Теорема Чейтина

Gregory Chaitin родился в 1947 в Чикаго.

Первая публикация «An improvement on a theorem of E. F. Moore», IEEE Transactions on Electronic Computers EC-14 (1965), pp. 466-467.

В 1966 его семья переезжает в Буэнос-Айрес.

В 1970 публикует абстракт «Computational complexity and Gödel's incompleteness theorem», AMS Notices 17 (1970), p. 672.

В апреле 1971 публикует статью «Computational complexity and Gödel's incompleteness theorem», ACM SIGACT News, No. 9 (April 1971), pp. 11-12. Это его десятая публикация.

Автор 15 монографий. В их числе:

- Algorithmic Information Theory, Cambridge University Press, 1987.
- Information-Theoretic Incompleteness, World Scientific, 1992.
- The Limits of Mathematics: A Course on Information Theory and the Limits of Formal Reasoning, Springer, 1998.
- The Unknowable, Springer, 1999.
- Exploring Randomness, Springer, 2001.
- From Philosophy to Program Size. Key Ideas and Methods. Institute of Cybernetics, Tallinn, 2003.
- Meta Math! The Quest for Omega, Pantheon, 2005.

Две статьи на русском языке:

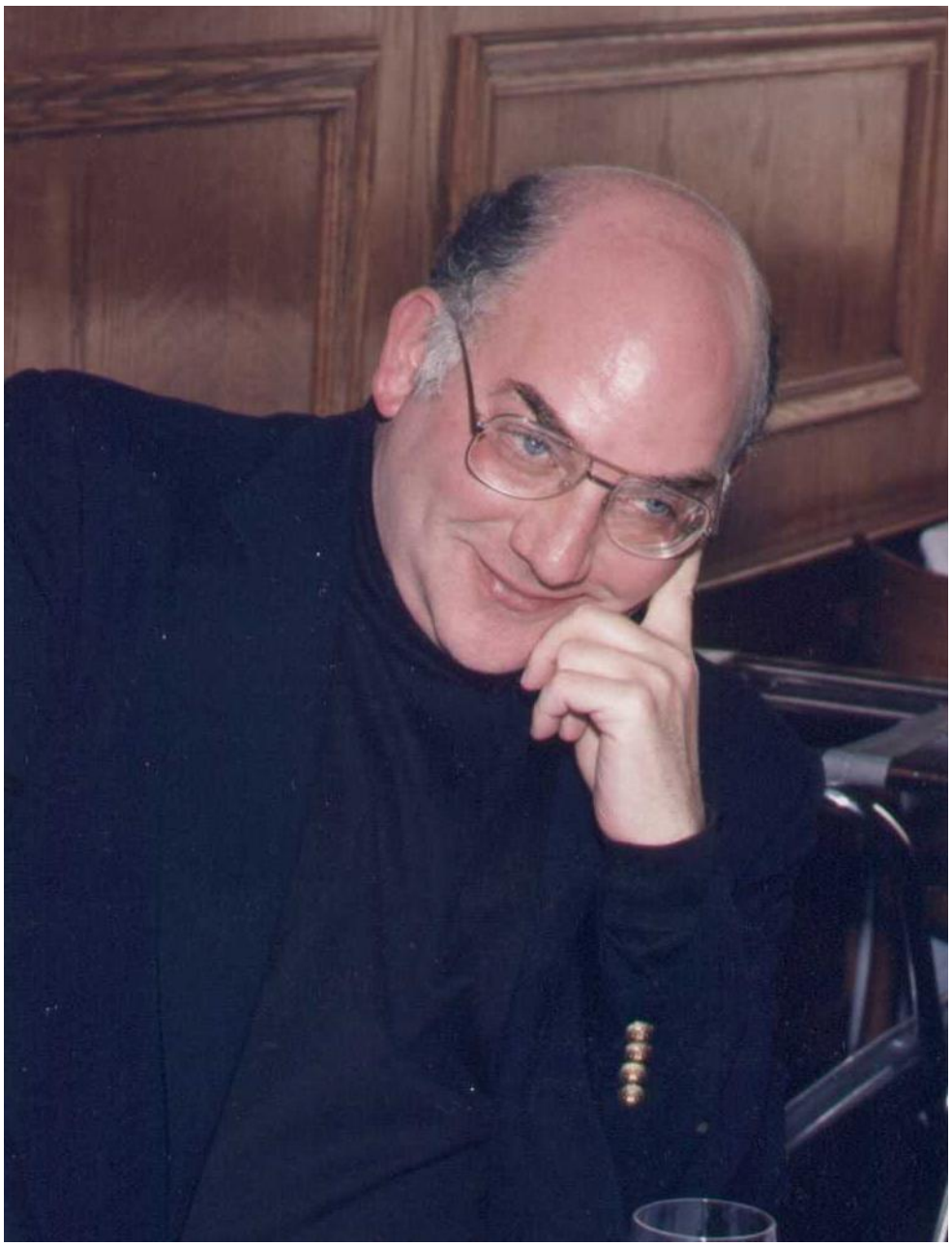
- «Случайность в арифметике», В мире науки, №9, 1988 г. http://z-mech.narod.ru/Lib/chaitin3.html
- «Пределы доказуемости», В мире науки**,** №6, 2006 http://elementy.ru/lib/430319

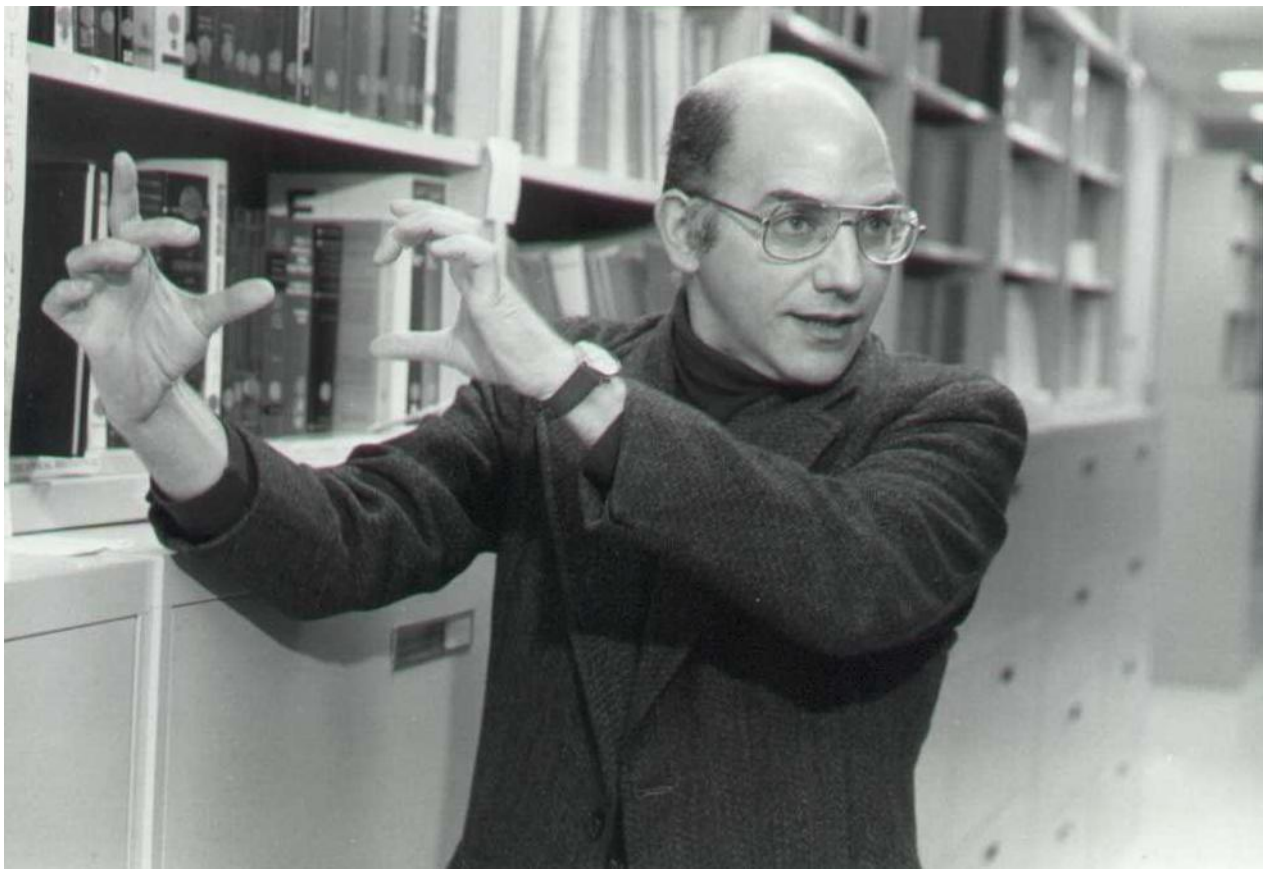
Интернет-ресурсы:

- Home page - http://www.umcs.maine.edu/~chaitin/
- Много публикаций на http://arxiv.org/
- Много можно найти на http://bookfi.org/

Публикации других

- Calude C.S. «Randomness and Complexity, from Leibniz to Chaitin»





Знание мы представляем в языке. В XX веке логиками был доказан ряд фундаментальных теорем об ограничениях, которые налагает язык на наши возможности представлять знание. В первую очередь я имею в виду теоремы К.Гёделя и А.Тьюринга.

В 1931 Гёдель показал, что всякая достаточно богатая теория, в которой мы можем сформулировать предложение, утверждающее собственную недоказуемость, неполна. Более того, мы не можем доказать непротиворечивость этой теории, если не выйдем за ее границы и не воспользуемся более богатым арсеналом средств. Теорема Гёделя не сильно впечатлила тех математиков, которые были свидетелями его доклада и даже не была включена в обзор наиболее важных результатов конференции. Причина заключалась в том. Что примеры недоказуемых предложений были слишком экзотичны и казались далекими от математической практики. Лишь некоторые математики по достоинству оценили ее.

В 1936 Тюринг построил математическую модель абстрактного вычислительного устройства, получившего впоследствии название машины Тюринга, и доказал, что не существует универсального метода, который по произвольной программе мог определить, завершится или нет ее выполнение. Этот результат получил название *неразрешимости проблемы остановки*. Легко показать, что теорема Гёделя является следствием неразрешимости проблемы остановки.

В 1970 Грегори Чейтин доказал еще одну теорему об ограниченных выразительных возможностях логических формализмов. Теорема Чейтина интересна своей большей приземленностью и тесной связью со многими вопросами методологии науки - понятиями закона науки, научной теории, аксиом теории и пр. У Чейтина разносторонние интересы, он не замыкается в математике и программировании, которым зарабатывает себе на жизнь, а стремится донести полученные результаты до возможно большего числа людей, которым они могут быть интересны.

Если в основании теоремы Геделя лежит вариант парадокса Лжеца, то в основании теоремы Чейтина лежит парадокс Берри - «*Наименьшее число, которое невозможно описать менее чем 1000 символов*».

Моей целью не является подробный анализ этого парадокса. Если есть желание, лучше ознакомиться с ним в изложении самого Чейтина в статье «The Berry Paradox», найти которую легко по приведенным мною ссылкам.

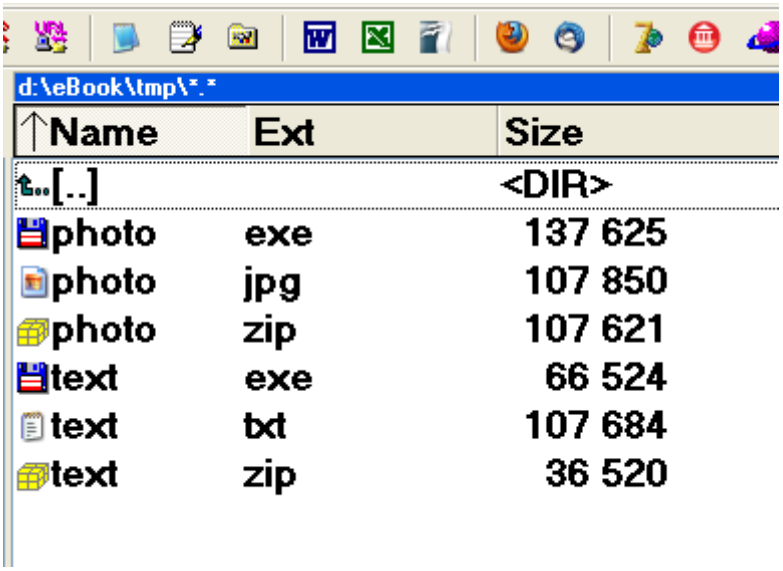
Для начала необходимо объяснить, что такое алгоритмическая сложность. Косвенным образом каждый пользователь компьютера сталкивается с ней каждый день.

Я взял два файла Photo.jpg и Text.txt размера 107.850 и 107.684 байт и заархивировал их с помощью ZIP, создав самораспаковывающиеся архивы размера 137.625 и 66.524 байт соответственно. Теперь, просто щелкнув мышкой по самораспаковывающемуся архиву, я запускаю скрытую в нем программу и в результате получаю распакованный исходный файл.

Если бы я заархивировал исходные файлы просто в zip-архивы, то получил бы файлы размера 107.621 и 36.520 байт соответственно. Легко проверить, что разница в размерах между самораспаковывающимся и просто запакованным файлом в обоих случаях равна 30.004 байт.

	Photo.jpg	**Text.txt**
Исходный файл	107850	107684
EXE	137625	66524
ZIP	107621	36520
EXE-ZIP	30004	30004

Привожу скриншот, чтобы подтвердить, что это реальный пример.



Если говорить нестрого, то алгоритмическая сложность данных - это минимальный размер их самораспаковывающегося архива. Поскольку архиваторы могут быть разными, то и качество упаковки у них может быть разным. В приведенном выше примере исходные файлы примерно равны по размеру, а самораспаковывающийся архив текстового файла в 2 раза меньше, чем архив фотографии. Отсюда следует, что алгоритмическая сложность текстового файла меньше, чем сложность файла с фотографией.

Вспомним, что все данные в компьютере хранятся в виде цепочек, состоящих из 0 и 1. Поэтому удобно говорить не о файлах, а о строках из 0 и 1. Также можно отвлечься от конкретного типа архиваторов и от exe-файлов.

Лучше всего было бы говорить в терминах машин Тюринга, но можно говорить и в терминах абстрактных компьютеров, поскольку все они являются реализациями математической модели Тюринга.

В дальнейшем изложении я буду опускать технические детали, которые не добавляют прозрачности результатам Чейтина, но которые легко восстановить в случае необходимости.

Обратимся к внутренностям абстрактного архиватора.

Допустим нам необходимо заархивировать строку символов, состоящую из 1.000 единичек «1». С этой задачей прекрасно справится следующая программа:

Старт
Напечатать 1000 раз «1»
Стоп

Как видим, она гораздо короче 1000 символов.

Если бы нам нужно было заархивировать строку, состоящую не из 1.000 единичек «1», а из 1.000.000 символов - чередующихся «01» - «0101…01», длина нашей программы увеличилась бы всего на 3 символа.

Старт
Напечатать 500000 раз «01»
Стоп

Мы видим, что в программе есть константная часть фиксированного размера и переменная часть, содержащая указание на то, сколько раз и что печатать. Константная часть реализует алгоритм распаковки, а постоянная часть в сжатом виде скрывает распаковываемые данные. Константная часть - это и есть те 30.004 байт нашего исходного примера, довесок к сжатым данным.

Обратим внимание на то, что верхней границей размера архива для конкретной строки S является сумма длины символов константной части программы плюс длина строки S.

$$\text{длина}(S) + C$$

Эта верхняя граница реализуется программой, которая внутри себя хранит строку S в исходном виде.

Старт
Напечатать 1 раз S
Стоп

Этот случай, когда программа вынуждена хранить внутри себя данные в несжатом виде, не является аномалией, а, согласно Чейтину, наиболее часто встречающимся случаем.

Представим себе, что мы получили строку в 1.000.000 символов путем бросания монеты «00101011100100101100 1…10111101», т.е. совершенно случайно. В этом случае мы не можем обнаружить внутри строки никаких закономерностей, которые позволили бы представить ее в виде повторения одинаковых структур, и потому вынуждены просто запомнить ее всю целиком. Эта строка обладает максимальной сложностью для строк такой длины.

Строгое определение алгоритмической сложности строки S выглядит следующим образом:

$$I(S) = \min L(p_S)$$

$L(p_S)$ - это длина/размер самораспаковывающейся программы *p*, содержащей внутри себя упакованную/сжатую строку S.

Словами это определение можно прочитать следующим образом:

Алгоритмическая сложность строки S - это длина самой короткой программы, результатом выполнения которой является S.

Одного этого определения достаточно для того, чтобы получить очень интересные следствия. Например, то, что 1) существуют строки символов произвольной сложности - $\forall n \exists s (I(s) > n)$; 2) для большинства строк их сложность примерно равна их длине. Возможность сжатия данных - это, скорее, исключение из правила, а не правило.

Первое утверждение непосредственно следует из того, что число строк дины *n* в алфавите из 0 и 1 равно 2^n, а число всех строк в этом же алфавите длины меньше, чем *n*, равно 2^n-2. Т.е. хотя бы двум срокам длины *n* мы не можем сопоставить строки меньшей длины, которые теоретически могли бы быть результатом их сжатия. На самом деле таких несжимаемых строк гораздо больше. Если мы возьмем произвольную строку *S* длины *n*, то строк меньшей длины, кодирующих ее, может быть не одна, а несколько, не все эти строки будут минимальной длины, некоторые строки меньшей длины не будут программами и т.д.

В качестве иллюстрации для второго утверждения скажу, что строк длины менее *n-8* символов 0 и 1 примерно в 500 раз меньше, чем строк длины от *n-8* до *n*.

Необходимо специально оговорить случай, когда кодируемые данные не имеют конечной длины. Такие случаи также встречаются очень часто. Например, число π. В свое время Тюринг предложил считать действительные числа вычислимыми, если мы можем вычислить их с любой заранее требуемой точностью.

Например, число π является действительным, и его структура выглядит случайной, но существует очень простая и короткая программа для вычисления этого числа с любой заранее заданной точностью.

$$\pi/4 = 1 - 1/3 + 1/5 - 1/7 + 1/9 - 1/11 + ...$$

Т.е. число π, с точки зрения теории алгоритмической информации, довольно простое. Существуют и другие алгоритмы вычисления π.

Мы - логики, и нам интересно, как можно оценить алгоритмическую сложность гильбертовских аксиоматических теорий. Опять же все очень просто. Одну из верхних границ дает известный *алгоритм Британского музея*. Суть его в следующем.

1. Мы можем записать все аксиомы формальной теории в виде строк символов.
2. Поскольку алфавит символов, посредством которых мы задаем формальные системы, конечен, мы можем составить программу, которая будет порождать все возможные цепочки в этом алфавите, в направлении увеличения их длины. Сначала - все возможные цепочки из одного символа, потом все возможные цепочки из двух символов и т.д.
3. По мере порождения цепочек для каждой из них мы будем проверять, удовлетворяет ли она определению доказательства в нашей системе. Если не удовлетворяет, то переходим к следующей цепочке, а если удовлетворяет, то выводим на печать последнюю формулу этой цепочки как теорему, и переходим к следующей цепочке.
4. Поскольку доказательство каждой теоремы - это конечная цепочка символов, то рано или поздно каждая теорема нашей теории будет выведена на печать.

Верхней границей алгоритмической сложности формальной теории будет суммарная длина аксиом плюс длина алгоритма Британского музея.

$$\text{длина}(Axioms)+C_{\text{Британский\_музей}}$$

Того, что я рассказал, достаточно, чтобы доказать основную теорему Чейтина, поскольку она очень проста. Гораздо интереснее философские следствия этой теоремы, но обсуждать их можно лишь после доказательства самой теоремы.

Доказательство теоремы Чейтина

Я буду излагать близко к первому доказательству Чейтина, которое было опубликовано им в статье «Computational complexity and Gödel's incompleteness theorem», ACM SIGACT News, No. 9 (April 1971), pp. 11-12.

Допустим, что у нас есть теория, в которой представима функция оценки алгоритмической сложности I(S). Мне не известна такая теория, но ее можно построить на базе арифметики. Достаточно взять того же Мендельсона, добавить к его формализации рекурсивных функций еще одну функцию длины выражений и дело почти сделано. Но даже если бы такой теории не существовало в природе, это нисколько не повлияло бы на значимость теоремы Чейтина, так как утверждение о том, что некоторая строка имеет конкретную алгоритмическую сложность, вполне осмысленно и должно оцениваться как истинное или ложное. Тем более, что верхняя граница сложности для любой строки вычисляется очень просто.

Итак, допустим, что у нас есть такая теория. В ней должна быть истинной формула $\forall n \exists s (I(s) > n)$. Мы будем интересоваться теоремами вида $I(S)>N$ для произвольной строки S и фиксированного N. Т.е. мы снимаем квантор всеобщности на какое-то число N, и теперь хотим найти строку S, алгоритмическая сложность которой больше, чем N.

Для этого мы можем воспользоваться алгоритмом Британского музея , написав программу, которая будет реализовывать следующее вычисление.

1. Программа порождает последовательно все теоремы нашей теории.
2. Для каждой новой теоремы она проверяет, имеет ли та вид $I(S)>N$ для произвольной заранее не фиксированной строки S.
3. Если нет, то программа порождает следующую теорему.
4. Если да, то программа выводит на печать строку S и останавливается.

Для определенности будем считать, что сама программа и все данные представлены в алфавите 0 и 1. Тогда длина программы равна

$$\lceil \log_2(N) \rceil+1+C$$

$\lceil \log_2(n) \rceil$ - это целая часть логарифма по основанию 2 от числа n. Легко проверить, что $\lceil \log_2(8) \rceil + 1 = 3 + 1 = 4 =$ длина(«1000»), а $\lceil \log_2(7) \rceil + 1 = \lceil 2.807355\ldots \rceil + 1 = 2+1 = 3 =$ длина(«111»)

$\lceil \log_2(N) \rceil + 1$ - количество бит, которые необходимы программе для хранения числа N.

C - некоторая константа, представляющая количество бит, необходимых программе для хранения аксиом теории и порождения ее теорем.

Поскольку предполагается, что наша теория корректна, т.е. в ней доказуемы лишь истинные формулы, выполняются следующее неравенства:

- $I(S) > N$
- $\lceil \log_2(N) \rceil + 1 + C \geq I(S)$

Комментариев требует лишь второе неравенство. Его истинность следует из того, что наша программа печатает строку S и останавливается, а I(S) - размер минимальной программы, которая печатает строку S и останавливается. Очевидно, что $\lceil \log_2(N) \rceil + 1 + C$ не может быть меньше I(S). Отсюда и следует истинность неравенства.

Из двух неравенств мы заключаем, что

$$\lceil \log_2(N) \rceil + 1 + C > N$$

Чем замечательно это неравенство?

1. Во-первых, тем, что строка S в нем уже не фигурирует.
2. Во-вторых, тем, что, как известно из школьного курса арифметики, функция логарифма растет гораздо медленнее ее аргумента. Из этого свойства логарифмов следует, что для достаточно больших N данное неравенство уже не будет выполняться. Каждый сам может проверить, что оно нарушается, например, при N=2*C+2.

Мы доказали следующую теорему.

> *Для всякой формальной теории T, в которой представима функция алгоритмической сложности I(x), существует такое число N_T, что ни одна теорема вида $I(S) > N_T$ не доказуема, хотя множество строк, для которых истинно утверждение $I(S) > N_T$, бесконечно. Множество теорем вида $I(S) > n$, доказуемых в теории T, конечно.*

Это и есть теорема Чейтина.

КОММЕНТАРИИ

Программное представление теорий

Мы привыкли к тому, что теория задается набором аксиом и правил вывода, а отождествляется с множеством всех теорем. Такой подход можно назвать платонистическим. Чейтин в данной теореме и других публикациях развивает подход к представлению теории как программы (не алгоритма), последовательно порождающей ее теоремы. Появляются новые параметры, которые могут быть использованы для исследования свойств теории. Алгоритму в отличие от программы невозможно сопоставить длину, а для программы это возможно. Размер минимальной программы может быть взят в качестве оценки дескриптивной сложности теории. Это лишь один из параметров, но полезными могут оказаться и другие. Естественно, что программа понимается не как конкретная реализация на одном из языков для обычных компьютеров, а теоретически - как программа машины Тюринга, МНР-программа, программа на чистом Лиспе, λ-терм или терм комбинаторной логики, которые мы в состоянии исследовать точными методами.

Рассматривая теорию как мн-во теорем, порождаемых некоторым алгоритмом, в качестве дополнительного параметра используют число шагов (применений правил вывода) для доказательства теоремы. Это уже есть, и в этом направлении доказан ряд интересных метатеорем. Например, для некоторых противоречивых теорий противоречие выводится за астрономически большое число шагов. С практической точки зрения такие теории могут рассматриваться как непротиворечивые.

Ошибки критиков

Очень часто причиной критики в отношении результатов Чейтина является их элементарное непонимание. Чейтин показал, что существуют строки произвольной сложности, которые могут быть закодированы программами приблизительно их же длины. Ему возражают следующим образом. «Берем теорию с равенством и, используя аксиому $\forall x(x=x)$, за один шаг снятием квантора получаем $S=S$ для произвольной строки S. Из этой теоремы извлекаем саму строку S». Все верно. Но дело в том, что требуется извлечь заранее заданную строку. Для этого в программе должна уже содержаться информация для идентификации строки. Можно дать человеку с улицы аксиому $\forall x(x=x)$ и попросить его вывести из нее роман «Война и мир». Это у него не получится, если у него заранее не будет текста романа в полном или закодированном виде.

В продолжение сказанного. Любую бинарную строку можно построить всего лишь из двух символов «0» и «1». Легко написать программу, которая будет последовательно без повторений генерировать все такие строки, в том

числе и совершенно случайной структуры. Но если мы хотим сгенерировать конкретную строку, для этого может понадобиться написать программу очень большой длины. Для этого мы могли бы к предыдущей программе генерации строк в качестве довеска добавить тест на то, что требуемая строка порождена, и программа должна остановиться. Этот тест должен содержать в себе всю информацию об этой строке, и именно от него будет зависеть размер программы. Т.е. есть принципиальное различие между *построением объекта и его именованием*. Построить зачастую очень легко, а именовать без потери информации - сложно.

Операциональные определения

Другой (логический) взгляд на теорию алгоритмической сложности. Что такое самораспаковывающийся архив некоторого файла? В логических терминах - это хорошо знакомый дефиниенс в операциональном определении.

Исходный файл ---> Архив
Дефиниендум ---> Дефиниенс

Мы кликаем мышкой по самораспаковывающемуся архиву и тем самым запускаем операциональное определение. Оно трансформирует дефиниенс в дефиниендум.

Например, у нас есть операциональное определение «окружности диаметра *d* с центром в точке *p*». Дефиниенсом будет процедура, которая строит окружность с центром в этой точке *p* и диаметра *d*.

В этом смысле *алгоритмическая сложность некоторого дефиниендума* - это *размер самого короткого дефиниенса в операциональном определении этой строки*.

Если дефиниендум сложен (роман «Война и мир»), то единственный способ дать ему минимальное операциональное определение заключается в том, чтобы спрятать/упаковать дефиниендум в дефиниенс.

Тогда теорема Чейтина может быть переформулирована следующим образом. В любой формальной теории операциональных определений мы можем лишь для конечного числа дефиниендумов установить, что их дефиниенс минимален по длине. Для бесконечного мн-ва дефиниендумов мы не можем быть уверены, что дали им самое удачное/короткое операциональное определение.

$N_{Berry} =_{def}$ Наименьшее число, которое нельзя определить менее чем 1000 символов.

Число Ω

С именем Чейтина также часто ассоциируют так называемое число Ω. Если сказать неформально, это вероятность того, что случайно выбранная программа завершает вычисление и останавливается. Как мы знаем, проблема остановки для программ неразрешима. Это был один из главных результатов Тюринга в статье 1936. Заслуга в том, что он строго определяет это число, хоть оно и невычислимо.

$$\Omega = \Sigma\, 2^{-|p|},\ p \text{ - программа останавливается}$$

Почему это число так интересно? Дело в том, что, как мы знаем, действительных чисел континуум, а программ лишь счетное мн-во. Подавляющее число тех действительных чисел, определение которых нам известно, вычислимы - число π, число *e*, √2, √3 и т.д. Действительных чисел, которые невычислимы, нам известны лишь единицы. Это выглядит парадоксально, так как, с точки зрения теории мн-в, их неизмеримо больше, чем вычислимых, но найти или, что то же самое, определить их очень сложно. Чейтин строго определил число Ω и показал его невычислимость, случайность его внутренней структуры и принципиальную несжимаемость. Сделать это не так просто, как может показаться. С другой стороны, понятно, что все эти свойства числа Ω впрямую зависят от результатов Тюринга.

Законы науки

Исходная идея философии и науки заключается в том, чтобы из небольшого числа начал объяснить все многообразие явлений окружающего нас мира. Фалес сказал, что *все есть вода,* и предпринял первую попытку. Парменид заметил, что *лишь умопостигаемое бытие, а не данные органов чувств, обладает действительной реальностью*, и тем самым, по выражению Б.Рассела, *изобрел метафизику, основанную на логике.*

Чейтин очень любит цитировать слова Лейбница и Г.Вейля, где они высказываются по поводу того, что можно считать законом природы (физика, математика). Они писали о том, что законы науки не должны быть сложнее тех данных, которые они объясняют. В противном случае понятие закона становится расплывчатым, ибо любой набор данных может быть ими охвачен. Через любой набор точек можно провести кривую, которая будет их соединять. Всегда можно просто запомнить исходные данные в виде таблицы и назвать их очередным законом науки. К настоящим законам науки должен быть применим принцип простоты (из двух теорий выбираем более простую; бритва Оккама).

Чейтин пишет, что возникает определенное противоречие. Подавляющее большинство строк символов принципиально несжимаемы. Это означает, что мы не можем свести их к более простым строкам. Внутри сложных строк нет никаких внутренних закономерностей, которые позволили бы их сжать. У нас не остается никакого другого выбора как принять их в качестве новых законов

природы\науки. Мы принимаем их без всякого обоснования, а просто потому, что считаем это целесообразным. Примеров в истории более чем достаточно - аксиома параллельных, P=NP, тезис Тюринга, аксиома выбора, гипотеза континуума.

Здесь есть небольшой нюанс, на котором сам Чейтин не акцентирует внимания, но который необходимо учитывать.

Как я говорил ранее, мы можем с помощью очень простых программ последовательно порождать любые строки произвольной сложности. Точно так же мы можем с помощью алгоритма Британского музея порождать любые теоремы арифметики. Принципиальной разницы между строками и теоремами арифметики нет. Несжимаемость отдельных строк в случае арифметики следует, как мне кажется, понимать в том смысле, что даже доказав многие теоремы, мы не можем объяснить, понять, почему они являются истинными утверждениями. Мы не сможем объяснить их путем сведения к более простым и более понятным теоремам. Единственным объяснением будет то, что они получены в результате доказательства, а это исключительно формальный критерий.