малого два года переписки, он дал ссылку на статью String theory и многое другое. Был, фактически, моим научным руководителем при обсуждении того, какой должна была бы быть теория строк для работы с вопросами теории алгоритмов и как написать эту теорию в соответствии со стандартами математических изданий. Его интуиция не раз помогла сузить область поиска ответа до вполне обозримой. И это он предложил дать доказательство невозможности индукции – в некоторых случаях – применительно к строкам. Дать это доказательство на базе накопившихся за время

переписки сведений о свойствах строк и теории алгоритмов. Я уже и не смогу перечислить весь его вклад, столько всего накопилось и «смешалось» в общий результат, поэтому считаю его тоже автором в том, что было сделано до дискуссии 14-15 февраля 2022 г, когда общение прервалось.

И даже наша последняя – на данный момент – дискуссия 14-15 февраля 2022 г. подтолкнула меня искать принципиальную ошибку в «доказательстве» одинаковой вычислимости по Тьюрингу и рекурсивных функций. Пусть это и было противоположно той позиции, которую отстаивал он, но область инициированного им спора оказалась очень конкретной и доступной для быстрого результативного исследования. В результате которого и появилась данная статья.

И совсем в завершение выскажу предполагаемый прогноз – каким будет ближайшее развитие в математике из-за новой информации в данной статье:

Придется пересмотреть теории вычислимости и алгоритмов из-за того, что у строк оказалось такое свойство как тип. Теперь в мире идей математики действительно есть необходимая для теории алгоритмов возможность получать «размер данных» от строк (включая числа-строки). Надо будет разбираться – достаточно ли этого для понимания работы алгоритмов в рамках мира идей математики, или потребуется переносить ещё что-то из реальности для правильного понимания времени и, возможно, чего-то ещё.

Разумеется, это всё работа не для одного человека. Надеюсь, государства дадут на соответствующие исследования денег своим математикам. И если действительно дадут денег на эту работу, то хотелось бы и самому в ней поучаствовать :)

5 Приложение 1. О «доказательстве» совпадения логики строк и натуральных чисел»

В разделе 1 «Вычисление длины числа-строки на ленте машины Тьюринга и отсутствие аналога в арифметике и среди рекурсивных функций для такой же вычислимости длины натурального числа» мы слегка коснулись возможности (мнимой) свести логику строк к логике натуральных чисел. Здесь мы подробней исследуем эту попытку и то, почему «очевидная» ошибка была настолько не очевидна, что логика натуральных чисел и логика строк считались неотличимыми на протяжении почти века – в рамках задач теории вычислимости, по крайней мере.

Метод «свести» строку к числу является частью нынешнего «мира идей» математиков и поэтому нужно упомянуть, как это было исполнено. И, кроме того, я добавлю в этот способ аргумент «номер типа» – мы в одной и той же арифметике можем вычислять числовые характеристики строк для разных типов. И мы обойдёмся без логики строк, при этом, если этот тип получен не как свойство строк, а как «внешний» аргумент.

Разумеется, чтобы реализовать эту программу, необходимо совершить практическую ошибку и считать, что можно произвольным образом задавать соответствие между числами и строками. Хотя на практике подобные соответствия должны быть заранее известны (заранее известно, какая система счисления используется в этом месте программы, какие символы можно считать цифрами и т.д.) и менять их «для своего удобства» не допустимо. Но мы исходим из того, что «нам можно». И посмотрим, что тогда можно сделать.

Во-первых, отметим неувязку обычной позиционной системы счисления в вопросе соответствия всем конечным строкам, которые можно составить из её цифр. А именно:

1. Пустая строка. Бывает, что на ленте МТ ничего не записано. Чему равна длина такой строки? Очевидно, что нулю. По крайней мере, так принято в ИТ. Вот для обозначения «пустоты» будем использовать обозначение $\ominus$. А когда нам необходимо указать тип (не касаясь пока вопроса, имеются ли разные типы пустых строк, подобно наличию разных типов чисел-строк), тогда будем обозначать этот тип нижним индексом так же, как для чисел-строк: $\ominus_U$.

2. Строки вида 00231. То есть – с ведущими нулями, когда это не число ноль. Если считать их тем же числом, которое получается при удалении ведущих нулей – то тогда возникает неувязка в смысле соответствия разным строкам одного и того же числа.

Во-вторых, нам надо как-то решить проблему того, что у нас не один тип (не одна позиционная система счисления с основанием 10, например), а поэтому, нам необходимо иметь основание системы счисления, чтобы совершить правильные действия с левым числом-строкой в конкатенации. Например, для десятичной системы это число надо умножить на десять – тут видна зависимость именно от системы счисления. А правое число (цифру) после этого можно добавить к результату данной операции с левым числом.

В принципе, основание системы счисления у нас в рассматриваемом случае равно индексу типа плюс 1 $(U+1)$ в функции получения цифры по номеру:

$\mathrm{e_U}(i_U)$

И, поскольку для конкатенации мы передаём результат работы именно этой функции, то в этот результат можно «упаковать» и число, соответствующее типу, и число, соответствующее цифре. Это можно сделать разными способами, не будем тут разбирать эти второстепенные детали. Желающие могут посмотреть, например, «канторовский номер пары».

Тогда из натурального числа, равного $\mathrm{e}(U,i)$ можно получить при помощи некоторых арифметических функций $\mathrm{el}(\ldots)$ и $\mathrm{er}(\ldots)$ первый и второй аргументы (тоже натуральные числа), переданные функции $\mathrm{e}(U,i)$:

$\mathrm{el}(\mathrm{e}(U,i)) = U$, получение «левого» числа пары, из номера этой пары

$\mathrm{er}(\mathrm{e}(U,i)) = i$, получение «правого» числа пары, из номера этой пары

Таким образом, можно арифметическими методами составить функцию, которая будет соответствовать конкатенации, если в качестве второго числа (правой части) конкатенации используется результат работы функции $\mathrm{e}(U,i)$.

И для десятичной записи числа конкатенацию $i \cdot \mathrm{e}(9,j)$ с цифрой j можно тогда вычислить так:

$i \times \mathrm{el}(\mathrm{e}(9,j)) + \mathrm{er}(\mathrm{e}(9,j))$

Но при таком методе результат работы функции $\mathrm{e}(U,i)$ сам по себе даёт не тот результат, который равен числу, соответствующего цифре с номером i. А в аксиоме о связи чисел и строк мы использовали функцию $\mathrm{e}(U,i)$ не только вместе с конкатенацией, но и отдельно от неё.

Однако ничто не мешает изменить аксиому так, чтобы вместо независимого применения функции $\mathrm{e}(U,i)$ в некоторых конъюнктивных членах, использовать её конкатенацию с «пустой строкой» в духе:

$\ominus \cdot \mathrm{e}(U,i)$

И, таким образом, нам удастся удовлетворить и требованиям, перечисленным выше (см. «во-первых») и сейчас (см. «во-вторых»). А именно, если заменить в разбираемой аксиоме первые два члена конъюнкции:

$(i_U \leq U_U \Rightarrow i_U = \mathrm{e}_\mathrm{U}(i_U))$

$\wedge(i_U < U_U \Rightarrow \mathrm{e}_\mathrm{U}(i_U)' = \mathrm{e}_\mathrm{U}(i'_U))$

На следующее:

$0_U = \ominus_U$

$\wedge(0_U < i_U \leq U_U \Rightarrow i_U = 0_U \cdot \mathrm{e}_\mathrm{U}(i_U))$

$\wedge(0_U < i_U < U_U \Rightarrow (i_U)' = 0_U \cdot \mathrm{e}_\mathrm{U}(i'_U))$

Использованный метод сопоставления строк и чисел взяты из статьи String theory – хотя там для каждого отдельного типа строки составляется своя отдельная теория и нет понятия «тип строки». Соответствующий этому методу способ записи числа при помощи цифр поясню на примере построения чисел из цифр 0-9 и пустой строки $\ominus$:

Ноль записывается как пустая строка $\ominus$

А вот число, записанное одной цифрой ноль, в такой системе счисления означает 1.

Поэтому число, состоящее из одной цифры от 0-9 будет соответствовать числам 1-10. Далее:

00 – это $(10+1)+0$

09 – это $(10+1)+9=20$

10 – это $(10 + 10 \times 1 + 1) + 0 = 21$

90 – это $(10 + 10 \times 9 + 1) + 0 = 101$

99 – это $(10 + 10 \times 9 + 1) + 9 = 110$

000 – это $(110 + 1) + 0 = 111$

И т.д.

Данный способ записи чисел в статьи String theory называется radix-n notation. Где n – основание для данной системы счисления. Наш пример соответствовал $n = 10$, потому что мы использовали 10 цифр: от 0 до 9. Напоминаю, что индекс типа U у нас тут на 1 меньше, чем n. Легко видеть, что любая комбинация цифр соответствует одному и только одному числу. Притом даже для пустой строки есть соответствующее ей число – ноль.

Тогда вся данная аксиома будет описывать всего лишь арифметические возможности представления натурального числа при помощи аналогов стандартной интерпретации (но в которой нулю соответствует отсутствие счётных палочек, а в остальных случаях их ровно столько, какое число им соответствует) и аналогов radix-n notation при произвольных основаниях, соответствующих типу U.

Тогда любые 0_U, $\ominus_U$ можно считать одним и тем же нулем, а типы не имеют значения.

Раз нам нужен аналог конкатенации, то нам нужен аналог способа дописывать младшую цифру к десятичному представлению – там мы умножаем исходное число на 10 и прибавляем цифру, которую надо «дописать» в конец. Что нам делать для той же цели с radix-n notation? Для простоты – с radix-10 notation.

Пусть в числе-строке длиной в m цифр все цифры будут 9. Тогда это число равно сумме геометрической прогрессии:

$10 + 10^2 + \ldots + 10^m$

Приведённая сумма просто пересчитывает комбинации для предыдущих чисел с разным количеством цифр – от 1 до m. Есть ещё случай нуля, но он ничего не добавляет. Поэтому мы суммируем количество комбинаций 10 цифр на каждом из i мест, где $1 \leq i \leq m$.

но лучше переписать так:

$(1 + 10 + 10^2 + \ldots + 10^m) - 1 = (10^{m+1} - 1)/(10 - 1) - 1$

Тогда следующее за ним число будет состоять из $(m + 1)$ цифр ноль, и оно будет равно:

$(10^{m+1} - 1)/(10 - 1)$

То есть, число из m нулей $000\ldots00$, записанное в форме radix-10 notation, будет равно следующему числу в десятичном представлении (по формуле геометрической прогрессии):

$(10^m - 1)/(10 - 1)$

На самом деле это $999\ldots99/9 = 111...11$ из m цифр 1 (в десятичном представлении), что мы и видели раньше на конкретном примере.

Если же у нас старшей будет цифра Y, а остальные m цифр будут нули, то число будет:

$(10^{m+1} - 1)/(10 - 1) + Y \times 10^m$

Потому что между соседними старшими цифрами при числе из $m + 1$ цифр и с нулями в остальных позициях происходит перебор всех значений цифр от 0 до 9 (поэтому основание степени

10) на m позициях.

Легко видеть, что увеличение общей длины числа на 1, от исходного числа, состоящего из одних только цифр ноль, до состояния тоже из одних цифр ноль, требует увеличения исходного числа до:

$(10^{m+1}-1)/(10-1)+10\times 10^m=(10^{m+1}-1)/(10-1)+10^{m+1}=$

$=(10^(m+2)-10^(m+1)+10^{m+1}-1)/(10-1)$

Данный результат посчитан как разница между конечным и начальным числами в следующей последовательности, записанный в radix-10 notation:

00...00

10..00

20..00

90..00

99..99

000...00

После сокращения равных по величине и противоположных по знаку слагаемых получим:

$(10^{m+2}-1)/(10-1)$

Что соответствует ранее полученной формуле.

Итак, у нас есть величина для базового числа, записанного только цифрами 0. От него можно перейти к числу с любой другой старшей цифрой и остальными нулями. И аналогично, можно перейти ко всем остальным цифрам, отличным от нуля.

Интересно, что отличные от нуля цифры вносят ровно тот вклад, который они вносят при «обычной» позиционной записи. А добавочная часть – от размера числа-строки в формате radix-n notation – соответствует «обычной» позиционной записи из одних единиц в том же количестве.

То есть, число-строка, записанное в формате radix-10 notation:

23815

Соответствует сумме десятичных чисел:

$11111+23815$

Если нам к первому числу в формате radix-10 notation надо дописать 9:

238159

То соответствующая ему сумма в десятичном виде станет:

$111111+238159$

А это значит, что от исходного числа 23815 в формате radix-10 notation переход к новому числу «приписыванием» нового разряда единиц 238159 происходит через умножение исходного числа на 10, добавлением 1 и добавлением нужной цифры. Действительно, в десятичном виде:

$(11111+23815)\times 10+1+9=111110+1+238150+9=111111+238159$

Разумеется, читатель легко может всё это проделать на уровне формул для общего случая самостоятельно.

Смысл данного Приложения в том, что при фиксированном типе radix-n notation имеется соответствие между арифметикой (вычислимостью рекурсивных функций в том числе) и вычислимостью по Тьюрингу (тут не полная выборка из всей вычислимости по Тьюрингу, а только для

чисел-строк типа radix-n notation). И даже для n как параметра radix-n notation можно построить «общее» соответствие с арифметикой.

Для фиксированного параметра n соответствие натуральных чисел с числами-строками типа radix-n notation доказано – если признать убедительными доводы, приведенными в статье String theory. Я, например, посчитал эти доводы убедительными. Там, конечно, используется не обычная аксиоматизация, но ведь доказано, что логика этой аксиоматики эквивалентна логике арифметики (второго порядка, но это не существенно в данном случае), поэтому своеобразие в аксиоматике не отменяет того, что мы имеем дело с арифметикой и её теоремами.

В данном Приложении добавлено лишь расширение соответствия между арифметикой и числами-строками типа radix-n notation с уровня фиксированного n до n как параметра.

В своих построениях для такого расширения я использовал рекурсивные функции – а это арифметические функции, как известно. В каждом конъюнктивном члене «аксиомы».

Но вот если объединить тип и число в общее данное – тогда уже не получается остаться в рамках арифметики, если придать этому «расширенному объекту» свойства чисел-строк. И это (выход за рамки арифметики) доказано для чисел-строк в разделе 2 на примере нарушения аксиомы индукции.

И вот тут интересный вопрос – для фиксированного типа соответствие с арифметикой есть, даже для случая типа, как параметра – соответствие есть, а для объединенных в «общий объект» тип-и-число – соответствия с арифметикой нет. В чём причина такого тонкого расхождения?

Ответ такой: при параметре вместо типа строки, у нас нет способа выбрать правильный способ расчета (для вычисления длины числа-строк, например) на основании свойства данных. Например, в арифметике число 10 одно и то же, а параметр (вместо типа) – это варианты его рассмотрения как числа-строки. Притом внутри арифметики речь не о настоящем числе-строке, а о некотором варианте расчёта для выбранного «внешним» способом типа числа-строки. То есть, соответствие с реальным типом числа-строки достигается «внешним» по отношении к теории (арифметике) образом. В этом выборе (в выборе типа как конкретного параметра из всех возможных) есть своя логика, но эта логика используется – вне теории в данном случае.

Нельзя, впрочем, утверждать, что любые типы строк имеют соответствующие им арифметические построения с параметрами вместо типов – как это было показано на примере radix-n notation. Есть такие типы строк, например, которые в принципе не являются числами.

И не следует подходить к данному Приложению как к попытке дать строгое доказательство тому, что все типы строк radix-n notation (при произвольном n как параметре), имеют соответствующую «проекцию» на арифметику – это лишь оценочная попытка понять, до какой степени простирается совпадение логики натуральных чисел и логики «удобных» типов чисел-строк. И, насколько можно предположить по такому достаточно беглому взгляду, дело всего лишь в типе числа-строки, как свойстве этого объекта (а не параметра в теории). А все остальные отличия между натуральными числами и числами-строками типа radix-n notation, вытекают из данного. Но, ещё раз повторюсь, это всего лишь гипотеза, если говорить строго. А настоящее Приложение – доводы в пользу этой гипотезы и некоторый план для попытки строго обосновать данную гипотезу, если подобное желание

возникнет.

Список литературы

[1] *John Corcoran; William Frank; Michael Maloney.* String Theory. The Journal of Symbolic Logic, Vol. 39, No. 4, Storrs, USA. 1974

[2] *Дж. Булос, Р. Джеффри.* Вычислимость и логика. Мир, М. 1994

[3] *Э. Мендельсон.* Введение в математическую логику. Наука, М. 1984

[4] *А.И. Мальцев.* Алгоритмы и рекурсивные функции. Наука, М. 1986

[5] *В.И. Игошин.* Теория алгоритмов. Инфра-М, М. 2013