Классика программирования

Чарльз Петцольд

Читаем Тьюринга

ДМК ПРЕСС
издательство

Чарльз Петцольд

Читаем Тьюринга

Путешествие по исторической статье Тьюринга о вычислимости и машинах Тьюринга

Charles Petzold

The Annotated Turing

Guided Tour through Alan Turing's Historic Paper on Computability and the Turing Machine

Wiley Publishing, Inc.

Чарльз Петцольд

Читаем Тьюринга

Путешествие по исторической статье Тьюринга о вычислимости и машинах Тьюринга

ДМК ПРЕСС
издательство
Москва, 2014

УДК 004.3.01:510.5
ББК 32.97
П29

Петцольд Ч.
П29 Читаем Тьюринга. Путешествие по исторической статье Тьюринга о вычислимости и машинах Тьюринга / пер. с анг. Борисова Е. В., Чернышова Л. Н. – М.: ДМК Пресс, 2014. – 440 с.: ил.

ISBN 978-5-97060-010-8

Книга, которую вы держите в руках, принадлежит перу известного американского популяризатора Чарлза Петцольда. В ней автор исследует главную работу Алана Тьюринга, посвященную проблеме разрешимости. Именно в этой работе впервые появились знаменитые машины Тьюринга, ставшие на многие годы универсальной теоретической концепцией computer science.

Автор тонко и деликатно проведет вас по самым потаенным уголкам, из которых родились на свет современные компьютеры и современное программное обеспечение.

Читателя ждет захватывающее путешествие в прошлое, из которого получилось наше настоящее и развивается будущее.

УДК 004.3.01:510.5
ББК 32.97

ISBN 978-0-470-22905-7 (анг.)
ISBN 978-5-97060-010-8 (рус.)

Содержание

Введение

Все, кто изучали историю, технологию или теорию вычислительных машин, вероятно, сталкивались с понятием *машины Тьюринга*. Машина Тьюринга – это воображаемый, не совсем даже гипотетический, компьютер, изобретенный в 1936 году английским математиком Аланом Тьюрингом (1912–1954) для того, чтобы помочь решить проблему математической логики. В качестве побочного продукта Тьюринг создал новое поле исследований, известное как теория вычислений или вычислимость, которая изучает возможности и ограничения компьютеров.

Хотя машина Тьюринга – довольно неправдоподобный компьютер, она полезна тем, что является чрезвычайно простой. Элементарная машина Тьюринга выполняет лишь несколько простых операций. Если бы эта машина делала чуть меньше, чем она делает, она не делала бы вообще ничего. Однако за счет комбинаций этих простых операций машина Тьюринга может выполнить любое вычисление, на которое способен современный цифровой компьютер.

Разобрав компьютер до оснований, мы можем лучше понять его возможности и, что немаловажно, его ограничения. Задолго до того, как было показано, что может компьютер, Тьюринг доказал, чего он *никогда* не сможет сделать.

Машина Тьюринга остается популярной темой для суждений и толкований. (Попробуйте набрать «машина Тьюринга» в вашем любимом поисковике в Интернете.) Однако я подозреваю, что оригинальная статья Алана Тьюринга, описывающая его изобретение, читается редко. Возможно, такое пренебрежение связано с ее заголовком «О вычислимых числах применительно к Entscheidungsproblem». Даже если вы можете выговорить без запинки это слово, сделав ударение на втором слоге и произнеся его как «шай», и знаете, что оно значит («проблема разрешимости»), у вас все же остается подозрение, что Тьюринг предполагает предварительное знакомство своего читателя с трудными немецкими математическими рукописями. Беглый просмотр статьи, где для обозначения состояний машины используется готический шрифт, отнюдь не помогает унять эти страхи. Так может ли читатель в наши дни взяться за статью, опубликованную 70 лет назад в *Трудах Лондонского математического общества*, и остаться на плаву достаточно долго, чтобы в полной мере проникнуться ею и, возможно, даже получить от нее удовольствие?

Обо всем этом – наша книга. Она содержит оригинальную 36-страничную статью Тьюринга «О вычислимых числах применительно к Entscheidungsproblem»[1] и последующие трехстраничные Исправления[2], а также вспомогательные главы и развернутые комментарии.

Чтение оригинальной статьи Тьюринга – это уникальное путешествие в его изобретательный и захватывающий образ мыслей, когда он создает машину, которая имела такие далеко идущие последствия для вычислений и, больше того, для нашего понимания ограничений математики, человеческого мышления и, возможно даже, природы вселенной. (Конечно, самого термина «машина Тьюринга» в статье Тьюринга нет. Он назвал ее «вычислительной машиной». Но термин «машина Тьюринга» использовался уже с начала 1937 года[3] и с тех пор остается общепринятым.) В своих комментариях к статье Тьюринга я счел полезным часто прерывать его изложение своими пояснениями и уточнениями. Я пытался (не всегда успешно) не прерывать его посреди предложения. В большей части своих объяснений я сохранял терминологию и систему обозначений самого Тьюринга, но время от времени ощущал необходимость ввести термины, которые Тьюринг не использует, но которые я счел полезными при объяснении его работы.

Текст статьи Тьюринга выделяется затененным фоном:

> Мы избежим путаницы, говоря чаще о вычислимых последовательностях, нежели о вычислимых числах.

Мы (то есть мой издатель и я) попытались сохранить типографский набор и верстку оригинальной статьи, за исключением некоторых причуд (таких, как пробелы перед двоеточиями), которые вызывали «панику» в современных текстовых редакторах. Сохранены все оригинальные разрывы строк. В статье Тьюринга есть несколько опечаток, ошибок и пропусков. Они оставлены без изменений, но я обращаю на них внимание в своих комментариях. Тьюринг часто

---

[1] Turing A. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 2nd series, Vol. 42 (1936), pp. 230–265.

[2] Turing A. On Computable Numbers, with an Application to the Entscheidungsproblem. A Correction, *Proceedings of the London Mathematical Society*, 2nd series, Vol. 43 (1937), pp. 544–546.

[3] Alonzo Church, review of «On Computable Numbers, with an Application to the Entscheidungsproblem», *The Journal of Symbolic Logic*, Vol. 2, No. 1 (Mar. 1937), 42–43.

ссылается на предыдущие части своей статьи путем указания номера страницы оригинала в журнале. Я оставил эти ссылки в покое, но снабдил свои комментарии подсказкой для поиска определенной страницы в этой книге. Иногда в тексте Тьюринга вы увидите номер в квадратных скобках:

> Если буквы заменить числами, как в §5, мы получим числовое [243]
> описание полной конфигурации, которое можно назвать ее описательным номером.

Это – конец одной и начало следующей страницы оригинала с ее номером. Мои сноски – номерные; сноски Тьюринга – символьные и тоже оттенены фоном. Если вы удалите страницы этой книги, вырезав и выбросив все, что не затенено, а потом склеите вместе остатки, у вас получится полная статья Тьюринга и один несчастный автор. Но гораздо интереснее, наверное, сначала прочитать эту книгу, а потом вернуться назад и прочитать саму статью Тьюринга без моих грубых вмешательств.

Статья Тьюринга расположена на страницах 84–362 этой книги, а исправления к ней – на страницах 349–362. Статья Тьюринга делится на 11 разделов (и приложение), которые начинаются на следующих страницах книги:

Первоначальным толчком к написанию Тьюрингом этой статьи было намерение решить задачу, сформулированную немецким математиком Давидом Гильбертом (1862–1943). Гильберта интересовал общий процесс определения доказуемости произвольных утвержде-

ний в математической логике. Нахождение этого «общего процесса» было известно как Entscheidungsproblem (с нем. – «решаемость задачи»). Хотя побуждением к написанию статьи для Тьюринга была, конечно же, Entscheidungsproblem, на деле бóльшая часть статьи – о вычислимых числах. По определению Тьюринга, это числа, которые могут быть вычислены машиной. Исследование вычислимых чисел составляет первые 60% статьи Тьюринга, которые могут быть прочитаны и поняты без знания работ Гильберта по математической логике или Entscheidungsproblem.

Различие между вычислимыми числами и «вещественными числами» крайне важно для изложения Тьюринга. По этой причине первые главы данной книги представляют собой предварительную подготовку к нашей классификации чисел, охватывающей целые числа, рациональные числа, иррациональные числа, алгебраические числа и трансцендентные числа, все из которых относятся также к вещественным числам. Я старался не полагаться на какие-либо предварительные знания, более сложные, чем математика средней школы. Понимая, что некоторых читателей от радостей средней школы могут отделять несколько десятилетий, я попытался освежить эти воспоминания. Прошу простить, если мое педагогическое рвение привело к таким объяснениям, которые унижают или оскорбляют кого-то.

Хотя я подозреваю, что эта книга будет читаться главным образом специалистами по информатике, программистами и прочими «технарями», я постарался доставить радость и непрограммистам, используя дружелюбный жаргон и терминологию из области искусства. Статья Тьюринга – это «один из интеллектуальных ориентиров прошедшего столетия»[1], и я надеюсь, что данная книга сделает эту статью доступной еще более широкой аудитории.

Чтобы угодить запросам разных читателей, я поделил эту книгу на четыре части:

- Часть I («Основы») охватывает исторический и математический фон, который необходим, чтобы приступить к чтению статьи Тьюринга.
- Часть II («Вычислимые числа») содержит бóльшую часть статьи Тьюринга и будет особенно полезна читателям, интересующимся машиной Тьюринга и вопросами вычислимости.

---

[1] John P. Burgess, предисловие к George S. Boolos, John P. Burgess, and Richard C. Jeffrey, *Computability and Logic*, fourth edition (Cambridge University Press, 2002), xi.

- Часть III («Entscheidungsproblem») начинается совсем коротким введением в математическую логику и продолжается разбором оставшейся части статьи Тьюринга.
- В части IV («И далее») обсуждается, как случилось, что машина Тьюринга стала важным инструментом для понимания компьютеров, человеческого сознания и самой вселенной.

Математическое содержание части III, конечно, гораздо сложнее, чем в предыдущих главах, и излагается в более быстром темпе. Те, кто не очень интересуется значением статьи Тьюринга для математической логики, могли бы даже при желании пропустить пять глав части III и перейти сразу к части IV.

Эта книга затрагивает несколько больших разделов математики, включая вычислимость и математическую логику. Я выбрал и предпочел только наиболее важные для понимания статьи Тьюринга темы и понятия. Многие детали опущены, и эта книга не заменит глубины и строгости, которую вы найдете в книгах, специально посвященных вычислимости и логике. Те читатели, которые намерены глубже покопаться в этих захватывающих областях исследований, могут руководствоваться библиографией.

За всю свою жизнь Алан Тьюринг опубликовал около 30 работ и статей[1], но не написал ни одной книги. Две статьи Тьюринга стали причиной его неугасающей известности. Конечно же, первая из них – «О вычислимых числах». Вторая – гораздо менее техническая статья под названием «Вычислительные машины и интеллект» (опубликована в 1950 году), где Тьюринг придумал то, что теперь в искусственном интеллекте называется тестом Тьюринга. Суть его в том, что если машина может заставить нас поверить, что она – человек, то мы с большой вероятностью должны считать, что она обладает интеллектом.

Машина Тьюринга и тест Тьюринга – это две заявки Алана Тьюринга на литературное и культурное бессмертие. На первый взгляд, они могут казаться двумя совершенно разными понятиями, но это не так. Машина Тьюринга – это, в сущности, попытка описать чисто механически действия человека, выполняющего математический ал-

---

[1] Эти и другие документы доступны в четырехтомнике *The Collected Works of A. M. Turing* (Amsterdam: Elsevier, 1992, 2001). Большая часть очень ценного материала была собрана Б. Джеком Коуплендом (B. Jack Copeland) в *The Essential Turing* (Oxford University Press, 2004) и в *Alan Turing's Automatic Computing Engine* (Oxford University Press, 2005). Первая книга содержит работы и статьи о машине Тьюринга, а вторая – о проекте компьютера ACE второй половины 1940-х годов.

горитм; тест Тьюринга – это оценка человеком действий компьютера. С самых ранних своих математических исследований и до самых поздних Тьюринг изучал связь между человеческим разумом и вычислительной машиной способом, который продолжает оставаться захватывающим и побуждающим.

Можно обсуждать работы Тьюринга, ничего не говоря о Тьюринге-человеке, и многие учебники по вычислимости не утруждают себя подробностями его биографии. Я не счел это возможным здесь. Секретная работа Тьюринга по криптоанализу во время Второй мировой войны, его участие в перспективных компьютерных проектах, его размышления об искусственном интеллекте, его сексуальная ориентация, его арест и судебное преследование за преступление в «грязной непристойности» и его ранняя смерть в результате явного самоубийства в возрасте 41 года – все это заслуживает внимания.

Моя работа по изложению основных фактов жизни Тьюринга была очень облегчена замечательной биографией *Алан Тьюринг: Загадка* (Simon & Schuster, 1983) английского математика Эндрю Ходжеса (род. 1949). Ходжес заинтересовался Тьюрингом отчасти из-за своего собственного участия в освободительном движении геев в 1970-х годах. Написанная Ходжесом биография вдохновила Хью Уайтмора на пьесу под названием *Взлом кода* (1986). На сцене и в сокращенной версии для телевидения в 1996 году роль Алана Тьюринга была исполнена Дереком Якоби.

Подобно ранним английским математикам и компьютерным первопроходцам Чарльзу Бэббиджу (1791–1871) и Аде Лавлейс (1815–1852), Тьюринг стал символом компьютерного века. Премия Тьюринга – это ежегодная награда в 100 000 долларов, вручаемая Ассоциацией вычислительных машин (ACM) за большой вклад в вычислительную технику. Существуют язык программирования Тьюринг (производный от Паскаля) и программное обеспечение «Мир Тьюринга» для сборки машин Тьюринга.

Имя Тьюринга стало почти нарицательным в программировании, причем настолько, что А. К. Дьюдни смог озаглавить свои «Экскурсии по информатике» как *Омнибус Тьюринга* (Dewdney A. K. *The Turing Omnibus: Excursions in Computer Science*. Computer Science Press, 1989). Книга «Западная культура в век компьютеров» Дж. Дэвида Болтера называется *Человек Тьюринга* (Bolter J. D. *Turing's Man: Western Culture in the Computer Age*. University of North Caroline Press, 1984), а критический анализ Брайена Ротмана традиционных математических понятий бесконечности *До бесконечности* получила забавный подза-

головок «Призрак в машине Тьюринга» (Rotman B. *Ad Infinitum: The Ghost in Turing's Machine*. Stanford University Press, 1993).

Алан Тьюринг вызывал некоторый научный интерес и за пределами математики и информатики. Сборник *Новый взгляд: Странные толкования в фантастике* (*Novel Gazing: Queer Readings in Fiction*, Duke University Press, 1997) содержит эссе Тайлера Кертина под заглавием «'Дурман' машин: *Neuromancer*, интернет-сексуальность и тест Тьюринга» (Tyler Curtain. *The 'Sinister Fruitiness' of Machines: Neuromancer*, Internet Sexuality, and the Turing Test). Заголовок доктора Кертина ссылается на известный «киберпанк»-роман Уильяма Гибсона *Neuromancer* (Gibson W. *Neuromancer*. Ace, 1984), в котором полиция Тьюринга помогает убедиться в том, что существа с искусственным интеллектом не стремятся повысить свой собственный интеллект.

Тьюринг появлялся в названиях еще нескольких романов. Марвин Минский (известный исследователь MIT в области искусственного интеллекта) сотрудничал с писателем-фантастом Гарри Харрисоном в *Варианте Тьюринга* (Harry Harrison, Marvin Minsky. *The Turing Option*. Warner Books, 1992), а профессор информатики из Беркли Христос Х. Пападимитриу разрешился сочинением *Тьюринг* (*Роман о вычислении*) (Christos H. Papadimitriou. *Turing (A Novel About Computation*. MIT Press, 2003).

В *Бреде Тьюринга* боливийского романиста Эдмундо Пас Сольдена (Soldán Edmundo Paz. *Turing's Delirium*. Trans. Lisa Carter. Houghton Mifflin, 2006) криптоаналитик по прозвищу Тьюринг вскрывает опасности использования его навыков для коррумпированного правительства. В романе Жанны Левин *Сумасшедшие мечты машин Тьюринга* (Levin Janna. *A Madman Dreams of Turing Machines*. Knopf, 2006) вымышленные жизни Алана Тьюринга и Курта Гёделя странно взаимодействуют в пространстве и времени.

Алан Тьюринг – персонаж книг *Криптономикон* Нила Стефенсона (Stephenson Neal. *Cryptonomicon*. Avon, 1999), *Загадка* Роберта Харриса (Harris Robert. *Enigma*. Hutchinson, 1995), *Кембриджский Квинтет: Работа научного предположения* Джона Л. Касти (Casti John L. *The Cambridge Quintet: A Work of Scientific Speculation*. Perseus Books, 1998) и, конечно, *Гёдель, Эшер, Бах* Дугласа Хофштадтера (Hofstadter Douglas. *Gödel, Escher, Bach*. Basic Books, 1979). От лица Алана Тьюринга ведется рассказ в новелле «Доктор Кто» Пола Леонарда – одной из частей *Теста Тьюринга* (Leonard Paul. Dr. Who. *The Turing Test*, BBC, 2000).

Хоть и приятно, что Алан Тьюринг почитается такими разными способами, существует опасность, что настоящая работа Тьюринга

окажется забытой. Даже тех, кто изучал теорию вычислений и полагает, что знает о машинах Тьюринга все, ждет, я надеюсь, несколько сюрпризов при знакомстве с самой первой машиной Тьюринга, построенной самим автором.

* * *

Эта книга была задумана в 1999 году. Потом в течение следующих пяти лет я писал понемногу и нерегулярно. Первые одиннадцать глав в основном были закончены в 2004 и 2005 годах. Последние семь глав я написал в 2007 и 2008 годах, прервав работу лишь для того, чтобы жениться (наконец!) на моей давней лучшей подруге и любви моей жизни Дирдре Синнотт.

Большое спасибо Лондонскому математическому обществу за разрешение перепечатать в полном объеме статью Алана Тьюринга «О вычислимых числах применительно к Entscheidungsproblem».

Уолтер Вильямс и Ларри Смит прочли ранние черновики этой книги, обнаружив ряд ошибок и предложив несколько полезных улучшений.

Людям из Wiley я вечно благодарен за их работу по превращению этого любимого моего проекта в реально изданную книгу. Крис Вебб продвигал книгу, исполняющий редактор Кристофер Ривера и производственный редактор Анджела Смит решили множество конструктивных и типографских проблем, а технический редактор Питер Бонфэнти помог мне приложить чуть больше усердия к техническим материалам. Многие из Wiley работали бескорыстно, помогая сделать эту книгу как можно лучше. Все остальные недостатки, недочеты или ужасные ошибки можно отнести только к автору.

Любой автор стоит на плечах тех, кто пришел раньше. В избранной библиографии перечислены лишь несколько из множества книг, которые помогали мне при написании этой книги. Кроме того, я хотел бы поблагодарить персонал Публичной библиотеки Нью-Йорка и особенно Библиотеку по науке, промышленности и бизнесу (Science, Industry, and Business Library, SIBL). Для получения оригинальных статей я широко пользовался JSTOR и обнаружил, что Wikipedia, Google Book Search и MathWorld Уолфрэма тоже полезны.

* * *

Информацию и ресурсы, относящиеся к этой книге, можно найти на веб-сайте www.TheAnnotatedTuring.com.

Чарльз Петцольд,
Нью-Йорк Сити и Роско, Нью-Йорк
Май, 2008

Часть I

Основы

Глава 1

Прах Диофанта покоится в этой могиле

Много веков назад в древней Александрии старик должен был хоронить своего сына. Убитый горем, он утешал себя составлением большого сборника алгебраических задач с решениями в книге, названной им *Арифметика* (*Arithmetica*). Вот, пожалуй, и все, что известно о Диофанте из Александрии, и большая часть этого исходит от загадки, которая, как полагают, была написана его близким другом вскоре после его смерти[1]:

> Прах Диофанта покоится в этой могиле. И она, о чудо, искусно поведает нам, сколь долог был его век. Шестую часть жизни Бог одарил его детством; когда минула еще одна двенадцатая часть, пушком покрылись его щеки; спустя седьмую долю жизни, Он зажег его брачную свечу, а на пятом году его брака Он послал ему сына. Увы, поздний и слабый ребенок, достигнув половины жизни отца своего, был забран холодной могилой. Еще четыре года горе свое утешал он наукой о числах, и тут конца жизни своей он достиг[2].

Эпитафия немного неоднозначна в отношении смерти сына Диофанта. Как в ней сказано, тот умер, «достигнув половины жизни отца своего», но что значит эта половина жизни отца – половина возраста Диофанта – момент смерти его сына или на момент его собственной смерти? Задачу можно решать любым способом, но вторая версия – сын Диофанта прожил половину лет, которые в итоге прожил сам

---

[1] Thomas L. Heath, *Diophantus of Alexandria: A Study in the History of Greek Algebra*, second edition (Cambridge University Press, 1910; Dover Publications, 1964), 3.

[2] *Greek Mathematical Works II: Aristarchus to Pappus of Alexandria* (Loeb Classical Library No. 362), translated by Ivor Thomas (Harvard University Press, 1941), 512–3.

Диофант, – имеет хорошее, простое решение в целых числах без долей лет.

Пусть x – это общее количество лет, прожитых Диофантом. Каждый отрезок жизни Диофанта – это либо доля его полной жизни (например, $x/6$ – это его детство), либо целое число лет (например, до рождения сына он был женат 5 лет).

Сумма всех этих периодов жизни Диофанта равна x, поэтому загадку можно записать просто алгебраически:

$$\frac{x}{6}+\frac{x}{12}+\frac{x}{7}+5+\frac{x}{2}+4=x.$$

Наименьшее общее кратное знаменателей этих дробей – 84, поэтому умножим на него все члены уравнения слева и справа:

$$14x + 7x + 12x + 420 + 42x + 336 = 84x.$$

Собрав множители x в левой части, а константы – в правой, получим:

$$84x – 14x – 7x – 12x – 42x = 420 + 336.$$

Или:

$$9x = 756.$$

А само решение:

$$x = 84.$$

Итак, до 14 лет Диофант был мальчиком, а через 7 лет смог, наконец, отрастить бороду. Спустя двенадцать лет, в возрасте 33 года, он женился, а через 5 лет у него родился сын. Сын умер в возрасте 42 года, когда Диофанту было 80, а сам Диофант умер 4 года спустя.

На самом деле есть более быстрый способ решения этой загадки: если задуматься глубже, то можно понять, что у автора загадки нет намерения утруждать нас дробными числами. «Двенадцатая часть» и «седьмая часть» жизни Диофанта должны быть целыми числами, поэтому возраст в год его смерти делится одинаково и на 12, и на 7 (и еще на 6 и на 2). Вот и умножьте 12 на 7, чтобы получить 84. Для зрелого возраста это близко к истине и потому, наверно, правильно.

Возможно, Диофант умер в 84 года, но крайне важный исторический вопрос – *когда?* Когда-то оценки периода жизни Диофанта

колебались от 150 года до н. э. до 280 года н. э.[1] Это довольно расплывчатый диапазон: он определенно ставит Диофанта после таких ранних александрийских математиков, как Евклид (блистал ок. 295 г. до н. э.[2]) и Эратосфен (ок. 276–195 г. до н. э.), но может сделать его современником Герона Александрийского (известного и как Герой и блиставшего в 62 г. н. э.), который написал книги по механике, пневматике и автоматах и, видимо, изобрел прообраз парового двигателя. Возможно, Диофант знал и александрийского астронома Птолемея (ок. 100–170 г. н. э.), известного главным образом по *Альмагесту*, содержавшему первую тригонометрическую таблицу и заложившему основы математики движения небесных тел, которая не была убедительно опровергнута вплоть до революции Коперника в XVI–XVII веках.

К сожалению, у Диофанта, видимо, не было контактов с другими александрийскими математиками и учеными. В последние примерно сто лет исследователи сходятся во мнении, что Диофант творил около 250 года н. э., и его самый главный труд *Арифметика* датируется, скорее всего, этим временем. Так что время рождения Диофанта приходится примерно на время смерти Птолемея. Пол Теннери, который редактировал каноническое греческое издание *Арифметики* (издано в 1893–1895 гг.), отмечал, что работа была посвящена «уважаемому Дионисию». Несмотря на распространенное имя, Теннери догадался, что это был тот самый Дионисий, который был главой школы Катехизиса в Александрии в 232–247 годах, а затем Епископом Александрийским в 248–265 годах. Таким образом, Диофант мог быть христианином[3]. Если это так, то можно усмотреть злую иронию в том, что один из ранних (но потерянных) комментариев к *Арифметике* был написан Ипатией (Hypatia) (ок. 370–415 гг.), дочерью Теона (Theon) и последней из великих александрийских математиков, которая была забита толпой христиан, настроенной против ее «языческой» философии.

Математика Древней Греции была традиционно сильнейшей в геометрии и астрономии. Диофант был этническим греком, но его отли-

---

[1] Эти даты до сих пор сохраняются в Simon Hornblower and Antony Sprawforth, eds., Oxford Classical Dictionary, revised third edition (Oxford University Press, 2003), 483.

[2] Все остальные даты александрийских математиков – из Charles Coulston Gillispie, ed., Dictionary of Scientific Biography (Scribners, 1970).

[3] Heath, Diophantus of Alexandria, 2, note 2. Сам же автор этой книги, похоже, сомневается в этом.

чало то, что он утешал свое горе после смерти сына «наукой о числах», или, как мы теперь ее называем, *алгеброй*. Видимо, от него исходит несколько алгебраических новшеств, включая использование символов и сокращений, означавших переход от словесной формулировки задачи к современной алгебраической нотации.

Шесть книг *Арифметики* (считается, что изначально их было 13) представляют собой задачи нарастающей сложности, большинство из которых едва ли труднее, чем загадка о возрасте Диофанта. Нередко задачи Диофанта имеют много неизвестных. Некоторые из его задач *неопределенны*, то есть имеют более одного решения. Все, кроме одной, задачи из *Арифметики* абстрактны в том смысле, что они строго числовые и не относятся к объектам реального мира.

Еще один элемент абстракции у Диофанта – возведение в степень. До того времени математикам были известны степени 2 и 3. Квадраты были нужны для вычисления площадей, а кубы – для вычисления объемов тел. Но Диофант допускал в своих задачах и более высокие степени: степень 4 (которую он назвал «квадрат квадрата»), 5 («квадрат куба») и 6 («куб куба»). Такие степени не имели физической аналогии в мире, который был известен Диофанту, и указывали на то, что Диофант не очень беспокоился о практичности своей математики. Это была просто занимательная математика без цели, но для развития ума.

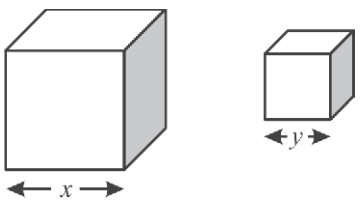
Вот первая задача из Книги IV[1]. Диофант формулирует ее сначала общими словами:

> Разделить заданное число на два куба так, чтобы сумма их сторон была равна другому заданному числу.

Затем он задает эти два числа:

> Заданное число – 370, заданная сумма сторон – 10.

Геометрически он имеет дело с двумя кубами разных размеров. Как современные алгебраисты, мы с вами могли бы обозначить стороны двух кубов x и y:



[1] Heath, Diophantus of Alexandria, 168.

Эти две стороны (x и y) составляют в сумме 10. Объемы двух кубов (x^3 и y^3) в сумме дают 370. Теперь запишем два уравнения:

$x + y = 10;$
$x^3 + y^3 = 370.$

Из первого уравнения следует, что y равно $(10 - x)$, и его можно подставить во второе уравнение:

$x^3 + (10 - x)^3 = 370.$

Теперь трижды перемножаем $(10 - x)$ и уповаем, чтобы кубы в конечном счете сократились:

$x^3 + (1000 + 30x^2 - 300x - x^3) = 370.$

К счастью, так и происходит, и после небольших группировок получаем:

$30x^2 - 300x + 630 = 0.$

Коэффициенты в левой части имеют общий множитель, поэтому желательно сократить обе части на 30:

$x^2 - 10x + 21 = 0.$

Теперь все почти готово. У нас есть два варианта. Если вспомнить формулы квадратных уравнений[1], то можно применить их. Или если еще свеж опыт решения подобных уравнений, можно, пристально взглянув на него и подумав некоторое время, разложить его, как по волшебству, следующим образом:

$(x - 7)(x - 3) = 0.$

Таким образом, длины двух сторон – 7 и 3. В сумме они дают 10, а их кубы – 343 и 27 – дают в сумме 370.

Диофант решает задачу совсем не так, как мы с вами. В сущности, он этого просто не может сделать. Хотя часто в задачах Диофанта много неизвестных, его система обозначений позволяет ему представлять только одно неизвестное. Однако он очень изобретательно восполняет данный недостаток. Вместо того чтобы обозначать стороны двух кубов как x и y, он говорит, что эти две стороны равны $(5 + x)$ и $(5 - x)$. Две эти стороны выражаются посредством одного неизвест-

---

[1] Для $ax^2 + bx + c = 0$ решение $x = \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}$.

ного x, и они на самом деле дают в сумме 10. Тогда он может возвести их в куб и приравнять сумму к 370:

$$(5 + x)^3 + (5 - x)^3 = 370.$$

Теперь это выглядит хуже того, с чем мы уже сталкивались, но стоит только развернуть кубические скобки, как члены уравнения начинают сокращаться, как сумасшедшие, и у нас остается лишь:

$$30x^2 + 250 = 370.$$

После некоторых простейших перегруппировок и последующего деления на 30 оно сводится к

$$x^2 = 4.$$

Или x равняется 2. Так как стороны – это $(5 + x)$ и $(5 - x)$, то их величины – 7 и 3.

Умение Диофанта решать такие задачи с меньшими, чем современный студент, усилиями – результат его потрясающей способности выражать должным образом две величины посредством одной переменной. Сработает ли этот метод в следующей задаче? Может, да. А может, нет. Разработка общих методов решения алгебраических уравнений – это то, чего у Диофанта, по сути дела, *нет* вообще. Как заметил один математик, «каждый вопрос требует весьма специфического метода, который зачастую не пригоден даже для очень близких задач. Именно поэтому современному математику даже после изучения ста диофантовых задач будет трудно решить сто первую»[1].

Понятно, что когда Диофант предлагает задачу с суммой кубов 370 и суммой сторон 10, он, конечно же, берет числа не с потолка. Он знает, что эти условия приводят к решению в целых числах. Действительно, термин *диофантово уравнение* появился для обозначения алгебраического уравнения, где допустимы только целочисленные решения. Диофантовы уравнения могут иметь множество неизвестных, и эти неизвестные могут возводиться в целочисленные степени, однако решения (если они есть) – всегда целочисленные. Несмотря на то что при формулировке своих задач Диофант часто использует вычитание, его решения никогда не содержат отрицательных чисел. «Очевидно, что об отрицательных величинах *как таковых*, то есть без

[1] Это цитата Германа Ганкеля (Hermann Hankel) (1874) из книги Heath, Diophantus of Alexandria, 54–55. Другие математики обнаружили общие подходы в методах Диофанта. См. Bashmakova I. G. Diophantus and Diophantine Equations (Mathematical Association of America, 1997), ch. 4.

некоего положительного значения, которое вычитается из чего-то, Диофант понятия не имел»[1]. Так же, как не было у него ни одной задачи с нулевым решением. Ноль у древних греков не считался числом.

Современные читатели Диофанта – особенно те, кто уже знают, что диофантовы уравнения имеют лишь целочисленные решения, – могут слегка удивиться, обнаружив у Диофанта *рациональные числа*. Рациональными их называют не потому, что они логичны или разумны в некотором смысле, а потому, что их можно представить как *отношение* (*ratio*) двух целых чисел. Например,

$$\frac{3}{6}$$

– рациональное число.

Рациональные числа появляются лишь в одной задаче *Арифметики*, которая содержит настоящие объекты реального мира, особенно такие вечно любимые, как вино и деньги. В условии задачи рациональных чисел будто бы нет, но они потребуются при ее решении:

> Человек покупает некоторое количество мер вина: одно вино – по 8, другое – по 5 драхм за меру. Он платит за них *квадратное* число драхм; а если мы добавим к этому числу 60, то получим квадрат, сторона которого есть целое число мер. Определите, сколько мер он купил по каждой цене[2].

Под «квадратным числом» Диофант понимает результат умножения некоторого числа на себя. Например, 25 – это квадратное число, так как оно равно 5 раз по 5.

После страницы расчетов[3] оказывается, что количество мер по 5 драхм – это рациональное число

$$\frac{79}{12},$$

а количество мер по 8 драхм – это рациональное число

$$\frac{59}{12}.$$

Давайте проверим эти результаты. (Проверить решение гораздо легче, чем найти его.) Если умножить 5 драхм на 79/12 и добавить к этому произведение 8 драхм на 59/12, получится, что человек за-

---

[1] Heath, Diophantus of Alexandria, 52–53.
[2] Heath, Diophantus of Alexandria, 224.
[3] Heath, Diophantus of Alexandria, 225.

платил в общей сложности $72^1/_4$ драхмы. Диофант утверждает, что человек заплатил «*квадратное* число драхм», то есть плата должна быть квадратом чего-то. Довольно любопытно, что Диофант считает $72^1/_4$ квадратным числом, раз оно может быть выражено отношением

$$\frac{289}{4},$$

где и числитель, и знаменатель – квадраты 17 и 2 соответственно. Таким образом, $72^1/_4$ – квадрат 17/2, или $8^1/_2$. Дальше Диофант говорит, что «если мы добавим к этому числу 60, то получим квадрат, сторона которого есть целое число мер». И здесь «целое число мер» не связано с целыми числами. Диофант (или, точнее, его английский переводчик сэр Томас Хит (Thomas Heath)) подразумевает под этим *общее* число мер. Добавление 60 к $72^1/_4$ дает в результате рациональное число $132^1/_4$, или

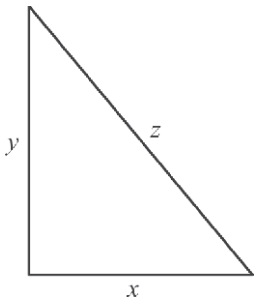
$$\frac{529}{4}.$$

И снова Диофант считает, что это квадрат, потому что и числитель, и знаменатель – квадраты 23 и 2 соответственно. Таким образом, общее число купленных мер – 23/2, или $11^1/_2$, что можно также получить сложением 79/12 и 59/12.

Наверное, самая известная задача в *Арифметике* – это задача 8 из Книги II: «Разбить заданное квадратное число на два квадрата», то есть найти такие x, y и z, что

$$x^2 + y^2 = z^2.$$

У этой задачи есть геометрическая интерпретация, связанная с теоремой Пифагора о соотношении сторон прямоугольного треугольника:



Задача имеет множество решений в целых числах, например x, y и z могут быть равны 3, 4 и 5 соответственно. (Сумма квадратов 9 и 16 дает 25.) Такое простое решение, видимо, не привлекает Диофанта, и он, приняв «заданное квадратное число» (то есть z^2) за 16, получает две другие величины в виде рациональных чисел 144/25 и 256/25. Для Диофанта обе они – конечно же, квадраты. Первая – квадрат 12/5, вторая – квадрат 16/5, а сумма – квадрат 4:

$$\left(\frac{12}{5}\right)^2 + \left(\frac{16}{5}\right)^2 = 4^2.$$

На самом деле не имеет значения, что Диофант допускает решение в рациональных числах, потому что оно равносильно решению в целых числах. Просто умножьте обе части уравнения на 5^2, или на 25:

$$12^2 + 16^2 = 20^2.$$

Или 144 плюс 256 равняется 400. Это, в сущности, то же самое решение, потому что это всего лишь другой способ измерения сторон. В решении Диофанта гипотенуза равна 4. Это могли быть, скажем, 4 дюйма. Теперь воспользуемся другой линейкой с делениями в одну пятую дюйма, и гипотенуза станет равной 20, а стороны – 12 и 16.

Целые числа появились, когда люди начали считать предметы. Рациональные числа, по всей видимости, появились, когда люди начали измерять предметы. Если длина одной моркови – три пальца, а другой – четыре, то длина первой моркови – 3/4 длины второй.

Рациональные величины иногда называют *соизмеримыми*, потому что два объекта с длинами, выраженными рациональными числами, всегда могут быть перемерены в целых единицах. Нужно только сделать новую единицу измерения достаточно малой.

Диофант писал *Арифметику* на греческом языке. Какие-то части его труда были переведены на арабский. Сначала, в 1575 году, их перевели на латынь, а затем, в 1621 году, они вышли в улучшенной редакции, когда к ним проявили интерес европейские математики. Пьер де Ферма (1601–1665) имел собственную копию латинского перевода 1621 года, поля которой он испещрил своими обширными заметками. В 1670 году сын Ферма издал эти заметки вместе с латинской *Арифметикой*. Одной такой заметкой сопровождалась и только что приведенная задача. Ферма писал:

> С другой стороны, невозможно разделить куб на два куба, либо биквадрат [степень 4] на два биквадрата, либо вообще *любую степень, кроме квад-*

рата, – на две с тем же показателем степени. Я нашел этому поистине изумительное доказательство, для которого, однако, поля не достаточно велики[1].

Ферма утверждает, например, что

$$x^3 + y^3 = z^3$$

не имеет решений в целых числах, как не имеет их ни одно подобное уравнение со степенями 4, 5, 6 и т. д. Вообще говоря, это не очевидно. Уравнение

$$x^3 + y^3 + 1 = z^3$$

очень и очень близко к

$$x^3 + y^3 = z^3,$$

но оно имеет множество решений в целых числах, например когда x, y и z равны 6, 8 и 9 соответственно. Уравнение

$$x^3 + y^3 - 1 = z^3$$

тоже весьма похоже, но и оно имеет множество решений в целых числах, например 9, 10 и 12. Почему же эти два уравнения имеют решения в целых числах, а уравнение

$$x^3 + y^3 = z^3$$

их не имеет?

Все задачи из *Арифметики* Диофанта имеют решения, но многие диофантовы уравнения, как у Ферма, видимо, их не имеют. Вскоре математикам стало интереснее не столько *решать* диофантовы уравнения, сколько определять, имеет ли вообще частное диофантово уравнение решение в целых числах.

Несуществующее доказательство Ферма стало известно как Великая теорема Ферма (или Большая теорема Ферма), и долгие годы было принято считать: что бы ни *думал* Ферма о своем доказательстве, оно, скорее всего, неверно. Только в 1995 году Теорема Ферма была доказана английским математиком Эндрю Уайлсом (род. в 1953 г.), который интересовался этой проблемой с десяти лет. (Для многих частных случаев, когда, например, показатель степени равен 3, отсутствие решения было установлено много раньше.)

Очевидно, доказательство того, что у некоторого диофантова уравнения *нет* приемлемого решения, гораздо привлекательнее, чем по-

[1] Heath, Diophantus of Alexandria, 144, note 3.

иск решения, когда известно, что оно есть. Если известно, что у частного диофантова уравнения есть решение, можно просто проверить все возможности. Допустимые решения – только целочисленные, поэтому начинаем пробовать 1, затем 2, 3 и т. д. Если же столь утомительная работа вам уже невмоготу, просто напишите компьютерную программу, которая проверит за вас все возможности. Рано или поздно она найдет решение.

Но если неизвестно, что решение существует, компьютерный подход с позиций грубой силы совершенно не годится. Его можно начать, но как узнать, когда его закончить? Как убедиться, что все последующие проверяемые наборы чисел будут совсем не тем, что мы ищем?

Всё проклятие чисел – в их *бесконечности*.

Глава 2

Иррациональные и трансцендентные числа

Начав считать 1, 2, 3, мы можем продолжать настолько долго, насколько захотим. Такие числа известны как *счетные*, *целые*, *кардинальные, натуральные*, и они, конечно, *кажутся* достаточно естественными и интуитивно понятными, потому что вселенная содержит очень много объектов, которые мы можем посчитать. Натуральные числа были, вероятно, первыми математическими объектами, постигнутыми первобытными людьми. У некоторых животных, видимо, тоже есть понимание чисел, пока они не становятся слишком большими.

Нуль веками не входил в натуральный ряд чисел, и даже теперь здесь нет твердого согласия. (Учебники по теории чисел обычно предупреждают на первой странице, включает ли автор нуль в натуральный ряд чисел.) По другую сторону нуля – отрицательные целые числа. Ко всем положительным и отрицательным целым числам, как и к нулю, больше всего подходит слово *целое* (или *целочисленное*). Целые числа уходят в бесконечность в двух противоположных направлениях:

... –3 –2 –1 0 1 2 3 ...

Для обозначения только положительных целых чисел, начинающихся с 1, лучше всего подходит термин *положительные целые*. Для положительных чисел, начинающихся с нуля (то есть 0, 1, 2, 3...), однозначным и не *слишком* многословным будет термин *неотрицательные целые*.

Рациональные числа – это числа, которые могут быть выражены отношением целых чисел (дробью), за исключением знаменателя 0. Например,

$$\frac{3}{5}$$

– рациональное число, обычно записываемое и в десятичной форме:

0,6.

Рациональные числа включают и все целые числа, потому что любое целое число (скажем, 47) может быть записано дробью со знаменателем 1:

$$\frac{47}{1}.$$

Любое число с конечным числом десятичных разрядов – тоже рациональное. Например,

–23,45678

может быть представлено отношением:

$$\frac{-2345678}{100000}.$$

Некоторые рациональные числа, например

$$\frac{1}{3},$$

требуют для представления в десятичной форме бесконечного количества цифр:

0,3333333333...

Это – все еще рациональное число, потому что оно – дробь. Более того, любое число с *повторяющимся набором* цифр где-то после десятичной запятой – это рациональное число. Число

0,234562345623456...

– рациональное, если набор цифр 23456 повторяется бесконечно. Чтобы показать, что оно – рациональное, давайте приравняем его к x:

x = 0,234562345623456...

Теперь умножим обе части равенства на 100 000:

100 000 x = 23 456,23456234562346...

Хорошо известно, что если вычесть одну и ту же величину из обеих частей равенства, то равенство сохранится. Это значит, что можно вычесть из второго равенства первое: вычитаем x из 100 000x и 0,23456... из 23 456,23456..., и десятичная часть исчезает:

99 999x = 23 456.

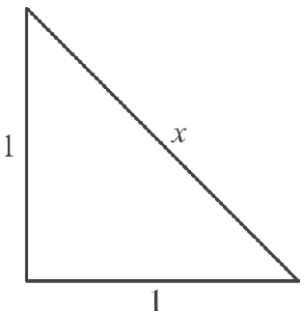
Таким образом:

$$x = \frac{23456}{99999}.$$

Это – отношение, а значит, рациональное число.

Только на первый взгляд кажется, что рациональные числа обладают совершенной полнотой. Если сложить два рациональных числа, получится другое рациональное число. Вычитание, умножение или деление рациональных чисел также дает в результате рациональное число.

Можно было бы предположить (как многие годы считали люди), что все числа – рациональные, но рассмотрим гипотенузу этого простого прямоугольного треугольника:



Согласно Теореме Пифагора,

$x^2 = 1^2 + 1^2$,

или

$x^2 = 2$,

или

$x = \sqrt{2}$.

Существует ли отношение двух целых чисел, которое, будучи умноженным на себя, дает в результате 2? Конечно, можно поискать и найти много рациональных чисел, которые дадут очень близкий к искомому результат. Одно из них:

$$\frac{53492}{37825}.$$

Правда, оно чуть меньше. Умноженное на себя, оно дает примерно 1,99995. Если мы продолжим поиски, то, возможно, найдем точное решение.

Или мы зря тратим свое время?

Трудно доказывать, что чего-то не существует, но математики придумали вид доказательства, которое часто подходит в подобных случаях. Оно называется *косвенным доказательством*, или *доказательством от противного*, на латыни – *reductio ad absurdum* («сведение к нелепости»). Вы начинаете с предположения. Затем из этого предположения вы делаете логические выводы, пока не «упираетесь» в противоречие. Это противоречие означает, что первоначальное предположение было неверным.

Доказательства *reductio ad absurdum* похожи на окольный путь, но они, судя по всему, чаще, чем мы думаем, распространены в обычной жизни. Алиби – разновидность *reductio ad absurdum*. Если ответчик был на месте преступления *и* в доме своей матери, это значит, что он был сразу в двух разных местах одновременно. Абсурд!

Давайте начнем с предположения, что квадратный корень из 2 – рациональное число. А раз так, то существуют такие целые числа a и b, что:

$$\frac{a}{b} = \sqrt{2}.$$

Являются ли a и b оба четными? Если да, разделим их оба на 2 и заменим их половинами. Если и они – все еще четные, делим их тоже на 2 и продолжим так до тех пор, пока или a, или b (или они оба) не станет нечетным.

Возведем обе части уравнения в квадрат:

$$\frac{a^2}{b^2} = 2.$$

Или:

$$a^2 = 2b^2.$$

Заметим, что квадрат a вдвое больше квадрата b. Это означает, что квадрат a четен, а это возможно лишь тогда, когда само число a – четное. Ранее мы установили, что a и b не могут быть оба четными, таким образом, теперь мы знаем, что нечетно число b.

Если a – четное, то оно вдвое больше некоторого целого числа, которое мы обозначим c:

$(2c)^2 = 2b^2$.

Или:

$4c^2 = 2b^2$.

Или:

$2c^2 = b^2$.

Это значит, что квадрат b – четное число, а раз так, то и b – тоже четное, как и a, что противоречит первоначальному предположению о том, что a и b не могут быть оба четными.

Следовательно, исходное предположение о том, что квадратный корень 2 – рациональное число, неверно. Квадратный корень 2 – бесспорно, *иррациональное* число. В десятичной форме его цифры следуют без видимых повторов цепочек цифр:

1,41421356237309504880168872420 97...

Это число невозможно записать точно, не имея бесконечных запасов бумаги, чернил и времени. Можно лишь приближаться к нему, и многоточие – признание нашего поражения. Ближе всего можно подойти к этому числу с помощью алгоритма его вычисления. (Именно это я и сделаю в главе 6.)

Как ни странно, есть причина, почему используемые нами термины – рациональный и иррациональный – выносят приговор здравомыслию чисел. Иногда иррациональные числа называют также *несоизмеримыми* (*surds*), с тем же корнем, что в слове *абсурд* (*absurd*). Иррациональные числа были известны древним грекам, но те им очень не нравились. Согласно преданию (но не достоверной истории), иррациональность квадратного корня из 2 установил в VI веке до н. э. ученик Пифагора Гиппазий. Далее легенда гласит, что это открытие настолько возмутило логичных и рациональных греков, что Пифагор и его последователи попытались погасить возмущение, сбросив Гиппазия в Средиземное море. Им, конечно же, хотелось, чтобы иррациональных чисел не было. Отказываясь в своих задачах от иррациональных чисел, Диофант следовал древней традиции: иррациональные числа были ему совсем не по вкусу.

В десятичной записи, которая есть у нас (но которой не было у древних греков), легко создавать числа, явно иррациональные. Достаточно лишь написать что-нибудь пикантное *без* повторений цифровых шаблонов. Вот, например, число с ненормальным, в некотором

роде, шаблоном из десятичных цифр, который определенно не повторяется:

0,00101101110111110111110111111...

После десятичной запятой идут 00 и 1, потом – 0 и две 1, потом – 0 и три 1 и т. д. Это не рациональное число! Его нельзя представить отношением двух целых чисел. Поэтому оно иррационально.

Квадратный корень из 2 – это решение следующего уравнения:

$x^2 - 2 = 0$.

Это то же уравнение, что было раньше, с разницей лишь в том, что 2 перенесено в левую его часть. Кубический корень из 17 (тоже иррациональный) – это решение следующего уравнения:

$x^3 - 17 = 0$.

Оба эти уравнения называются *алгебраическими уравнениями*. Вот другое алгебраическое уравнение:

$-12x^5 + 27x^4 - 2x^2 + 8x - 4 = 0$.

В алгебраическом уравнении – одна переменная, обычно называемая x. (Алгебраические уравнения – это не диофантовы уравнения, потому что диофантовы уравнения могут иметь много переменных.) В алгебраическом уравнении – ряд членов (в последнем примере – 5), сумма которых равна нулю. Каждый член содержит переменную в степени – целой или нулевой. (Поскольку нечто в нулевой степени равняется 1, пятый член можно понимать как произведение –4 на нулевую степень x.) Каждая возведенная в степень переменная умножается на целочисленный коэффициент, в этом примере –12, 27, –2, 8 и –4. Коэффициенты могут быть нулевыми, что имеет место с «пропавшим» членом x^3.

К алгебраическим уравнениям сводятся многие реальные задачи, поэтому они считаются весьма важными. Общий вид алгебраического уравнения:

$$a_N x^N + a_{N-1} x^{N-1} + ... + a_2 x^2 + a_1 x + a_0 = 0,$$

где N – положительное целое число и a_i – целые числа. Можно записать его короче в виде суммы:

$$\sum_{i=0}^{N} a_i x^i = 0.$$

В приведенном выше уравнении

$$-12x^5 + 27x^4 - 2x^2 + 8x - 4 = 0$$

$N = 5$ (это самый высокий показатель степени, называемый степенью многочлена), $a_5 = -12$, $a_4 = 27$, $a_3 = 0$ и т. д.

Решения алгебраического уравнения (называемые также корнями уравнения) называются алгебраическими числами. Многочлен степени N имеет не более N различных решений. Алгебраическое уравнение из главы 1

$$x^2 - 10x + 21 = 0$$

подходит к этому определению. Оно имеет решения 3 и 7.

Квадратный корень из 2 – одно из решений алгебраического уравнения

$$x^2 - 2 = 0.$$

Отрицательный квадратный корень из 2 – второе его решение.

Алгебраические числа включают в себя все целые и все рациональные числа. Например, целое число 5 является решением алгебраического уравнения

$$x - 5 = 0,$$

а 3/7 – решением уравнения

$$7x - 3 = 0.$$

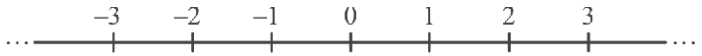
Некоторые алгебраические уравнения имеют своим решением квадратные корни из отрицательных чисел:

$$x^2 + 5 = 0.$$

Это уравнение кажется неразрешимым, потому что любое умноженное на себя число – положительное, и добавление к нему 5 никогда не даст в результате нуль. Квадратные корни из отрицательных чисел называются *мнимыми* (*комплексными*) *числами*. (Квадратному корню из –1 для удобства назначен символ *i*.) Вопреки своему названию мнимые числа очень полезны и имеют практическое применение в реальности, но их не будет ни в статье Тьюринга, ни в этой книге.

Примерно в XVIII веке в противовес мнимым числам математики заговорили о *вещественных* (*действительных*) числах. По определению, вещественные числа включают *все* числа, кроме квадратных корней из отрицательных чисел.

Вещественные числа имеют также отношение к *континууму*, потому что вещественные числа можно представить как все точки непрерывной прямой:



На этой прямой отмечены некоторые целые числа, но очевидно, что сами они не могут образовать непрерывную прямую.

Как не могут этого и рациональные числа. Конечно, рациональные числа заполняют прямую довольно плотно. Между любыми двумя рациональными числами, скажем, a и b, можно вставить другое рациональное число – среднее арифметическое этих двух:

$$\frac{a+b}{2}.$$

Однако между рациональными числами еще существуют промежутки, где обитают иррациональные числа. Например, один из таких промежутков содержит квадратный корень из 2.

Теперь перейдем к теме классификации чисел с двух сторон. С одной стороны, мы определили класс *алгебраических чисел*, которые являются решениями алгебраических уравнений. Этот класс чисел включает целые, рациональные и многие иррациональные числа, такие как квадратные и кубические корни. С другой стороны, мы определили класс *вещественных чисел*, то есть всех чисел, которые не являются квадратными корнями из отрицательных чисел. Теперь сам собой напрашивается следующий вопрос:

Являются ли все вещественные числа также алгебраическими? Или: есть ли такие вещественные числа, которые не являются решениями алгебраических уравнений?

В 1740-х годах Леонард Эйлер (1707–1783) – неутомимый швейцарский математик, чье имя произносится «Ойлер», – предположил, что неалгебраические числа действительно существуют, и назвал их *трансцендентными* (от лат. *transcendentis* – запредельный. – *прим. перев.*) числами, потому что они выходят за пределы алгебраических. Доказать существование трансцендентных чисел было трудно. Как доказать, что конкретное число не является *решением* некоторого довольно длинного и замысловатого алгебраического уравнения?

Вопрос о существовании трансцендентных чисел оставался открытым вплоть до 1844 года, когда французский математик Жозеф Лиу-

вилль (1809–1882) придумал число и смог доказать, что оно – неалгебраическое. Вот первые 30 десятичных разрядов числа Лиувилля:

0,110001000000000000000000001000000...

Правда, этот фрагмент совсем не раскрывает всего числа. Лиувилль строит это «сумасшедшее» число с помощью *факториалов*. Факториал числа – это произведение числа и всех положительных целых чисел, не превышающих его самого, которое обозначается восклицательным знаком:

$1! = 1$
$2! = 1 \times 2 = 2$
$3! = 1 \times 2 \times 3 = 6$
$4! = 1 \times 2 \times 3 \times 4 = 24$
$5! = 1 \times 2 \times 3 \times 4 \times 5 = 120$

и т. д. Число Лиувилля (как его иногда называют) содержит 1 в разрядах с номерами 1, 2, 6, 24, 120 и т. д. Лиувилль создал это число именно для того, чтобы доказать, что оно не является решением какого-либо алгебраического уравнения. Нарастающее расстояние между ненулевыми цифрами числа – ключ к доказательству[1].

В 1882 году немецкий математик Фердинанд Линдеман (1852–1939) доказал, что одно из самых известных иррациональных чисел всех времен – тоже трансцендентное. Это число π – отношение длины окружности к ее диаметру:

π = 3,14159265358979323846264338327 95...

Линдеман показал, что π не является решением алгебраического уравнения, что позволило понять очень старую задачу: больше двух тысячелетий математики, и не только они, бились над «квадратурой круга». Задача ставится просто: дан круг; с помощью линейки и циркуля построить квадрат той же площади, что и круг. (Похожая задача выпрямления окружности требует построения отрезка прямой линии с длиной, равной длине окружности.) Люди так исступленно бились над этой задачей, что древние греки даже окрестили эту навязчивую идею словом τετραγωνίζειν, буквально тетрагония (*tetragonize*)[2].

---

[1] Доказательство обсуждается в Edward B. Burger and Robert Tubbs, *Making Transcendence Transparent: An Intuitive Approach to Classical Transcendental Number Theory* (Springer, 2004), 9–26.

[2] Hobson E. W. *Squaring the Circle: A History of the Problem* (Cambridge University Press, 1913), 3.

Использование линейки и циркуля для построения геометрических фигур равносильно решению алгебраических уравнений определенного вида. Поскольку π не является решением алгебраического уравнения, его невозможно получить с помощью геометрических построений. Добиться квадратуры круга с помощью линейки и циркуля невозможно.

Другое знаменитое трансцендентное число обозначается буквой *e* (в честь Эйлера). Если вычислять это число по формуле

$$\left(1+\frac{1}{N}\right)^N,$$

увеличивая значение *N*, можно получить следующее его приближение:

e = 2,71828182845904523536028747135 27...

Можно также вычислить *e*, суммируя бесконечный ряд, содержащий факториалы:

$$1+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\frac{1}{4!}+\frac{1}{5!}+\ldots$$

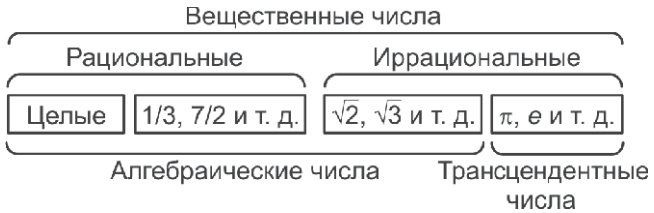
Число *e* можно вычислить, но оно не является решением какого-либо алгебраического уравнения.

На протяжении прошлого века было найдено множество трансцендентных чисел, однако общего способа определения трансцендентности числа не существует. Например, по числу

π^π

суд все еще не состоялся.

Статья Тьюринга (и эта книга) ограничивается вещественными (не мнимыми) числами, и в следующей диаграмме приведены самые важные классы области вещественных чисел:



На этой диаграмме не соблюден масштаб.

Погодите, сейчас объясню, что я имею в виду.

Все эти классы чисел бесконечны. Верно? Есть бесконечное число целых чисел, бесконечное число дробей, бесконечное число иррациональных чисел. Верно? Бесконечность – это бесконечность. Верно? Но бесконечности не могут быть разного *размера,* не так ли? Разве может быть одна бесконечность больше другой бесконечности?

Верно?

Тема бесконечности никогда не была простой, независимо от того, с каких позиций – философских, теологических или математических – ее рассматривали. Однако в математике обойти стороной бесконечность вряд ли возможно. Мы вынуждены исследовать это понятие со всей смелостью, на какую только способны.

Бесконечное стремление натуральных чисел быть все больше и больше, кажется, лежит в самом основании нашего понимания бесконечно большого. Досчитав до некоторого числа, мы всегда можем добавить к нему еще одну единицу. Конечно, вещественные числа тоже могут становиться бесконечно большими, но лишь потому, что следуют по пятам натуральных чисел. Вещественные числа позволяют нам задуматься и о бесконечно малых, когда мы делим континуум на все меньшие и меньшие части.

Похожи ли в чем-то эти две бесконечности – бесконечность никогда не заканчивающихся натуральных чисел и бесконечность плотности континуума? Или же они совсем разные?

Последующее обсуждение будет несколько проще, если мы примем на вооружение некоторые элементы теории множеств. *Множество* – это совокупность объектов, которые называются *элементами* множества. Множество обычно заключается в пару фигурных скобок. Например,

$$\{1, 2, 3, 4\}$$

– множество из первых четырех положительных целых чисел. Элементы множества уникальны. Так, две четверки в одном и том же множестве недопустимы. Порядок следования элементов множества не имеет значения. Множество

$$\{4, 1, 3, 2\}$$

равно предыдущему. Количество элементов множества называется *кардиналом*, или *мощностью*, множества. Мощность конечного множества, приведенного выше, равна 4. Говорят, что множества одинаковой мощности *равносильны* (или равномощны. – *прим. перев.*).

Одни множества имеют конечную мощность, другие – бесконечную. Рассмотрим множество положительных целых чисел:

{1, 2, 3...}.

Его мощность, определенно, бесконечна. То же самое верно для множества положительных *четных* целых чисел:

{2, 4, 6...}.

Каково отношение между мощностями этих двух множеств?

Возможно, наше чутье подсказывает нам, что первое множество содержит *вдвое* больше элементов, чем второе, потому что во втором множестве отсутствуют все нечетные числа.

Конечно, это лишь одно из мнений, и оно было бы верно, если бы два множества были конечными. Но можно ли говорить о том, что одно множество «вдвое больше» другого, если оба они бесконечны?

Давайте попробуем посчитать элементы второго множества. Что на самом деле значит *посчитать*? Это значит установить соответствие между элементами множества и натуральным рядом чисел: водя носом по предметам, мы говорим «номер 1, номер 2, номер 3...».

Мы можем посчитать положительные четные целые числа в бесконечном множестве, соотнося каждому из них натуральное число:

1	2	3	4	5	6	7	8	…
↕	↕	↕	↕	↕	↕	↕	↕	↕
2	4	6	8	10	12	14	16	…

Каждому положительному целому числу соответствует четное число. Каждому четному числу соответствует положительное целое число. При таком взгляде теперь выходит, что два множества имеют одинаковые размеры, что означает их равносильность. Что это – парадокс или нет? (На самом деле эта странная особенность бесконечных множеств была замечена Галилеем еще в 1638 году[1] и иногда называется парадоксом Галилея.)

Казалось, этот парадокс никого и нисколько не беспокоил до тех пор, пока в схватку с ним не вступил Георг Кантор (1845–1918).

---

[1] Galileo Galilei, *Two New Sciences*, second edition, trans. Stillman Drake (Wall & Emerson, 2000), 40–41. Перевод основан на *Opere di Galileo Galilei* (Florence, 1898), VIII, 78–79. Вероятно, Галилей был не первым, кто обратил внимание на этот парадокс: дополнительно см. Stephen Cole Kleene, *Mathematical Logic* (Wiley, 1967; Dover, 2002), с. 176, сноска 121.

Кантор – математик, внесший огромный вклад в основы теории множеств, – родился в Санкт-Петербурге. Его отец был торговцем, который наставлял сына так, чтобы тот превзошел его в том, чем он сам занимался. Мать Кантора была из семьи музыкантов Бём. Кантор проявил свои таланты и в искусстве, и в музыке, но в 17 лет решил «посвятить свою жизнь математике»[1]. Он посещал Политехникум (Polytechnicum) в Цюрихе и университет в Берлине. В 1869 году Кантор получил место преподавателя в университете города Галле, где оставался до конца жизни.

В 1873 году в письме математику Рихарду Дедекинду (1831–1916) Кантор, размышляя о соответствии вроде соответствия между натуральными и четными числами, задавался вопросом об установлении подобного соответствия между натуральными и *вещественными* числами. Он подозревал, что его нет, но не знал, почему. «Я не могу найти объяснения, которого ищу; возможно, оно очень простое», – писал Кантор[2]. Знаменитые последние слова.

Говорят, что множество, чьи элементы образуют паросочетания с натуральными числами, – *перечислимое* (или иногда – *нумерованное* или *счетное*). Множество перечислимо, если его элементы можно некоторым образом упорядочить или перечислить списком, а всякий список может быть пронумерован, то есть сопоставлен натуральному ряду 1, 2, 3 и т. д. Все конечные множества – безусловно, перечислимые. Настоящая проблема – в бесконечных множествах.

Рассмотрим, например, целые числа, включающие отрицательные, положительные и нуль. Перечислимо ли такое множество? Да, потому что мы можем перечислить все целые числа, начиная с нуля:

0
1
–1
2
–2
3
–3
. . .

---

[1] Joseph Warren Dauben, *Georg Cantor: His Mathematics and Philosophy of the Infinite* (Harvard University Press, 1979; Princeton University Press, 1990), 277.

[2] Georg Cantor, letter of November 29, 1873, in *From Kant to Hilbert: A Source Book in the Foundations of Mathematics* (Oxford University Press, 1996), Vol. II, 844.

Обычно целые числа перечисляются не так, но этот пример ясно показывает, что единым списком охвачены все целые числа.

Довольно интересно, что рациональные числа *тоже* перечислимы. Давайте начнем с положительных рациональных чисел и не будем беспокоиться о том, что в списке есть повторы:

1/1
1/2
2/1
1/3
2/2
3/1
1/4
2/3
3/2
4/1
...

Видите закономерность? У первого элемента списка сумма числителя и знаменателя равна 2. У следующих двух элементов списка суммы числителя и знаменателя дают 3. Следующие три элемента списка имеют числители и знаменатели, которые в сумме дают 4. И так далее. Продолжив подобным образом дальше, получим список всех положительных рациональных чисел. Отрицательные рациональные числа можно включить в список, чередуя их с положительными. Следовательно, рациональные числа перечислимы.

В опубликованной в 1874 году статье «О свойстве множества вещественных алгебраических чисел»[1] Кантор показал, что даже алгебраические числа перечислимы. Если помните, алгебраические числа – это решения алгебраических уравнений общего вида

$$a_N x^N + a_{N-1} x^{N-1} + ... + a_2 x^2 + a_1 x + a_0 = 0,$$

где N – положительное целое число и a_i – целые числа. Для любого частного алгебраического уравнения давайте сложим все коэффициенты (значения a_i) и само N. Назовем это число *высотой* уравнения. Для заданной высоты (например, 5) существует конечное число алгебраических уравнений. Каждое уравнение имеет не более N решений. Таким образом, все алгебраические числа могут быть перечислены в порядке их высот и решений. Следовательно, алгебраические числа перечислимы.

---

[1] Более доступно изложено в *From Kant to Hilbert*, Vol. II, 839–843.

А трансцендентные числа? Можно ли их как-то посчитать? Вряд ли! Ведь нет даже общей процедуры определения трансцендентности некоторого числа!

А вещественные числа, которые включают в себя алгебраические и трансцендентные числа? Их можно посчитать?

В той же статье 1874 года, где Кантор показал перечислимость алгебраических чисел, он показал также, что вещественные числа *не* перечислимы.

Кантор начал свое доказательство с предположения, что вещественные числа *перечислимы*. Он предположил, что есть некий способ перечислить вещественные числа и что их перечислили в следующем списке, элементы которого обозначены как ω с индексом:

$\omega_1 \ \omega_2 \ \omega_3 \ \omega_4 \ \omega_5 \ \omega_6 \ldots$

Кантор собирается показать, что этот список неполон, что независимо от того, как он был построен, он просто не может содержать все вещественные числа.

Выберем любое число α (альфа) и большее число β (бета). Их можно представить на числовой оси таким образом:

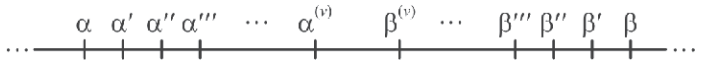
Теперь начнем двигаться по нашему перечислимому списку вещественных чисел, пока не найдем первые два вещественных числа, попадающих между α и β. Эти два числа больше α и меньше β. Назовем меньшее из них α′, а большее – β′:

Продолжаем двигаться по списку вещественных чисел дальше, пока не найдем два новых числа между α′ и β′. Назовем их α″ и β″.

И еще раз:

Очевидно, что этот процесс должен продолжаться бесконечно. У нас всегда будет возможность найти еще два числа между последними двумя числами.

Как это узнать? Легко! Предположим, мы находимся в таком положении:



где верхний индекс (v) *задает* количество v штрихов (миллион, миллиард, триллион или еще больше), но обязательно конечное. Теперь, независимо от того, как долго продолжается поиск по списку перечислимых вещественных чисел, невозможно найти еще одну пару чисел, которая попадает между $\alpha^{(v)}$ и $\beta^{(v)}$. Тогда очевидно, что список действительных чисел неполон. В списке пропущены все числа между $\alpha^{(v)}$ и $\beta^{(v)}$. Например, число посередине между $\alpha^{(v)}$ и $\beta^{(v)}$ – это их среднее арифметическое, или:

$$\frac{\alpha^{(v)}+\beta^{(v)}}{2}.$$

И это – только начало. В списке пропущено много других чисел.

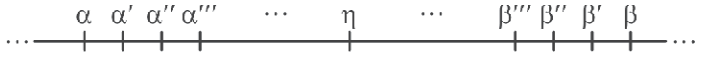
И этот процесс, как известно, может продолжаться бесконечно. Значения α продолжают расти, а значения β продолжают уменьшаться, но наибольшее α не станет больше наименьшего β. (Найдя новую пару чисел, попадающих внутрь последней пары α и β, меньшее – всегда α, а большее – всегда β.) Как α, так и β имеют границу – *предел*, – который Кантор обозначает символом бесконечности в верхнем индексе: α^{∞} и β^{∞}.

Возможно ли, чтобы α^{∞} было меньше β^{∞}? Обратите внимание:

Нет, это невозможно. Если α никогда не становятся больше α^{∞}, а β никогда не становятся меньше β^{∞}, то в списке вещественных чисел пропущены все числа между α^{∞} и β^{∞}, например:

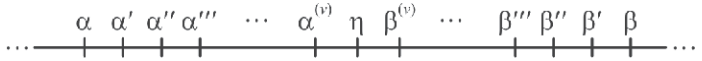
$$\frac{\alpha^{\infty}+\beta^{\infty}}{2}.$$

В конце концов, должно случиться так, что α^{∞} станет равным β^{∞}. Кантор называет этот предел η (греческая буква «эта»):



Поскольку данный процесс бесконечен (мы уже установили, что он не может остановиться ни в одной так ли точке), α, как и β тоже, никогда не достигают η. Теперь мы знаем, что это означает, не? Это означает, что значения η нет в первоначальном списке вещественных чисел!

Если бы значение η *было* в списке, оно появилось бы когда-нибудь при поиске следующих α и β, но рассмотрим α и β, которые появились списке прямо перед η:



Теперь в списке вещественных чисел между $\alpha^{(v)}$ и $\beta^{(v)}$ пропущены все числа, кроме η.

Все варианты исчерпаны. Ни один не работает, ни один не имеет смысла, и все это – из-за ошибочности исходного предположения о том, что вещественные числа можно пронумеровать. Стало быть, мы не можем этого сделать.

Целые числа перечислимы. Рациональные числа перечислимы. Даже алгебраические числа перечислимы. А вот вещественные – нет.

Кантор полагал, что неперечислимость вещественных чисел – новое доказательство существования трансцендентных чисел. (Если трансцендентных чисел нет, то вещественные числа – это то же, что алгебраические, и, следовательно, они перечислимы.) В итоге Кантор понял то, что существует по крайней мере два вида бесконечности – перечислимая и неперечислимая, бесконечность натуральных чисел и бесконечность континуума. Бесконечные множества натуральных, рациональных и даже алгебраических чисел – перечислимые. Как только мы добавляем трансцендентность, мы вдруг оказываемся в совершенно ином мире. Мы обнаруживаем разные мощности двух разных бесконечностей: одна мощность относится к натуральным, рациональным и алгебраическим числам; другая мощность – это мощность вещественных чисел и континуума.

Работа Кантора была спорной в своё время и не вполне свободна от этого и теперь. Но после Кантора не было математика, который не думал бы о бесконечности почти так же. Кроме того, разделение бесконечности на перечислимую и неперечислимую оказалось чрезвычайно полезным в условиях, когда воображение *одной* лишь бесконечности простого типа еще пугало человеческий разум.

Известен миф о том, что длительным созерцанием бесконечности Кантор довел себя до безумия. Действительно, последние примерно

20 лет своей жизни Кантор проводил в психиатрических больницах, но это была, скорее всего, разновидность депрессии, которая проявлялась вне зависимости от рода его занятий[1]. И все же обострения у Кантора приступов психической болезни, видимо, были вызваны усталостью и напряжением, а напряжение, наверное, было обусловлено неприятием его необычной математической теории. В периоды выздоровления Кантор проявлял отличные от математических интересы. Он занимался философией, богословием, метафизикой и гипотезой о том, что пьесы, приписываемые Уильяму Шекспиру, писал Фрэнсис Бэкон.

У конечных и бесконечных множеств довольно много различий. Одно серьезное различие – в *собственных подмножествах*, то есть подмножествах, не совпадающих с самим множеством. Мощность собственного подмножества конечного множества всегда меньше мощности самого множества. Это более чем очевидно. Мощность собственного подмножества бесконечного множества тоже может быть меньше мощности самого множества. (Например, множество натуральных чисел – это собственное подмножество множества вещественных чисел, их мощности различны.) Однако в некоторых случаях мощность собственного подмножества бесконечного множества *равна* мощности самого множества. Это возможно только для бесконечных множеств. Множество натуральных чисел – собственное подмножество множества целых чисел, которое является собственным подмножеством множества рациональных чисел, которое является собственным подмножеством множества алгебраических чисел. У всех этих бесконечных множеств *одинаковая* мощность. Они равносильны.

Также верно и то, что различные собственные подмножества вещественных чисел равносильны друг другу. Рассмотрим множество вещественных чисел от 0 до 1. Можно установить взаимно-однозначное соответствие между ним и множеством вещественных чисел, больших 1. Стоит только поделить 1 на каждое число из множества. Например, 0,5 соответствует 2; 0,25 – 4; 0,1 – 10, а 0,0001 – 10 000. Этот простой факт оказывается очень полезным: он означает, что можно изучать свойства вещественных чисел, ограничившись диапазоном от 0 до 1, и всё, что мы обнаружим, будет применимо ко всем вещественным числам. (Тьюринг, как и Кантор, тоже пользуется этим свойством в своей статье.)

---

[1] Dauben, *Georg Cantor*, 285.

Изучая бесконечные множества, Кантор сделал и другие удивительные открытия: он обнаружил, что можно установить взаимнооднозначное соответствие между континуумом – вещественными числами на прямой – и точками двумерной плоскости и даже точками любого *N*-мерного пространства.

Давайте ограничимся, например, участком плоскости с координатами x и y, заключенными между 0 и 1. Каждая точка на плоскости может быть представлена парой чисел (x, y), и каждое из этих двух чисел может иметь бесконечное число цифр после десятичной запятой. В следующем выражении каждая цифра числа x после десятичной запятой обозначается буквой a с индексом:

$x = 0, a_1 a_2 a_3 a_4 ...$

То же для y:

$y = 0, b_1 b_2 b_3 b_4 ...$

Теперь возьмем эти цепочки цифр и «сплетём» из них новое число:

$0, a_1 b_1 a_2 b_2 a_3 b_3 a_4 b_4 ...$

Получилось одно вещественное число, заключающее в себе два вещественных числа. Каждая двумерная точка соответствует одному вещественному числу континуума. Следовательно, множество точек плоскости обладает той же мощностью, что и множество вещественных чисел прямой. Кантор был так поражен этим открытием, что изменил своему немецкому. «*Je le vois, mais je ne le crois pa*», – писал он Дедекинду[1]. «Я вижу это, но я этому не верю».

В 1891 году Кантор опубликовал другое доказательство неперечислимости вещественных чисел[2], и с тех пор оно так и осталось в умах людей. Доказательство Кантора имело дело не с числами, а с множествами и было более общим, чем пример, который я собираюсь привести, но идея та же. По причинам, которые станут вполне понятными, оно получило название *диагонального доказательства*, или *диагонального процесса*, или *диагонального аргумента*, или *диагонализации*. Но как бы оно ни называлось, в его название входит слово *диагональ*.

Давайте ограничимся вещественными числами от 0 до 1. Предположим, что мы изобрели некий способ перечисления всех веществен-

---

[1] Письмо от 29 июня 1877 года в *From Kant to Hilbert*, Vol. II, 860.

[2] Georg Cantor, «On an Elementary Question in the Theory of Manifolds», *From Kant to Hilbert*, Vol. II, 920–922.

ных чисел. (Как можно догадаться, что это еще одно доказательство *от противного*.) Допустим, список начинается как-то так:

0,1234567890...
0,2500000000...
0,3333333333...
0,3141592653...
0,0010110111...
0,4857290283...
0,0000000000...
0,9999999999...
0,7788778812...
0,2718281828...
...

Кажется, мы избежали хорошего начала. Список включает 0, 1/4, 1/3, π/10, *e*/10, то странное иррациональное число с меняющимся числом единиц, которое я приводил раньше, и некоторые другие, не вполне узнаваемые числа. Каждое число в списке имеет бесконечное количество десятичных разрядов (даже если они только нулевые), и в нем – бесконечное количество чисел.

Хотя этот список бесконечен, мы можем убедиться, что в нем коечего нет. Посмотрим на цифры, образующие диагональ списка от левого верхнего угла к нижнему правому. Эти цифры выделены жирным шрифтом:

0,**1**234567890...
0,2**5**00000000...
0,33**3**3333333...
0,314**1**592653...
0,0010**1**10111...
0,48572**9**0283...
0,0000000**0**00...
0,9999999**9**99...
0,778877881**2**...
0,27182818**2**8...
...

Теперь составим число из этих выделенных цифр:

0,**1531190918**...

Поскольку список вещественных чисел бесконечен и количество цифр в каждом из них тоже бесконечно, у этого числа – бесконечное

число цифр. Теперь увеличим каждую отдельную цифру этого числа на 1 (цифра 9 меняется на 0):

0,**26422010259**...

Есть ли это новое число в исходном списке? Давайте действовать последовательно: является ли это новое число первым в списке? Нет, это не так, потому что первая цифра первого числа в списке – 1, а первая цифра нового числа – 2.

Является ли оно вторым числом в списке? И снова нет, потому что вторая цифра второго числа в списке – 5, а вторая цифра нового числа – 6.

Является ли оно третьим числом в списке? Нет, потому что третья цифра третьего числа в списке – 3, а третья цифра нового числа – 4.

И так далее. Новое число не может быть *N*-ым числом в списке, потому что *N*-ая цифра *N*-го числа в списке не равна *N*-ой цифре нового числа.

Таким образом, список неполон, и наше исходное предположение неверно. Невозможно перечислить вещественные числа от 0 до 1. Мы еще раз убедились, что вещественные числа неперечислимы.

Что будет, если провести такой же эксперимент со списком алгебраических чисел? Мы уже знаем, как пронумеровать алгебраические числа, так что это не проблема. Построив диагональ и изменив все ее цифры, мы получим число не из списка. Это значит, что полученное число не алгебраическое. Оно – трансцендентное.

По-разному упорядочивая список алгебраических чисел и устанавливая разные правила выделения в нем уникальной диагонали, мы каждый раз будем получать еще одно трансцендентное число.

Для обозначения мощности множества натуральных чисел (и, следовательно, всякого перечислимого бесконечного множества) Кантор в 1895 году выбрал первую букву еврейского алфавита с нулевым нижним индексом – $\aleph_0$ (произносится как «алеф нуль»). Кантор назвал его первым *трансфинитным* числом. Он объединил его с другими трансфинитными числами ($\aleph_1$, $\aleph_2$, $\aleph_3$ и т. д.), чтобы создать целостную трансфинитную математику.

Если мощность перечислимых множеств – $\aleph_0$, то какова мощность неперечислимого множества вещественных чисел? Можно ли хотя бы представить такую мощность?

Можно. Давайте начнем с примера конечных множеств. Вот множество всего из трех элементов:

$\{a, b, c\}$.

Сколько всего подмножеств у этого множества? (Множество всех подмножеств множества называется *степенью множества*.) Попробуем сделать это вручную, не забывая только о пустом множестве и самом множестве из трех элементов:

{} {*a*, *b*}
{*a*} {*a*, *c*}
{*b*} {*b*, *c*}
{*c*} {*a*, *b*, *c*}

У множества из трех элементов – восемь подмножеств, и не случайно:

$2^3 = 8$.

Показатель степени – число элементов исходного множества. Результат – число подмножеств этого множества. Множество из 4 элементов имеет 16, или 2^4, подмножеств. У множества из 5 элементов – 32 подмножества.

Существует более систематический способ подсчета этих подмножеств, который лучше раскрывает данное соотношение. Давайте составим таблицу, столбцы которой – это 3 элемента исходного множества, а цифры 0 и 1 в столбцах – признак вхождения этого элемента в подмножество:

a	*b*	*c*	**Подмножество**
0	0	0	{}
0	0	1	{*c*}
0	1	0	{*b*}
0	1	1	{*b*, *c*}
1	0	0	{*a*}
1	0	1	{*a*, *c*}
1	1	0	{*a*, *b*}
1	1	1	{*a*, *b*, *c*}

Наборы из 0 и 1 в трех первых столбцах – это те же двоичные числа от 0 до 7. Три бита порождают 8 двоичных чисел, а общее правило:

Мощность степени множества = $2^{\text{мощность исходного множества}}$.

Степень множества из 10 элементов состоит из 1024 элементов. Степень множества из 100 элементов состоит из 1 267 650 600 228 22 9 401 496 703 205 376 элементов.

Теперь рассмотрим натуральный ряд, включающий для этой цели также 0:

{0, 1, 2, 3, 4, 5,...}.

Мощность этого множества – $\aleph_0$. Сколько у него подмножеств? Иными словами, какова мощность его степени? По аналогии она должна быть

$2^{\aleph_0}$.

Наверное, нужны дополнительные разъяснения. Давайте построим таблицу, как для конечного множества (но, конечно, не такую полную). Над столбцами – все элементы множества натуральных чисел. В столбце – признак (0 или 1) вхождения натурального числа в конкретное подмножество, указанное справа:

0	**1**	**2**	**3**	**4**	**5**	**…**	**Подмножество**
0	0	0	0	0	0	...	{}
1	0	0	0	0	0	...	{0}
0	1	0	0	0	0	...	{1}
1	1	0	0	0	0	...	{0, 1}
0	0	1	0	0	0	...	{2}
1	0	1	0	0	0	...	{0, 2}
0	1	1	0	0	0	...	{1, 2}
1	1	1	0	0	0	...	{0, 1, 2}
...	...	...	...	...	...	...	...

То, чего мы, в сущности, добиваемся, – это список всевозможных бесконечных комбинаций из 0 и 1. Давайте поместим десятичную запятую (и нуль. – *прим. перев.*) перед каждой цепочкой цифр списка:

0,000000...
0,100000...
0,010000...
0,110000...
0,001000...
0,101000...
0,011000...
0,111000...
...

Тогда все они – двоичные числа в диапазоне от 0 до 1, причем (судя по способу их создания) *все* двоичные числа от 0 до 1 и, следовательно, все вещественные числа от 0 до 1[1]. Ранее было показано, что вещественные числа из диапазона от 0 до 1 могут быть поставлены в соответствие всем вещественным числам, что означает, что вещественные числа могут быть поставлены в соответствие элементам степени множества натуральных чисел. Поэтому эта степень множества имеет ту же мощность, что континуум.

Таким образом, мощность континуума

$$2^{\aleph_0},$$

где $\aleph_0$ – мощность множества натуральных чисел.

Кантор доказал, что невозможно установить взаимно-однозначное соответствие между элементами любого непустого множества и элементами его степени, что очевидно для конечных множеств, но не так очевидно для бесконечных. Теперь этот факт известен как теорема Кантора, и он был главным результатом его статьи 1891 года, в которой предложен метод диагонализации. Как и то, что множество может иметь степень множества, а степень множества – свою собственную степень множества и т. д. Все эти множества имеют разные мощности.

Кантор выяснил, что мощность континуума – это следующее за $\aleph_0$ трансфинитное число, которое он назвал $\aleph_1$. Это предположение называют *континуум-гипотезой* Кантора, и оно может быть выражено математически так:

$$\aleph_1 = 2^{\aleph_0}.$$

Кантор изо всех сил пытался доказать свою гипотезу, но так никогда и не смог этого сделать. Проблема заключалась в том, что между $\aleph_0$ и мощностью континуума могло быть какое-то другое трансфинитное число[2].

---

[1] Может также показаться, что мы наткнулись на метод перечисления всех вещественных чисел от 0 до 1. Закономерность уже понятна – первая цифра после запятой меняется с 0 на 1, вторая цифра меняется со скоростью, вдвое меньшей, и т. д. – и мы могли бы легко продолжать этот список настолько долго, насколько пожелаем. Однако ошибка в том, что список никогда не будет содержать трансцендентное число. Каждое число в списке имеет конечное число ненулевых цифр после запятой.

[2] Существование множества этой промежуточной мощности – это так называемая континуум-гипотеза – первая из 23 проблем Гильберта, о которых он доложил на II Математическом конгрессе в 1900 году. В 1940 году Гёдель доказал, что отрицание континуум-гипотезы недоказуемо, а в 1963 году Коэн доказал недоказуемость континуум-гипотезы. – *прим. перев.*

Несмотря ни на что, глубокий смысл всего этого состоит в том, что мощность перечислимых множеств не просто меньше мощности континуума

$$\aleph_0 < 2^{\aleph_0},$$

а много, много, много, много, много меньше:

$$\aleph_0 << 2^{\aleph_0}.$$

Единственное различие между континуумом и перечислимыми множествами – наличие трансцендентных чисел. Мы вынуждены заключить, что трансцендентные числа (существование которых даже не было доказано до 1844 года) действительно составляют подавляющее большинство всех возможных чисел – более того, *практически все* возможные числа.

На протяжении многих тысячелетий наши представления о числах были совершенно искажены. Как люди мы ценим опрятность, порядок, стандарты и живем в мире соглашений и условностей. В числах нам интересно только то, что имеет для нас смысл. Подсчитывая животных на ферме, мы придумали натуральные числа. Проводя измерения, мы придумали рациональные числа, а высшая математика дала нам алгебраические числа. Мы зачерпнули все эти числа из континуума, не обратив внимания на бездонные глубины, в которых они плавают, как микроскопические бактерии в океане. Мы тешим себя иллюзией, что рациональные числа преобладают над иррациональными, а алгебраических чисел значительно больше, чем трансцендентных, в придуманной нами жизни, конечно. Хотя на самом деле в мире континуума каждое число – скорее трансцендентное.

Что же собой представляют все эти трансцендентные числа? Большинство из них – всего лишь последовательности случайных цифр без всякого порядка, смысла или значения. Действительно, любая последовательность случайных цифр – почти наверно трансцендентная.

Метните стрелу в мишень. Теперь, используя максимально возможный увеличитель и точнейшую линейку, измерьте расстояние от стрелы до центра мишени. Сначала измерьте целое число сантиметров, затем целое число миллиметров, затем целое число микронов и т. д. Вероятность того, что расстояние будет в точности равно рациональному числу, скажем 1,437 сантиметра, ничтожна.

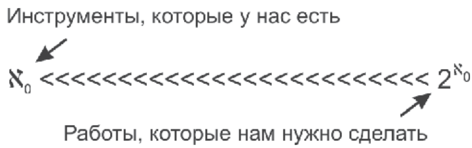
Конечно, на каком-то шаге измерений мы неизбежно придем к тому, что имеем дело с реальностью. Не похоже, чтобы стрела расщепила атом! Нет, стрела воткнется между молекулами мишени, и как только

наш увеличитель проникнет в царство этих молекул, мы увидим, что для проведения точных измерений они слишком сильно вибрируют и возникают заметные искажения из-за конечной длины световой волны, и в какой-то момент встанет ребром принцип неопределенности Гейзенберга, и тогда мы вообще больше ни в чем не можем быть уверены.

При таком увеличении вся идея «континуума» оказывается безнадежно смешной, и мы можем даже испытать желание перевести взгляд от молекул к вселенной в целом и задуматься о том, существует ли вообще бесконечность в реальном мире – особенно с учетом того, что Большой взрыв, вполне вероятно, выбросил лишь *конечное* количество материи и энергии в *конечный* момент времени в прошлом, чтобы создать вселенную, скорее дискретную, чем непрерывную.

Мы могли бы также задаться вопросом: исследование Кантора о перечислимых и неперечислимых множествах – лишь некая очень абстрактная (и пока несколько сомнительная) область теоретической математики или же от этого занятия действительно есть какая-то польза.

Хотя в реальности бесконечность найти трудно, тем не менее в ее математической идее есть огромная польза. Оказывается, что некоторые математические доказательства, которые имеют реальные применения, включая изложенные в статье Тьюринга, основаны на различии между перечислимыми и неперечислимыми множествами, как показано на этом рисунке:



Видите проблему?

Глава 3

Столетия прогресса

Чем меньше секунд оставалось до полуночи в пятницу 31 декабря 1999 года, тем большим беспокойством и страхом наполнялись предстоящие новогодние праздники. Существовала вероятность, если не неизбежность, того, что с боем часов по Всемирной компьютерной сети прокатится волна серьезных технологических сбоев и аварий. Этот кризис стал бы не актом глобального терроризма, а результатом простого небольшого сокращения, которое использовалось программистами в течение почти полувека. В программах, написанных для большого числа разнообразных приложений во множестве различных систем, программисты ради экономии бесценной компьютерной памяти представляли год лишь последними двумя его цифрами, например 75 вместо 1975. В полночь этот двузначный год сменился бы с 99 на 00, не увеличившись, а внезапно став намного меньше. То, что когда-то было невинным сокращением, стало коварной ошибкой, обозначенной высокотехнологичной мнемоникой Y2K.

Конечно, сами программисты десятки лет знали о надвигающейся угрозе. Тревожные предупреждения широкой публике начались примерно в 1998 году с таких книг, как *Y2K: День, когда рухнет мир* (Y2K*: The Day the World Shut Down*); *У черты Y2K* (*Deadline Y2K*); *Y2K: Уже слишком поздно* (*Y2K: It's Already Too Late*); *Y2K: План защиты себя, вашей семьи, ваших средств и вашего общества 1 января 2000* (*Y2K: An Action Plan to Protect Yourself, Your Family, Your Assets, and Your Community on January 1, 2000*); *101 способ пережить кризис Y2K* (*101 Ways to Survive the Y2K Crisis*); *Y2K для женщин: Как защитить свой дом и семью во время грядущего кризиса* (*Y2K for Women: How to Protect Your Home and Family in the Coming Crisis*); *Кризис, вкладывающий капитал в 2000-й год: Как получить выгоду от грядущего компьютерного краха Y2K* (*Crisis Investing for the Year 2000: How to Profit from the Coming Y2K Computer Crash*); *Y2K: разумный ответ на массовую истерию* (*Y2K: A Reasoned Response to Mass Hysteria*); *Духовное выживание во время кризиса Y2K* (*Spiritual Survival During the Y2K Crisis*);

Y2K: Ошибка тысячелетия – взвешенный христианский ответ (*Y2K: The Millennium Bug – A Balanced Christian Response*); *Пробуждение: Верх Y2K* (*Awakening: The Upside of Y2K*) и для детей *Y2K-9: Пес, который сторожит мир* (*Y2K-9: The DogWho Saved theWorld*). Вскоре к ним присоединились телевизионные новостные каналы и журналы. Апрельский номер журнала *Wired* за 1999 год поместил на зловеще черной обложке крупный текст «Свет погас» («Lights Out») с подзаголовком «Учитесь любить Y2K»[1].

Сообщалось, что компьютеры используются почти во всех наших электронных технологиях, а сценарии бедствия рисовались от массового отключения электроэнергии и ограничения подачи воды до падающих с неба самолетов, автомобильных аварий, неуправляемых микроволновых печей и устройств видеозаписи.

XX век был эрой колоссального научно-технического прогресса, и теперь новые технологии приготовились лишить нас большой доли нашего благодушия.

Никогда еще предыдущие смены веков не сопровождались такими страхами. XIX век тоже был эрой громадного научно-технического прогресса, но в ту полночь ничто не предвещало взрыва, и новое столетие было встречено с оптимизмом. Ученые были на пороге всеобщего знания. Такие видные физики, как лорд Кельвин (1824–1907), предсказывали неминуемое решение последних из немногих оставшихся загадок вселенной, включая природу эфира, который пронизывал весь космос и обеспечивал среду для распространения света и других электромагнитных излучений.

В математике – наверное, самой близкой к информатике – дисциплине XIX века, – тоже были достигнуты большие успехи и ожидались еще большие. Потенциальный кризис был изящно преодолен, и математика казалась сильнее, чем когда-либо.

Потенциальный кризис математики в XIX веке коснулся той ее области, которая брала отсчет примерно с 300 года до н. э., – геометрии Евклида. (Хотя имя Эйлера (Euler) произносится как «ой-лер», имя Евклида (Euclid) произносится как «ев-клид».)

Начала Евклида начинаются с ряда определений, за которыми следуют пять *постулатов* (или *аксиом*) и некоторые общие понятия. Из этих нескольких опорных положений Евклид выводит сотни теорем.

---

[1] *Wired*, Volume 7, Issue 4 (April, 1999), архив на www.wired.com/wired/archive/7.04.

Первые четыре постулата Евклида настолько очевидны, что едва ли стоит их разжевывать. В переводе сэра Томаса Хита первые три устанавливают, что с помощью линейки и циркуля можно проводить линии и окружности, да и четвертая так же проста:

1. Провести прямую линию от одной точки до другой точки.
2. Провести непрерывный отрезок на прямой.
3. Описать окружность с любым центром и радиусом.
4. Все прямые углы равны между собой.

По сравнению с краткостью и самоочевидностью первых четырех аксиом, пятая довольно длинна и неуклюжа:

5. Если при пересечении двух прямых третьей сумма внутренних прилежащих углов меньше двух прямых углов, то эти две прямые, если продолжать их бесконечно, пересекутся с той стороны, где сумма углов меньше двух прямых углов[1].

Этот постулат определяет условия, при которых прямые линии не параллельны.

Начиная с самых ранних комментариев к *Началам* Евклида, пятый постулат вызывал споры. Некоторые математики считали, что пятый постулат – лишний, избыточный и на самом деле может быть выведен из первых четырех. Но все попытки вывести пятый постулат проваливались. Успех приходил лишь тогда, когда делались неявные предположения, которые были равносильны пятому постулату.

В начале XIX века некоторые математики начали применять иной подход, предполагая нечто *противоречащее* пятому постулату. Например, что две прямые всегда пересекаются независимо от углов, которые они образуют с третьей прямой. Или что две прямые никогда не пересекаются. Если бы пятый постулат Евклида был и вправду лишним, то это противоречие всплыло бы где-то по пути, и пятый постулат был бы доказан *методом от противного*.

Таких попыток было немного. Карл Фридрих Гаусс (1777–1855) в Германии, Йохан Больяи (1802–1860) в Венгрии и Николай Иванович Лобачевский (1792–1856) в России независимо друг от друга обнаружили, что альтернатива пятому постулату Евклида приводит не к противоречию, а, наоборот, к созданию необычных – но в целом непротиворечивых – геометрических пространств.

---

[1] Thomas L. Heath, *The Thirteen Books of Euclid's Elements*, second edition (Cambridge University Press, 1926; Dover Publications, 1956), Vol. 1, 154–155.

Не столь искушенные древние математики, наверное, с отвращением отвергли бы эти неевклидовы геометрии или впали в отчаяние от того, что элементарная геометрия изуродована этими абсурдными построениями. Но математики приняли такую альтернативу Евклиду и получили хороший урок о природе математики.

Включив сомнительный пятый постулат в свои опорные положения, Евклид оказался прав. Этот постулат был нужен для того, чтобы зафиксировать геометрию Евклида как геометрию на плоскости, но данный постулат не был единственно возможным. При замене его другим получалась геометрия, столь же правильная, как евклидова, и столь же (если не более) интересная. Имеют ли эти неевклидовы геометрии хоть какое-то отношение к тому, что мы так беспечно называем «реальностью»? Порой да. Одна из неевклидовых геометрий определяется на сферической поверхности, и в некотором смысле сфера более «реальна», чем плоскость.

Кроме того, математики XIX века обновили и углубили понятие *аксиоматического метода*, который Евклид использовал в своих *Началах*. (Хотя Евклид и Аристотель различали постулаты и аксиомы[1], в настоящее время эти различия почти совсем исчезли.) Математическая система начинается с отдельных аксиом и продолжается доказательствами следствий из этих аксиом. В зависимости от нашего настроения эти аксиомы могут совпасть или не совпасть с нашим интуитивным пониманием реальности. В течение двух тысячелетий стремление подражать реальности фактически закрепостило геометрию. Если бы аксиомы освободились от реальности и стали достаточно абстрактными, сама математика тоже освободилась бы для исследования новых перспектив. Аксиомы и, в итоге, математика должны трактоваться абстрактно без каких-либо неявных предположений.

Или, как однажды высказался один молодой преподаватель математики, «нужно быть всегда в состоянии сказать: вместо точек, прямых и плоскостей – столы, стулья и пивные кружки»[2].

Этим молодым преподавателем математики был Давид Гильберт (1862–1943), позже ставший на этой стезе одним из выдающихся математиков своего времени. Гильберт родился близ Кёнигсберга, портового города на Балтийском море, а в то время – столицы Вос-

[1] См. Thomas Heath, *Mathematics in Aristotle* (Oxford University Press, 1949; Thoemmes Press, 1998), 50–57 или Howard Eves, *Foundations and Fundamental Concepts of Mathematics*, 3rd edition (PWS-Kent, 1990; Dover, 1997), 29–32.

[2] Constance Reid, *Hilbert* (Springer-Verlag, 1970, 1996), 57.

точной Пруссии. К моменту рождения Гильберта Кёнигсберг уже был известным математическим центром. Это был город, где семь мостов, перекинутых через реку Преголю, нашли свое место в топологической задаче, решенной Леонардом Эйлером.

Кроме того, Кёнигсберг был родиной Кёнигсбергского университета, где учился и преподавал философ Иммануил Кант (1724–1804). Гильберт тоже посещал этот университет и недолго преподавал в нем, но в 1895 году появилась вакансия в университете Гёттингена, и Гильберт занял ее. Гильберт впервые посетил Гёттинген девятью годами ранее «и был очарован небольшим городком и симпатичной холмистой сельской местностью, столь отличавшимися от шумного Кёнигсберга с ровными лугами вокруг»[1].

Университет Гёттингена был известен еще до появления в нем Гильберта. Там в 1833 году Гаусс и физик Вильгельм Вебер (1804–1891) вместе работали над электромагнитным телеграфом. Гёттинген с отделением математики во главе с Гильбертом и Феликсом Клейном (1849–1925) готов был стать Меккой для математиков всего мира.

В ранние годы Гильберт сделал себе имя, занимаясь нерешенными задачами из области алгебраических инвариантов и числовых полей, но в 1898–1899 учебных годах его интересы приняли необычный оборот. Гильберт вел курс геометрии – предмета, который обычно не преподают в университетах, – для студентов, которые уже получили полную дозу Евклида при начальном образовании.

Геометрия Гильберта строилась подобно евклидовой – начиналась с аксиом (точнее с нескольких *групп* аксиом), из которых выводилось множество теорем, – но уровень строгости был непревзойденным. Гильберт полностью переосмыслил геометрию и заново аксиоматизировал ее. Это был современный Евклид со знанием всех неевклидовых геометрий, введенных в его концепцию. В 1899 году Гильберт издал свои лекции по геометрии в книге *Grundlagen der Geometrie* (*Основания геометрии*), которые мгновенно стали классикой математики. (2-е английское издание с 10-го немецкого вышло в издательстве Open Court Press в 1971 году и до сих пор переиздается.) (Есть и русский перевод этой книги: *Гильберт Д.* Основания геометрии. – М.-Л.: Гостехиздат, 1948.– *прим. перев.*)

Книга Гильберта названа не *началами* геометрии, как у Евклида; она была именно *Grundlagen* – *основами* или *основаниями* – геометрии. Гильберту было важнее не доказывать теоремы геометрии, а поставить ее на прочное аксиоматическое основание. Одной из целей

[1] Reid, *Hilbert*, 25.

построения основ геометрии была демонстрация непротиворечивости ее аксиом, то есть что они никогда не приводят к противоречиям. Гильберт сделал это, построив аналог своей геометрии на плоскости вещественных чисел. В сущности, это была аналитическая геометрия в декартовой системе координат. Непротиворечивость геометрии Гильберта перешла затем в проблему непротиворечивости арифметики вещественных чисел.

Гильберт был не единственным математиком, интересовавшимся в то время созданием основ математики. В 1889 году Джузеппе Пеано (1858–1932) применил аксиоматический подход в той области, где только далекие от математики люди видели в этом необходимость, – обосновании арифметики натуральных чисел. Менее известный тогда (а ныне высоко ценимый) Готлоб Фреге (1848–1925) переосмыслил математическую логику на радикально новой системе обозначений, которую он описал в 1879 году в брошюре под названием *Begriffschrift* (примерно «*исчисление понятий*»)[1]. Фреге написал свои собственные *Grundlagen – Grundlagen der Arithmetik* (1884), где он попытался на базе математической логики построить основы арифметики вещественных чисел. Позже Фреге развил эту систему в своем большом труде и в 1893 году выпустил его первый том *Grundgesetze* (или Основные законы) *der Arithmetik*, где соединил теорию множеств и математическую логику для обоснования законности вещественных чисел.

С этими основами, заложенными незадолго до смены веков, математика, казалось, была на правильном пути, и Давида Гильберта пригласили для вступительной речи на Втором Международном математическом конгрессе в Париже в августе 1900 года. Речь должна была дать толчок математике нового столетия, однако у Гильберта не было уверенности в том, что же нужно сказать.

Гильберт обратился за советом к своему хорошему другу по университету Кёнигсберга математику литовского происхождения Герману Минковскому (1864–1909). Минковский предложил сделать речь Гильберта взглядом вперед, а не назад:

> Было бы очень заманчиво попытаться заглянуть в будущее и перечислить задачи, на которых математики должны испытать себя в наступающем веке. Такой подход заставит людей говорить о вашей лекции десятилетия спустя[2].

---

[1] Английский перевод: Jean van Heijenoort, ed., *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931* (Harvard University Press, 1967), 1–82.

[2] Reid, *Hilbert*, 69.

Таким образом, 8 августа 1900 года Гильберт начал свою речь, выстроенную именно так, как предлагал Минковский:

> Кто из нас не был бы рад приподнять завесу, за который скрывается будущее; бросить взгляд на перспективы нашей науки и тайны ее развития в грядущих столетиях? Каковы главные цели, к которым будут стремиться новые поколения ведущих математиков? Какие новые методы и новые факты на бескрайних и плодородных просторах математической мысли откроют новые столетия?[1]

Затем Гильберт в общем и целом коснулся некоторых из проблем, которые потребуют решения от математиков нового столетия. Он уверил своих слушателей, что эти задачи уже ждут своего решения:

> Хотя кажется, что эти задачи нам непосильны и мы беспомощны перед ними, мы, однако, твердо убеждены, что их решения должны вытекать из конечного числа чисто логических процессов... Эта уверенность в разрешимости каждой математической проблемы – мощный стимул в работе. Мы слышим в себе вечный зов: есть проблема; ищите ее решение; вы можете найти его только разумом, ибо в математике нет *ignorabimus*[2].

Математики в аудитории Гильберта легко разложили несколько необычное латинское слово на глагол и время и поняли его как «Мы не будем знать». Но часть его слушателей, наверное, связала его с известной лекцией 1876 года, которую физиолог Эмиль Дюбуа-Реймонд (1818–1896) закончил довольно пессимистично: «что касается загадки материи и силы... ученый должен раз и навсегда смириться с вынесенным ему еще более жестким приговором: Ignorabimus»[3].

Согласно Дюбуа-Реймонду, природа материи и энергии останется всегда непознанной. Оптимизм Гильберта совершенно не мог мириться с таким положением. Он ясно дал понять, что в математике нет «Мы не будем знать».

Затем Гильберт призвал своих коллег к решению 23 нерешенных проблем из нескольких областей математики. (Из-за ограниченного времени в устном докладе были упомянуты только 10; все 23 проблемы появились в опубликованном варианте.) Хотя некоторые из задач

---

[1] Цитируется по Ben H. Yandell, *The Honors Class: Hilbert's Problems and Their Solvers* (A. K. Peters, 2002), 389. Английский вариант речи был первоначально опубликован в *Bulletin of the American Mathematical Society*, Vol. 8 (July 1902) в переводе Dr. Mary Winston Newson. Слегка пересмотренный вариант появился в Jeremy J. Gray, *The Hilbert Challenge* (Oxford University Press, 2000), 240.

[2] Цитируется по Yandell, *The Honors Class*, 395.

[3] Цитируется по Gray, *The Hilbert Challenge*, 58.

были весьма отвлеченными, остальные были фундаментальными для своих областей.

Пункт № 1 касался «Проблемы Кантора о мощности континуума»: представляет ли собой мощность континуума следующее трансфинитное число вслед за мощностью множества натуральных чисел или между ними есть другие трансфинитные числа, которые должны быть приняты во внимание. К тому времени работа Георга Кантора стала менее спорной, а Гильберт был одним из самых больших ее сторонников.

Проблема № 2 была связана с «Непротиворечивостью аксиом арифметики». При обосновании непротиворечивости своей геометрии Гильберт опирался на непротиворечивость системы вещественных чисел и ее арифметики. Теперь сама система вещественных чисел нуждалась в аксиоматизации и в «*доказательстве того, что она непротиворечива в том смысле, что конечное число логических шагов, опирающихся на нее, никогда не приведет к противоречивому результату*»[1].

Проблема № 10 у Гильберта по-немецки звучит так:

Entscheidung der Lösbarkeit einer diophantischen Gleichung.

Обратите внимание на слово *Entscheidung*. Это очень важное слово в данной книге. Оно означает *решение*, *разрешимость*, *вычислимость*. Полностью 10-я проблема Гильберта звучит следующим образом:

10. Определение разрешимости диофантова уравнения

Дано диофантово уравнение с произвольным числом неизвестных и рациональными целочисленными коэффициентами. *Разработать процесс, согласно которому за конечное число шагов можно установить, разрешимо ли это уравнение в рациональных числах*[2].

Да, прошло 1650 лет со времен *Арифметики* Диофанта, а математики все еще бились над диофантовыми уравнениями. В то время как некоторые математики работали с определенными видами диофантовых уравнений, Гильберт потребовал *общего* процесса решения. Заметьте, что он не требует общего метода *решения* всех диофантовых уравнений. Все, что ему нужно, – *установить* разрешимость.

---

[1] Yandell, *The Honors Class*, 397.

[2] Yandell, *The Honors Class*, 406. Она переведена как «Разрешимость решаемости диофантовых уравнений» в Ivor Grattan-Guinness, «A Sideways Look at Hilbert's Twenty-three Problems of 1900», *Notices of the American Mathematical Society*, Vol. 47, No. 7 (August 2000), 752–757. Термин «рациональный целочисленный» означает обычные, известные нам целые числа.

Возьмем любое диофантово уравнение. Оно разрешимо? У него есть решение? Гильберту нужен процесс, и, похоже, у него нет ни малейшего сомнения в том, что такой процесс существует. Его нужно только найти.

Слова, которые употребил Гильберт при постановке данной проблемы, будут задавать тон этой частной *Entscheidung*-проблеме и другим *Entscheidung*-проблемам на годы вперед. Гильберту был нужен *процесс* с *конечным числом операций*. Короче говоря, Гильберту был нужен *алгоритм*, но это слово (*Algorithmus* – и по-английски, и по-немецки) тогда не употреблялось, по крайней мере в его современном смысле. В современном значении слово получило распространение в литературе о компьютерах только в 1960-х годах[1].

В той речи 1900 года Гильберт пригласил свою аудиторию «приподнять завесу», за которой скрывается XX век. Ни он, ни кто-то другой, наверное, не могли вполне представить себе зрелище, которое открылось бы им. Стоило физикам предположить, что они близки к абсолютному знанию, как их надежды рухнули в 1905 году, известном теперь как *annus mirabilis* (*год чудес*) физика Альберта Эйнштейна (1879–1955). В одном только этом году Эйнштейн опубликовал докторскую диссертацию и еще четыре статьи, в которых закладывались принципы относительности и квантовой механики.

Прежнего ощущения, что вселенная линейна, Евклидова и полностью детерминирована, уже больше не было. Пространство и время сорвались со своих якорей в релятивистском мире. В известной статье 1907 года об относительности друг Гильберта Герман Минковский придумал слово *Zaumreit* или *пространство-время*. (Минковский приехал в Гёттинген в 1902 году, но в 1909 году внезапно умер от аппендицита.) В итоге самым известным результатом столетия в квантовой механике стал так называемый Принцип неопределенности (1927).

Возможно, отвечая на эти новые сдвиги и неопределенности, в вызывающих и пугающих направлениях стало развиваться современное искусство и музыка. Прежние изобразительные формы были отброшены и воссозданы в кубистической живописи и скульптуре. Как

---

[1] См. *Oxford English Dictionary*, 2nd edition, I, 313. См. также вводные страницы Donald E. Knuth, *The Art of Computer Programming, Volume 1, Fundamental Algorithms*, 3rd edition (Addison-Wesley, 1997). Самый известный из всех алгоритмов – метод Евклида для нахождения наибольшего общего делителя двух чисел, хотя первое известное употребление термина «Алгоритм Евклида», видимо, относится только к началу XX века.

только «объективная» реальность стала менее правдоподобной, сюрреалисты погрузились в свое подсознание и иррациональные мечты.

В музыке хроматизм таких поздних романтиков, как Вагнер и Дебюсси, казался почти незыблемым, как вдруг уступил место резким диссонансам и новым рваным ритмам бунтарской *Весны Священной* Игоря Стравинского на ее Парижской премьере 1913 гола. В начале 1920-х годов изобретение австрийским композитором Арнольдом Шёнбергом 12-тональной музыки представляло собой не что иное, как пересмотр постулатов музыкальной гармонии ради создания «неевклидовой» музыки.

Математика XX века не избежала этих волнений. Первые резкие ноты прозвучали в 1902 году.

Готлоб Фреге, родившийся в немецком городе Висмар в 1848 году, получил степень доктора философии в Гёттингене за два десятилетия до прихода туда Гильберта, а затем начал преподавать в университете Йены, где оставался в течение 44 лет. Первый том работы его жизни *Grundgesetze der Arithmetik* вышел в 1893 году, где предпринята попытка систематического построения всей математики, начиная с математической логики – направления, известного сейчас как *логицизм*. Этот первый том продавался так плохо, что издатель даже не хотел слышать о втором томе, поэтому в 1902 году Фреге попытался издать его за свой счет.

Тем временем первый том *Grundgesetze der Arithmetik* обрел нового ценного читателя.

Им был Бертран Рассел (1872–1970), необыкновенный человек, который публиковал свои первые работы по математике еще при королеве Виктории и прожил так долго, что протестовал против Вьетнамской войны. Рассел родился в аристократической и интеллигентной семье. Его дедушка Джон Рассел (1792–1878) был премьер-министром Англии; его крестный отец Джон Стюарт Милль (1806–1873) был философом-прагматиком. Интерес к математике проявился у Рассела рано:

> В возрасте одиннадцати лет я начал с Евклида вместе с моим братом в качестве наставника. Это было одно из важных событий в моей жизни, ослепляющим, как первая любовь. Я не мог себе представить, что в мире существует еще что-то столь же восхитительное. После того как я изучил пятое предположение, мой брат сказал мне, что его обычно считают сложным, хотя я не обнаружил в нем никакой сложности. Тогда его впервые осенило, что у меня могут быть некоторые умственные способности[1].

---

[1] Bertrand Russell, *The Autobiography of Bertrand Russell, 1872–1914* (George Allen and Unwin Ltd, 1967), 36.

В 1902 году Рассел работал над книгой *Принципы математики* (вышедшей в следующем году) и обнаружил теоретико-множественную проблему как у Пеано, так и у Фреге.

Элементами множества могут быть другие множества, а множества могут включать в качестве элементов еще и *самих себя*. Рассел подумал: что это за множество, которое содержит все множества, которые не содержат себя? Содержит ли это множество себя? Если нет, то это – множество, которое не содержит себя, и, таким образом, оно должно содержать себя; но если оно действительно содержит себя, то это уже не множество, которое не содержит себя.

Теперь это называется парадоксом Рассела, и он стал последним из нескольких парадоксов, которые мучили математиков на протяжении, по меньшей мере, двух тысячелетий. Позже Рассел провел аналогию с городом брадобреев, которые бреют всех тех, кто не бреется сам. А кто бреет брадобрея?

Рассел написал письмо Фреге с вопросом о множестве, которое содержит все множества, не содержащие себя[1], и Фреге был обескуражен. Он быстро написал приложение ко второму тому *Grundgesetze der Arithmetik*, но проблема так и не была снята. Парадокс оказался основным недостатком, который слегка поколебал главную работу жизни Фреге.

Обнаруженный Бертраном Расселом парадокс происходил из способности множества включать себя в качестве своего элемента. Если бы такой вид самовключения был исключен, теория множеств могла бы освободиться от риска возникновения парадоксов. Рассел начал разрабатывать *теорию типов*, которую он слегка затронул в *Принципах математики*, а затем обсудил более подробно в статье 1908 года[2]. Рассел построил иерархию множеств. Внизу иерархии множество типа 1 могло содержать только элементарные объекты (например, числа). Множества типа 1 могли принадлежать только множествам типа 2. Множества типа 2 могли принадлежать только множествам типа 3 и т. д.

К тому времени, когда Рассел опубликовал статью 1908 года, в его планах было уже нечто большее. Рассел готов был начать работу над вторым томом *Принципов математики*, а его первый учитель и наставник Альфред Норт Уайтхед (1861–1947) тоже готовился писать второй том *своей* более ранней книги *Трактат об универсальной*

---

[1] *From Frege to Gödel*, 124–125.

[2] Bertrand Russell, «The Theory of Types» in *From Frege to Gödel*, 150–182.

алгебре (1898). Рассел и Уайтхед поняли, что их цели пересекаются и примерно в 1906 году начали работать вместе над тем, что должно было стать самой важной со времен Аристотеля книгой по логике.

Почти 2000 страниц *Principia Mathematica* (*Основания математики*) А. Н. Уайтхеда и Бертрана Рассела были изданы в трех томах в 1910, 1912 и 1913 годах. В отличие от более ранней *Principia Mathematica* – заголовка, под которым иногда известна *Philosophiæ Naturalis Principia Mathematica* (1689) Исаака Ньютона, – у Уайтхеда и Рассела латинским было только название. Возможно, на выбор такого названия повлияла гораздо более короткая *Principia Ethica* (1903) их коллеги по Кембриджу Джорджа Эдварда Мура (1873–1958)[1]. Уайтхед, Рассел и Мур были членами Апостолов Кембриджа, элитного тайного общества, посвятившего себя написанию философских статей и поеданию тостов с сардинами.

Хотя *Principia Mathematica* была написана не на латыни, она была и не совсем английской. Большая часть книги была густо испещрена формулами, которые со слов одного из первых ее читателей покрывали страницы «как куриные следы на снегу птичника зимним утром»[2].

Principia Mathematica объединяла теорию типов и математическую логику, в основном Пеано и Фреге, но в обозначениях Пеано, а не Фреге с его своеобразными символами. *Principia Mathematica* продолжает логицизм Фреге и достигает кульминации, когда Уайтхед и Рассел доказывают, что:

$$1 + 1 = 2$$

Это труднее, чем кажется![3]

До этого интерес Давида Гильберта к логицизму был довольно неровным. В 1904 году Гильберт выступил на Третьем Международном математическом конгрессе (проходившем в Гейдельберге) с докладом «Об основаниях логики и арифметики», где наметил некоторые из

---

[1] I. Grattan-Guinness, *The Search for Mathematical Roots, 1870–1940: Logics, Set Theories and the Foundations of Mathematics from Cantor through Russell to Gödel* (Princeton University Press, 2000), 380.

[2] Grattan-Guinness, *The Search for Mathematical Roots*, 454.

[3] Предварительный результат – в разделе *54.43: «Из этого предположения будет следовать, если будет определено арифметическое сложение, что 1 + 1 = 2», – более просто дан в сокращенном варианте: Alfred North Whitehead and Bertrand Russell, *Principia Mathematica to *56* (Cambridge University Press, 1997), 360. Фактически доказательство заканчивается только в *110.643 в Томе II со скромным выводом «Предположение выше иногда полезно».

возможных подходов, тогда как в *Principia Mathematica* во главу угла уже был поставлен именно логицизм.

С выходом *Principia Mathematica* Гильберту и Расселу, конечно же, было бы интересно начать сотрудничество в логике и математике, но вмешались мировые события. 4 августа 1914 года Великобритания объявила войну Германии, которая днями раньше объявила войну России и Франции. Великой войне суждено было продолжаться до 1918 года.

Ни Рассел, ни Гильберт не были милитаристами. В 1914 году немецкое правительство попросило видных ученых и деятелей искусства подписать обращение, опровергающее «вражескую ложь и клевету». Гильберт не мог определить истинность этих утверждений о Германии (своего рода политическая *Entscheidung*-проблема) и потому отказался его подписать[1]. Рассел, будучи всю жизнь активным противником войн, принимал участие во многих общественных протестах и в 1916 году был лишен должности в Колледже Тринити, а позже заключен на пять месяцев в тюрьму[2].

В 1917 году Гильберт пригласил Рассела читать лекции в Гёттингене. Но даже если бы британское правительство не конфисковало паспорт Рассела, возможность такого визита было трудно представить, пока страны находились в состоянии войны[3].

11 сентября 1917 года Гильберт снова проявил отвагу на поле боя за основания математики, выступив в Швейцарском математическом обществе в Цюрихе с докладом на тему «Аксиоматическое мышление». (Хотя война все еще шла, Гильберт мог встречаться с математиками других стран в Цюрихе, так как Швейцария придерживалась нейтралитета.) Это выступление положило начало тому, что в начале 1920-х годов стало известно как *Программа Гильберта*, которая отходила от логицизма и ставила своей целью строгую аксиоматизацию всей математики. Для анализа аксиоматических систем Гильберт предложил «метаматематику» и «теорию доказательств», которые использовали бы математическую логику, для того чтобы делать выводы о конструкции других математических систем.

---

[1] Reid, *Hilbert*, 137.

[2] Математик Г. Х. Харди (G. H. Hardy) позже написал об этих событиях памфлет, выпущенный в 1942 году издательством Кембриджского университета и распространявшийся частным образом. Он был напечатан повторно: G. H. Hardy, *Bertrand Russell and Trinity: A College Controversy of the Last War* (Cambridge University Press, 1970).

[3] Grattan-Guinness, *Search for Mathematical Roots*, 471, сноска 28.

Этот подход в математике известен как *формализм*. В понимании Гильберта построение формальной математической системы начинается с определений, аксиом и правил построения теорем из аксиом. В идеале построенная система должна обладать четырьмя взаимосвязанными свойствами:

- независимость;
- непротиворечивость;
- полнота;
- разрешимость.

Независимость означает, что нет никаких лишних аксиом – ни одна аксиома не может быть выведена из других аксиом. Математики подозревали, что пять постулатов Евклида *не* обладают свойством независимости. Именно поэтому они пытались вывести пятый постулат из остальных четырех. Позже было установлено, что постулаты Евклида действительно независимы.

Непротиворечивость – это, безусловно, самая важная характеристика любой аксиоматической системы. Не должно быть возможности вывести две теоремы, противоречащие друг другу!

Предположим, например, что разрабатывается некая новая математическая система. Эта система состоит из символов, аксиом и правил, которые применяются для вывода теорем из аксиом. То, что мы в основном делаем, – применяем аксиомы для вывода теорем, наших доказательств. Но правила подразумевают также синтаксис правильной формулы (well-formed formula нередко называется WFF, а произносится «WOOF»), которая допустима в системе. Сначала можно составить правильную формулу, не выводя ее в системе, а затем попытаться показать, что она – следствие аксиом, то есть результат применения при доказательстве аксиом и правил.

Я собираюсь показать две правильные формулы гипотетической математической системы. Вот первая формула, которую мы назовем *A*:

тарабарщина = трам-пам-пам.

Знак равенства означает, что выражения в обеих его частях считаются равными в некотором смысле. Вот формула *B*:

тарабарщина ≠ трам-пам-пам.

Она отличается от формулы *A* лишь тем, что на месте знака равенства в ней стоит знак неравенства. Формула *B* – отрицание или противоречие *A*.

Формулы *A* и *B* противоположны. Лишь одна из них может быть истинной. Сегодня понятие «истина» в математической логике порой так же скользко, как в жизни. У меня нет никакого желания ударяться здесь в метафизические рассуждения, поэтому я определю истину просто как «согласие с аксиоматическими допущениями».

Если из аксиом выводятся *обе* формулы *A* и *B*, то аксиоматическая система противоречива, но этого мало – она еще и *бессмысленна*. Бессмысленна потому, что это противоречие расползается по всей системе и делает в ней всё одинаково ложным и истинным одновременно, логическая катастрофа, обычно известная как *ex falso quodlibet* (ложь влечет все, что угодно).

Это и есть непротиворечивость.

Полнота – возможность вывести из аксиом *все* истинные формулы. Истинные формулы выводятся путем доказательств. Если из аксиом невозможно вывести *ни одну* из формул *A* или *B* (то есть не доказуемы ни *A*, ни *B*), то говорят, что аксиоматическая система неполна. Что такое истина? Может быть, она нам вообще неизвестна, или, может быть, нас осенила прекрасная мысль, которая истинна, но мы просто не можем этого доказать.

Различие между истинностью и доказуемостью может быть тонким: если что-то недоказуемо, то это обычно значит, что у нас нет уверенности в истинности этого, однако это не мешает нам изрекать истины без надлежащих доказательств. Например, почти все уверены в истинности гипотезы Гольдбаха: любое целое четное число, большее двух, есть сумма двух простых чисел. Однако ее называют «гипотезой», потому что она все еще остается одной из великих недоказанных математических проблем всех времен.

(Здесь я несколько упростил различие между «доказуемостью» и «истинностью». Доказуемость – *синтаксическое* понятие; оно основано на аксиомах системы и правилах, применяемых для вывода теорем. Тогда как истина – *семантическое* понятие, зависящее от реального смысла, которым мы наделяем символы системы. Я остановлюсь на этом вопросе подробнее в части III данной книги.)

Столь же важной для Гильберта была разрешимость, или Entscheidung. Ему была нужна разрешающая процедура – общий метод определения доказуемости любой заданной правильной формулы (well-formed formula, WFF).

Если выяснилось, что математическая система неполна, следует ли из этого, что разрешающей процедуры тоже не существует? Не обязательно. Допустим, что ни одна из формул *A* и *B* не может быть

доказана. Система неполна, но может существовать разрешающая процедура, которая могла бы проанализировать обе формулы *A* и *B* и прийти точно к такому же заключению о том, что ни одна из них не может быть доказана. Разрешающая процедура должна существовать, даже несмотря на неполноту системы.

Лучше и строже была бы, конечно, такая разрешающая процедура, которая определяла бы не доказуемость, а истинность. Подобная процедура выясняла бы истинность формул *A* и *B*, даже если ни одна из них не может быть доказана, то есть выведена из аксиом.

Впервые Гильберт предложил идею разрешающей процедуры в своей речи в 1900 году в Париже в связи с диофантовой проблемой. Доклад Гильберта «Аксиоматическое мышление» в 1917 году в Цюрихе также затрагивал «проблему разрешимости математической задачи за конечное число шагов». Из всех аспектов аксиоматических систем, сказал он, разрешимость – «самая известная и наиболее обсуждаемая, поскольку ведет к пониманию сущности математического мышления»[1]. (Конечно, когда Гильберт говорит «самая известная и наиболее обсуждаемая», он, наверное, имеет в виду свое математическое окружение, а именно: себя, своих коллег и студентов Гёттингена. Уайтхед и Рассел в *Principia Mathematica* вообще не беспокоились ни о полноте, ни о разрешимости.)

Возможно, Гильберт был первым, кто соединил слова *Entscheidung* и *Problem*, но первое письменное употребление этого слова из пяти слогов встречается у одного из ассистентов Гильберта Генриха Бехмана (1891–1970) в его выступлении 10 мая 1921 года в Гёттингенском математическом обществе под названием «Entscheidungsproblem und Algebra der Logik». В историческом плане описанная Бехманом гипотетическая разрешающая процедура просто поражает (курсив – из немецкого оригинала неопубликованного архива документов Бехмана):

> Наиболее важная характерная черта этой проблемы – в том, что в качестве метода доказательства принимается лишь *механическое вычисление* согласно заданным инструкциям, без какой бы то ни было умственной деятельности в самом строгом смысле. Если хотите, можно говорить о *механическом*, или *машиноподобном*, мышлении[2]. (Возможно, впоследствии такую процедуру могла бы выполнять машина.)

---

[1] William Ewald, ed., *From Kant to Hilbert: A Source Book in the Foundations of Mathematics* (Oxford University Press, 1996), Vol. II, 1113.

[2] Цитируется по Paolo Mancosu, «Between Russell and Hilbert: Behmann on the Foundations of Mathematics», *The Bulletin of Symbolic Logic*, Vol. 5, No. 3 (Sept. 1999), 321.

Если бы Бехман довел эту мысль до логического завершения, мы бы говорили сегодня о машинах Бехмана, а не о машинах Тьюринга!

В 1922–1923 учебном году Гильберт вел курс логических основ математики и тоже начал употреблять слово Entscheidungsproblem[1], но из Гёттингена в большой математический мир оно вышло фактически только в 1928 году. В этом году ассистент Гильберта Вильгельм Аккерман (1896–1962) помог собрать учебные лекции Гильберта (некоторые относились к 1917–1918 учебному году) в тонкую книжку, вышедшую под названием *Grundzüge der Theoretischen Logik*[2] (переводится как *Основы математической логики*), книгу, которая известна теперь как «Hilbert & Ackermann».

Hilbert & Ackermann по охвату тем и амбициям даже не приближалась к *Principia Mathematica*. Она касалась лишь основ математической логики без всякой теории множеств или логицизма. Однако по своему влиянию она оказалась далеко за пределами своих скромных 120 страниц. Основу книги составляло *engere Funktionenkalkül*, или «ограниченное функциональное исчисление» (более известное теперь как «логика предикатов первого порядка»), которое включало вопросы полноты и разрешимости.

Одним из первых читателей Hilbert & Ackermann был австрийский студент-математик из Вены по имени Курт Гёдель (1906–1978). О логике предикатов первого порядка Гёдель прочел следующее:

> Полна ли аксиоматическая система, хотя бы в том смысле, что все логические формулы, которые верны для любой области определения ее членов, могут быть выведены из них, – это все еще нерешенный вопрос[3].

Отрывок относится к формулам логики первого порядка, которые истинны вне зависимости от интерпретации пропозициональных функций (известных теперь как предикаты) и области определения этих функций. Могут ли все эти «всюду справедливые» – как их называли – формулы быть выведены из аксиом? Гёдель принял вызов, и его докторская диссертация 1929 года показала, что логика предикатов первого порядка полна в этом смысле. Она известна как теорема

---

[1] Wilfried Sieg, «Hilbert's Programs: 1917–1922», *The Bulletin of Symbolic Logic*, Vol. 5, No. 1 (March 1999), 22.

[2] D. Hilbert and W. Ackermann, *Grundzüge der Theoretischen Logik* (Verlag von Julius Springer, 1928). Второе издание вышло в 1938 году и отражало новейшие исследования прошедшего десятилетия, а английский перевод второй немецкой редакции вышел в 1950 году в издательстве Chelsea Publishing Company. Английского перевода первого немецкого издания нет.

[3] Hilbert & Ackermann, *Grundzüge der Theoretischen Logik*, 68.

Гёделя о полноте, и если бы ни этот вклад Гёделя в развитие математической логики, его не так помнили бы сегодня. Но для Гёделя это было лишь начало.

Полнота логики предикатов первого порядка была важным, хотя и ожидаемым, результатом. Она показала, что аксиомы и механизмы доказательства были достаточны для получения всех всюду справедливых высказываний. Однако математическая логика не существует в вакууме. Одной из важнейших целей логики предикатов было обеспечение прочного фундамента и каркаса для чисел и арифметики. Для создания основ теории чисел такой подход требует добавления к логической системе аксиом. Это было главной целью *Principia Arithmetica*. Станет ли логика предикатов первого порядка после добавления этих аксиом полной в том, гораздо более сильном смысле, что каждое высказывание или его отрицание доказуемо? Иногда это называется «полнотой отрицания», и она – гораздо более трудная цель. Это была следующая задача, за которую взялся Гёдель.

Весной 1930 года Давид Гильберт завершил преподавание. Ему было 68 лет. Позже в том же году ему было присвоено звание почетного гражданина Кёнигсберга, где он родился. Отправляя послание «Логика и познание природы», Гильберт был, как всегда, оптимистичен[1]. Прошло 30 лет с тех пор, как он сказал своей парижской аудитории, что для математика не существует слов «Мы не будем знать», и теперь он вновь заявлял это: «Нет *ignorabimus* в математике и, по-моему, ни в одной другой области естествознания». Гильберт рассказал, как философ Огюст Комт однажды сказал, что мы никогда не узнаем химического состава далёких звезд, и как эта задача всего несколькими годами позже была решена:

> Истинная причина того, что Комт не способен был решить неразрешимую задачу, – по-моему, в том, что неразрешимых задач нет вообще. На глупое *ignorabimus* мы отвечаем:
>
> Мы должны знать.
> Мы будем знать.

Wir müssen wissen. Wir werden wissen.

За день до того, как Гильберт стал почетным гражданином Кёнигсберга, Гёдель тоже был там на конференции по математике.

7 сентября 1930 года Гёдель заявил, что ему удалось доказать, что добавление к логике предикатов первого порядка аксиом, позволяющих вывести арифметику (включая сложение и умножение), делает

---

[1] *From Kant to Hilbert*, Vol. II, 1157–1165.

систему неполной. В такой системе он вывел и формулу, и ее отрицание. Если арифметика непротиворечива, то одно из этих утверждений должно быть истинным. Однако ни одно из них не могло быть доказано.

С помощью метода, названного позже гёделевой нумерацией, Гёдель воспользовался разработанной в рамках системы арифметикой, чтобы поставить в соответствие каждой формуле и каждому доказательству номер. После чего ему удалось построить формулу, которая утверждала собственную недоказуемость. Это созвучно математической формулировке Парадокса лжеца («Все, что я говорю, включая и это высказывание, есть ложь»), но на самом деле это не так. Формула ничего не утверждает о своей истинности или ложности, но вместе с этим она *недоказуема*. Если арифметика непротиворечива, то эта формула не может быть ложной, потому что это привело бы к противоречию. Формула должна быть истинной – но только в метаматематическом смысле, потому что истина не является собственным понятием логической системы – что означает, что она действительно недоказуема.

Статья Гёделя была опубликована в следующем году под названием «О формально неразрешимых предложениях *Principia Mathematica* и родственных систем I»[1]. Римская цифра I указывала на то, что Гёдель намеревался развить свою статью дополнительными доказательствами. Но статья оказала настолько сильное воздействие, что вторая часть уже не понадобилась.

Одной из решающих предпосылок для теоремы о неполноте было то, что арифметика непротиворечива. Как следствие Гёдель показал также, что доказательство непротиворечивости арифметики в рамках

---

[1] В оригинале статья называлась «Über formal unentscheidbare Sätze der Principia mathematica und verwandter Systeme I» и была опубликована в *Monatshefte für Mathematik und Physik*, Vol. 38 (1931), 173–198. Первый английский перевод был сделан Бернардом Мельтцером (Bernard Meltzer) из Университета Эдинбурга и появился в книге Kurt Gödel, *On Formally Undecidable Propositions of Principia Mathematica and Related Systems* (Basic Books, 1962; Dover Publications, 1992). Второй перевод профессора Эллиота Мендельсона (Elliott Mendelson) из Королевского колледжа Нью-Йорка появился в книге Martin Davis, ed. *The Undecidable: Basic Papers on Undecidable Propositions, Unsolvable Problems and Computable Functions* (Raven Press, 1965), 5–38. Третий перевод Йена ван Хейенорта (Jean van Heijenoort) (при участии Гёделя) появился в его книге *From Frege to Gödel*, 596–616. Кроме того, есть перевод в книге Kurt Gödel, *Collected Works, Volume I, 1929–1936* (Oxford University Press, 1986), 144–195. Статью часто обозначают как «Gödel 1931».

самой системы было невозможно. Поскольку определенные формулы не могли быть ни доказаны, ни опровергнуты, оказалось, что эти формулы противоречивы. (Значит ли это, что арифметика и элементарная теория чисел противоречивы? Едва ли, и никто не верит, что это так. Проблема в том, что непротиворечивость не может быть доказана в рамках самой системы.)

На слушаниях теоремы Гёделя о неполноте у Давида Гильберта была довольно странная для математика реакция. Он был «несколько рассержен»[1], но в конечном счете стал включать результаты Гёделя в свою программу.

Иные математики просто потеряли интерес к математической логике. Бертран Рассел, видимо, постоянно испытывал упадок сил после написания *Principia Mathematica*:

> Наконец работа была завершена, но мой мозг так никогда и не оправился полностью от напряжения. С тех пор я стал определенно менее, чем прежде, способным к контакту со сложными абстракциями. Это часть причины, хотя ни в коем случае не вся, изменения характера моей работы[2].

Рассел начал искать другие интересы, например писать о философии, политике и общественных проблемах. В 1950 году он получил Нобелевскую премию по литературе «в знак признания его разнообразных и значительных произведений, в которых он отстаивает человеческие идеалы и свободу мысли»[3]. К тому времени многие уже забыли, что сначала он был математиком.

Венгерский математик Джон фон Нейман (1903–1957), который бывал в Гёттингене в середине 1920-х годов, после Гёделя тоже забросил логику (с его слов), но позже способствовал применению математической логики при разработке электронных цифровых машин.

Теорема Гёделя о неполноте была, конечно, не самым худшим для Гёттингена. В 1933 году нацистская партия приказала отстранить всех евреев от преподавания в немецких университетах. Для Гёттингена, где в течение многих десятилетий единственным критерием были интеллектуальные способности, указ оказался губительным. Рихард Курант (1888–1972) уехал в Соединенные Штаты, где он нашел место в университете Нью-Йорка. (Сегодня Институт математических наук Куранта занимает целое здание на 4-й Западной улице Манхэттена.) Герман Вейль (1885–1955) не был евреем, но еврейкой

---

[1] Reid, *Hilbert*, 198.
[2] The Autobiography of Bertrand Russell, 1872–1914, 153.
[3] http://nobelprize.org/nobel prizes/literature/laureates/1950.

была его жена. Как и Альберт Эйнштейн, Вейль уехал в Институт перспективных исследований в Принстоне, Нью-Джерси. Пауль Бернайс (1888–1977) потерял работу преподавателя, но оставался самым верным помощником Гильберта вплоть до отъезда в Цюрих. Бернайсу в значительной степени приписывают написание двухтомника *Grundlagen der Mathematik* (1934, 1939), хотя книги выходили под именами Гильберта и Бернайса.

На банкете Гильберт оказался сидящим рядом с министром просвещения. «И как теперь с математикой в Гёттингене после его освобождения от еврейского влияния?» – спросил он Гильберта. Тот ответил: «Математика в Гёттингене? В сущности, её больше нет»[1].

Для тех математиков, которые продолжали исследования в математической логике, – теперь все больше и больше не в Гёттингене – проблемы все еще оставались нерешенными. Как вполне заслуженно в книге Hilbert & Ackermann 1928 года издания выделено курсивом, «*Entscheidungsproblem muß als das Hauptproblem der mathematischen Logik bezeichnet werden*» – «Проблема разрешимости должна считаться главной проблемой математической логики»[2]. Из теоремы Гёделя о неполноте не следовало, что разрешающего процесса не существует, но смысл ее был в том, что такой разрешающий процесс не мог определить *истинность* какой-либо формулы. В лучшем случае он мог определить доказуемость формулы.

Девять страниц книги Hilbert & Ackermann были посвящены *Entscheidungsproblem* в логике предикатов первого порядка, и почти на половине из них обсуждались «Решения проблемы разрешимости для частных случаев». Для нескольких стандартных (и общих) типов формул в математической логике уже были разработаны разрешающие процессы. И казалось не таким уж маловероятным, что общий разрешающий процесс тоже возможен.

Но этого не случилось. В 1936 году американский математик Алонзо Чёрч (1903–1995) пришел к выводу (тоже заслуживающему курсива) о том, что «*общий случай Entscheidungsproblem engere Funktionenkalkül* [логики предикатов первого порядка] *неразрешим*»[3].

Работая независимо от Чёрча и применяя совершенно другую методологию, Алан Тьюринг пришел к тому же выводу, что «Entschei-

---

[1] Reid, *Hilbert*, 205.
[2] Hilbert & Ackermann, *Grundzüuge der Theoretischen Logik*, 77.
[3] Alonzo Church, «A Note on the Entscheidungsproblem», *The Journal of Symbolic Logic*, Vol. 1 No. 1 (March 1936), 41.

dungsproblem Гильберта не может иметь решения»[1]. Так он заявляет в начале своей статьи, а в конце заключает «следовательно, Entscheidungsproblem не может быть решена»[2].

К тому времени, когда Чёрч и Тьюринг опубликовали свои работы, Гильберту было 74 года. Кроме того, Гильберт попал под подозрение нацистов, которые заинтересовались его именем *Давид*[3]. Свои последние годы Гильберт провел в одиночестве. В 1943 году он умер. На надгробной плите Гильберта в Гёттингене – слова

Wir müssen wissen.
Wir werden wissen.

Мы должны знать. Мы будем знать. За исключением того, что теперь, когда люди читают слова Гильберта, все, о чем они могут подумать, – это Гёдель, Чёрч и Тьюринг, неполнота и неразрешимость.

Во время войны родной город Гильберта Кёнигсберг был в значительной степени разрушен британской бомбардировкой. В 1945 году он перешел к русским, а после войны вошел в состав России. В 1946 году Кёнигсберг был переименован в Калининград в честь советского Председателя Верховного Совета.

Переселявшиеся туда русские намеревались разрушить все остатки немецкой культуры, а оставшиеся там немцы стали жертвами сталинских злодеяний. Расположение города на Балтийском море сделало его идеальным для военно-морской базы. Десятилетиями он был закрыт для посетителей.

После распада Советского Союза город Калининград остался в составе России, но превратился в анклав, отделенный от остальной части страны, зажатый между Литвой и Польшей и известный главным образом высоким уровнем преступности.

Проблема Y2K, которая, как некоторые предсказывали, должна была добавить к ужасам XX века завершающую катастрофу, оказалась не такой уж скверной. Вот как выдохнула первая полоса *Нью-Йорк Таймс* первым утром 2000-го года:

1/1/00:
Технология и 2000

---

[1] Alan Turing, «On Computable Numbers, with an Application to the Entscheidungsproblem», *Proceedings of the London Mathematical Society*, 2nd Series, Vol. 42 (1936), 231. (Страница 88 этой книги.)
[2] Там же, 262. (Страница 350 этой книги.)
[3] Reid, *Hilbert*, 209.

Большое облегчение;
Компьютеры выстояли
в первые часы '00

На самом деле программисты не закладывали бомбу замедленного действия в большинство важнейших систем. Программисты в большинстве своем гораздо выше этого! Более того, они провели многие часы за довольно тяжелой работой по локализации многих потенциальных проблем. Зачастую изменить компьютерные программы совсем нетрудно. Именно поэтому их называют *soft*ware (*soft – мягкий, гибкий – прим. перев.*).

Компьютерные программы разрабатываются и поддерживаются как текстовые файлы, называемые *исходным кодом*. А текстовые файлы могут быть прочитаны и проанализированы другими программами. Программисты сумели написать специальные программы для проверки существующих исходных кодов, чтобы локализовать возможные проблемные участки. Такие программы могли, например, находить имена переменных, содержащие символы «year» или «yr», после чего программист мог проверить, как программа обращается с календарными годами.

Поскольку эти потенциальные ошибки проблемы Y2K были обнаружены и устранены, кому-то должен был прийти в голову еще более дерзкий план: можно ли написать программу, которая анализирует другие программы и локализует другие ошибки? Конечно, такая программа была бы очень сложной, но после того как она была бы написана, она могла бы использоваться для отладки любой другой программы, а это было бы чрезвычайно полезно.

Да, практически это сложно, но *возможно* ли теоретически?

И ответом будет *Нет*. Общего алгоритма поиска ошибок *не может быть*. Это тоже один из неутешительных выводов статьи Алана Тьюринга о вычислимых числах и Entscheidungsproblem.

Часть II

Вычислимые числа

Глава 4

Годы учебы

Когда Алану Тьюрингу было 10 лет, кто-то дал ему книгу Эдвина Тенни Брюстера «Естественные чудеса, которые должен знать каждый ребенок». Как позже говорил Тьюринг[1], эта книга открыла глаза молодому человеку на науку и, возможно, даже оказала более глубокое влияние на его понимание отношений между людьми и машинами. Как говорилось в книге: «тело – это, конечно, машина»:

> Это весьма сложная машина, во много-много раз более сложная, чем любая машина, когда-либо созданная руками человека; но все-таки в конце концов – машина. Она была уподоблена паровому двигателю. Но это было до того, как мы узнали о принципах его работы так же много, как теперь. На самом деле она – газовый двигатель; как двигатель автомобиля, моторной лодки или самолета[2].

К началу XX века идея о том, что люди – это машины, стала настолько наивной, что могла теперь обсуждаться даже в детских книжках. Но так было не всегда. Два века отделяли Алана Тьюринга от Жюльена Оффре де Ламетри (1709–1751), французского врача и философа, чья скандальная работа 1747 года «L'Homme Machine» («Человек-машина»)[3] откровенно изображала тело человека и даже его мозг как работающую машину. Алан Тьюринг рос с представлением о том, что его тело – машина; больше всего его помнят за исследование связей между машинами и человеческим разумом.

Алан Мэтисон Тьюринг родился 23 июня 1912 года в частной лечебнице Паддингтона, районе Лондона. Его отец был чиновником Индийской гражданской службы. Его мать родилась в Мадрасе, была дочерью инженера, который заработал состояние на строительстве

---

[1] Andrew Hodges, *Alan Turing: The Enigmaa* (Simon & Schuster, 1983), 11. Вся биографическая информация о Тьюринге взята из этой книги.

[2] Цитируется в Hodges, *Alan Turing*, 13.

[3] Julien Offray de La Mettrie, *Machine Man and Other Writing*, переведенные и отредактированные Энном Томсоном (Издательство Кембриджского университета, 1996). *(Ламетри Ж. О.* Сочинения. М.: Мысль, 1976).

мостов и железных дорог в Индии. Родители Тьюринга встретились в 1907 году на корабле, шедшем из Индии в Англию, и поженились в том же году в Дублине. Они вернулись в Индию в начале 1908 года. Алан, второй из двух мальчиков, был зачат в Индии в 1911 году, но родился в Англии.

Большую часть своего раннего детства Алан и его старший брат Джон оставались в Англии на попечении семьи пенсионеров, тогда как их родители жили в Индии, что было обычным для того времени. В 1922 году Алан начал посещать Хазельхёрст (Hazelhurst), подготовительную школу в Кенте. Его главными интересами были картография, шахматы и химия[1]. В 1926 году он был принят в Шерборн (Sherborne), одну из самых старых английских общедоступных школ. Всеобщая забастовка в Шерборне в первый день первого семестра помешала Тьюрингу добраться до школы по железной дороге. Тогда Алан решил преодолеть 60 миль до школы на велосипеде; об этом подвиге сообщила местная газета[2].

В Шерборне Алан плохо сходился с другими мальчиками. Он всегда был застенчив, уединялся и казался вечно растрепанным и заляпанным чернилами. «Любая черта его характера служила поводом для насмешек, особенно его робкий дрожащий высокий голос – не то чтобы заикание, но медлительность, будто в ожидании окончания какого-то трудоемкого процесса перевода своей мысли в форму человеческой речи»[3]. Он мог бы реабилитировать себя, выделяясь в учебе, но этого не случалось. Лишь в математике он проявил признаки подлинного интеллектуального таланта.

В 1929 году Алан пребывал в восторге от «Природы физического мира» (1928), популярной и впечатляющей книги Кембриджского астронома сэра Артура Эддингтона, в которой исследовалось применение новых наук – теории относительности и квантовой теории. Кроме того, Алан был очарован одноклассником по имени Кристофер Морком, который разделял интересы Алана в науке и математике и происходил из гораздо более интересной, чем у Тьюринга, научной семьи. Дедом Кристофера по материнской линии был сэр Джозеф Сван, который в 1879 году независимо от Эдисона изобрел лампу накаливания.

Очень похоже, что гомосексуализм Алана Тьюринга обнаружился именно в то время и что Кристофер был его первой любовью. Од-

---

[1] Hodges, *Alan Turing*, 17.
[2] Hodges, *Alan Turing*, 21.
[3] Hodges, *Alan Turing*, 24.

нако нет никаких признаков того, что между этими двумя подростками имело место что-то телесное. Они проводили вместе химические опыты, обменивались математическими уравнениями и изучали новую астрономию и физику по книгам Эддингтона и сэра Джеймса Джинса, еще одного Кембриджского профессора астрономии.

Кембридж был местом, куда стремились честолюбивые английские ученые, а Тринити был его колледжем с наилучшей научной и математической репутацией. В декабре 1929 года Алан и Кристофер отправились на неделю в Кембридж, чтобы сдать экзамены и насладиться пребыванием в альма-матер Фрэнсиса Бэкона, Исаака Ньютона и Джеймса Клерка Максвелла. Результаты экзамена были опубликованы в «Таймс» неделю спустя после их возвращения в Шерборн. Алан их не сдал, а Кристофер сдал. Кристоферу нужно было собираться в Тринити, а лучшее, на что мог надеяться Алан, – это еще раз в следующем году попробовать поступить в Тринити или, может быть, в один из других колледжей Кембриджа.

Два месяца спустя Кристофер внезапно заболел и через неделю умер от бычьего туберкулеза, которым он заразился в детстве. Один из их одноклассников в Шерборне написал в письме: «Бедный, славный Тьюринг был почти сражен таким ударом. Они, должно быть, были очень хорошими друзьями»[1]. Хотя Алан Тьюринг имел другие более близкие отношения с мужчинами, очевидно, ни к кому другому он не был привязан так, как к Кристоферу Моркому.

В декабре 1930 года Тьюринг пытался еще раз сдать экзамены в Тринити и вновь не поступил. Вторым его вариантом был Кингс. К этому времени он решил сосредоточиться на математике и готовился самостоятельно, штудируя классический труд Г. Х. Харди «Курс чистой математики», вышедший тогда пятым изданием. Алан Тьюринг приступил к учебе в колледже Кингс Кембриджа осенью 1931 года.

В следующем году Тьюринг взялся за только что вышедшую книгу по математическим основам квантовой механики «Математические основы квантовой механики» («Mathematische Grundlagen der Quantenmechanik» – *нем.*) молодого венгерского математика Джона фон Неймана, чья фамилия произносится как «Ной-ман». Нейман провел середину 1920-х, работая с Давидом Гильбертом в Гёттингене, месте большинства ранних математических исследований по квантовой механике. В 1930 году он эмигрировал в Соединенные Штаты, чтобы преподавать в Принстоне, и был среди первых математиков, приня-

---

[1] Hodges, *Alan Turing*, 46.

тых в 1933 году на работу в Институт перспективных исследований. Теперь жизни Джона фон Неймана и Алана Тьюринга начали пересекаться в нескольких интересных направлениях.

Наверное, впервые Тьюринг встретился с фон Нейманом летом 1935 года, когда тот прервал свою работу в университете Принстона для чтения лекций в Кембридже по курсу почти периодических функций. Тьюринг уже был знаком с работой фон Неймана по этой теме: этой же весной Тьюринг опубликовал свою первую двухстраничную статью «Эквивалентность левой и правой почти периодичности» (*Журнал Лондонского математического общества*, 1935), в которой подробно остановился на статье фон Неймана, опубликованной в предыдущем году.

Никто, наверное, не мог предположить, что они встретятся вновь в следующем году в Принстоне (Нью-Джерси).

Интерес Тьюринга к изысканному миру математической логики начался, возможно, в 1933 году, когда он прочитал работу Бертрана Рассела 1919 года «Введение в математическую философию», которая заканчивается так:

> Если эта небольшая книжка подвигнет какого-то студента к серьезному изучению математической логики, она достигнет главной цели, ради которой была написана[1].

В весеннем семестре 1935 года Тьюринг записался на курс по основаниям математики Максвелла Хермана Александра Ньюмэна (1897–1984), известного по инициалам М. Х. А. Ньюмэн и прозвищу Макс. Макс Ньюмэн пользовался заслуженной репутацией за свою работу по комбинаторной топологии, но он, видимо, был в Кембридже еще и человеком, наиболее осведомленным в математической логике. Кульминацией курса Ньюмэна было доказательство теоремы Гёделя о неполноте. (Вводные курсы математической логики магистерского уровня до сих пор строятся подобным образом.)

В курсе Ньюмэна освещалась также нерешенная Entscheidungsproblem. «Существует ли точный метод или, как выразился Ньюмэн, *механический процесс*, который, будучи примененным к математическому утверждению, давал бы ответ о его доказуемости?»[2]. Под «механическим процессом» Ньюмэн, конечно же, подразумевал не

---

[1] Bertrand Russel, *Introduction to Mathematical Philosophy*, second edition (George Allen & Unwin Ltd, 1920; Dover Publications, 1993), 206.

[2] Hodges, *Alan Turing*, 93.

машину. Машины могут иметь способности к простой арифметике, но едва ли – к настоящей математике. Напротив, Ньюмэн указывал на тип процесса, который в конечном счете получит название *алгоритм* – набор точных (но в сущности «бессмысленных») инструкций для решения задачи. Это, наверное, то, с чего Тьюринг в начале лета 1935 года начал работу по проблеме разрешимости[1]. К этому времени ему была назначена кембриджская стипендия, которая составляла 300 фунтов в год. Позднее Тьюринг говорил, что главная идея, приблизившая его к Entscheidungsproblem, пришла к нему, когда он лежал на лугах Грантчестера, популярном месте отдыха кембриджских студентов примерно в двух милях от колледжа Кингс.

К апрелю 1936 года Тьюринг был в состоянии предоставить Максу Ньюмэну черновик своей статьи «О вычислимых числах в применении к Entscheidungsproblem»[2].

В статье Тьюринга предпринят необычный для математического доказательства подход: она начинается с описания вымышленной вычислительной машины, способной выполнять несколько простых операций. Несмотря на простоту этой машины, Тьюринг утверждает, что она функционально эквивалентна человеку, выполняющему математические операции. Он вводит эти машины для вычисления чисел. Первая, приводимая в качестве примера, машина Тьюринга вычисляет число 1/3 в двоичном виде (0,010101...). Вторая вычисляет иррациональное число, вероятно еще и трансцендентное (0,001011011101111...). Он убеждает нас в том, что можно определить машины для вычисления π, e и других известных математических констант. Тьюринг даже создает Универсальную машину, которая может моделировать действия любой другой вычислительной машины.

И все же машины Тьюринга – как назвали эти воображаемые устройства – не могут вычислить любое вещественное число. Создаваемые им машины имеют конечное число операций, и, представляя эти операции числами, он смог показать, что каждую машину можно однозначно представить одним целым числом, названным им описателем (Descripton Number). Таким образом, машины Тьюринга перечислимы. Вычислимые числа – числа, которые могут быть вычислены машинами Тьюринга, – тоже должны быть перечислимыми, но вещественные числа (из доказательства Кантора) неперечислимы. Вычислимые числа, конечно, включают алгебраические числа, а так-

[1] Hodges, *Alan Turing*, 96.
[2] Hodges, *Alan Turing*, 109.

же такие трансцендентные числа, как π и e, но поскольку вычислимые числа перечислимы, они просто не могут охватить все вещественные числа.

Машины Тьюринга небезошибочны. Можно определить машину Тьюринга, которая работает неправильно или просто вообще ничего не делает. Тьюринг делит свои машины на «приемлемые» («satisfactory») и «неприемлемые» («unsatisfactory»).

Поскольку машины Тьюринга полностью определяются своим описателем, можно создать машину Тьюринга, которая на основе анализа описателя определяет, приемлема ли конкретная машина. Тьюринг доказывает, что это не так: не существует общего процесса определения, является ли машина Тьюринга приемлемой. Единственный способ одной машине Тьюринга «понять» другую – это шаг за шагом выполнить все ее действия. Короче говоря, чтобы понять, что собирается делать машина, фактически ее нужно запустить.

То, что справедливо для машин Тьюринга, относится также и к компьютерным программам: в общем случае ни одна компьютерная программа не может проанализировать работу другой, кроме как воспроизведением ее шаг за шагом. Тьюринг также доказывает, что невозможно определить машину Тьюринга, которая делает нечто, кажущееся совсем простым, например устанавливает, напечатает ли некая машина когда-либо цифру 0. В заключительном разделе своей статьи (который обсуждается в части III этой книги) Тьюринг конструирует логико-математическое утверждение, равносильное определению, напечатает ли когда-либо 0 заданная машина Тьюринга. Но так как он уже установил, что такую машину определить невозможно, это утверждение логически недоказуемо, и, следовательно, «Entscheidungsproblem не может быть разрешена» (страница 262 статьи Тьюринга и страница 311 этой книги).

Почти в то же самое время, когда Макс Ньюмэн читал черновик статьи Тьюринга, он получил оттиск короткой статьи «Замечание о Entscheidungsproblem»[1] американского математика Алонзо Чёрча. В ней, опираясь на более раннюю свою статью[2], Чёрч тоже приходит к заключению, что Entscheidungsproblem «неразрешима».

---

[1] Alonzo Church, «*A Note no no Entschiedungsproblem*», The Journal Symbolic Logic, Vol. 1, No. 1 (Mar.1936), 40–41.

[2] Alonzo Church, «An Unsolvable Problem of Elementary Number Theory», *American Journal of Mathematics*, Vol. 58, No. 2 (Apr. 1936), 345–363. Both of Church's papers appear in Martin Davis, ed. *The Undecidable* (Raven Press, 1965).

Тьюринг был опустошен. Как правило, это означало, что статья не может быть опубликована и обречена на забвение, но Макс Ньюмэн понял, что подход Тьюринга был новаторским и значительно отличался от подхода Чёрча. Он рекомендовал Тьюрингу во что бы то ни стало направить свою статью в Лондонское математическое общество для публикации. (В опубликованной статье указано, что Общество получило ее 28 мая 1936 года.) В письме своей матери от 29 мая Тьюринг так объяснял ситуацию:

> Тем временем в Америке появилась статья, написанная Алонзо Чёрчем и решающая то же самое, но другим способом. Несмотря на это, мы с мистером Ньюмэном решили, что метод отличается настолько, что моя статья тоже имеет право на публикацию. Алонзо Чёрч живет в Принстоне, поэтому я принял вполне твердое решение о поездке туда[1].

31 мая Макс Ньюмэн написал письма и Алонзо Чёрчу, и секретарю Лондонского математического общества. Чёрчу он написал:

> Любезно присланный мне недавно оттиск вашей статьи, где вы определяете «вычислимые числа» (calculable numbers) и показываете, что Entscheidungsproblem в логике Гильберта неразрешима, несколько огорчил здесь молодого человека, А. М. Тьюринга, который только что представил для публикации статью, в которой он применил определение «Вычислимых чисел» («Computable numbers») с той же самой целью. Его трактовка – заключающаяся в описании машины, которая допускает любую вычислимую последовательность, – несколько отличается от Вашей, но представляется настолько значимой и, думаю, настолько важной, что он должен приехать и поработать с вами в следующем году, если это вообще возможно[2].

Секретарю Лондонского математического общества Ф. П. Уайту Макс Ньюмэн написал:

> Думаю, Вы знаете историю статьи Тьюринга о Вычислимых числах. По достижении заключительного ее этапа из Принстона от Алонзо Чёрча пришел оттиск статьи, во многом предвосхищающей результаты Тьюринга. Несмотря на это, я надеюсь, что статью можно будет опубликовать. Подходы в большой степени отличаются, а результат настолько важен, что разные его трактовки должны представлять интерес. Главный результат как Тьюринга, так и Чёрча состоит в том, что Entscheidungsproblem, над которой в течение очень многих лет работали ученики Гильберта, – то есть проблема поиска механического способа установить, представляет ли данная строка символов формулировку теоремы, выводимой из аксиом Гильберта, – неразрешима в ее общем виде[3].

---

[1] Hodges, Alan Turing, 113.
[2] Hodges, Alan Turing, 112.
[3] Hodges, Alan Turing, 113.

Теперь Тьюрингу нужно было добавить к своей статье приложение, показывающее, что его понятие вычислимости и понятие «эффективной вычислимости» Чёрча эквивалентны. Это приложение было получено Лондонским математическим обществом 28 августа 1936 года.

Статья Тьюринга была опубликована в *Трудах Лондонского математического общества* в ноябре и декабре 1936 года[1]. Исправление на трех страницах было опубликовано в декабре 1937 года[2]. Обзор статьи из четырех параграфов Алонзо Чёрча в *Журнале символической логики* за март 1937 года содержит высказывание «человек-вычислитель с карандашом и бумагой и с подробными инструкциями, может считаться разновидностью машины Тьюринга»[3], в котором впервые появляется термин «машина Тьюринга».

Статья Тьюринга делится на одиннадцать разделов и приложение. Она начинается с введения, которое подготавливает к описанию этой новой категории чисел, представляемых Тьюрингом.

[230]

О ВЫЧИСЛИМЫХ ЧИСЛАХ ПРИМЕНЕНИТЕЛЬНО К ENTSCHEIDUNGSPROBLEM

А. М. Тьюринга.

[Получено 28 мая 1936 г. – Прочитано 12 ноября 1936 г.]

«Вычислимые» числа кратко можно описать как вещественные числа, чье десятичное представление можно вычислить конечным способом.

Здесь Тьюринг ограничивается рассмотрением вещественных чисел и предполагает, что вычислимые числа – *подмножество* вещественных, что означает существование некоторых вещественных

---

[1] Статья была разбита на два ежемесячных выпуска (названных «частями») *Трудов*: первые 11 страниц оказались в *Томе 42, Части 3* (от 30 ноября 1936 г.), а остальные – в *Томе 42, Части 4* (от 23 декабря 1936 г.). В 1937 году части, изданные с октября 1936 года по апрель 1937 года, были изданы вместе как *2-я Серия, Том 42*. Это объясняет, почему год издания статьи Тьюринга дается по-разному: как 1936-й (год издания отдельных частей), 1937-й (год, когда она была закончена, был издан Том 42), или 1936–1937 (которые являются датами всех частей, включенных в Том 42).

[2] Конкретно Том 43, Часть 7 (выпущенный 30 декабря 1937 г.), который тогда появился во 2-ой Серии, Том 43, который включает части, выпущенные с мая по декабрь 1937 года.

[3] Alonzo Church, Review of «On Computable Numbers, with an Application to the Entscheidungsproblem», *The Journal of Symbolic Logic*, Vol. 2, No. 1 (Mar. 1937), 42–43.

чисел, которые невычислимы. Конечно, это не сразу очевидно. Под «десятичным представлением» Тьюринг подразумевает, что число 1/3 должно представляться как 0,33333..., а π должно вычисляться как 3,14159..., что, казалось бы, прямо противоречит его «конечному способу». Понятно, что мы никогда не сможем вычислить до конца десятичные числа 1/3 или π. Несмотря на это, «способ» в статье Тьюринга относится не к процессу, а к *методу* определения цифр. Метод, который говорит, что «следующая цифра – 4, следующая – 7, следующая – 0...», очевидно, можно применить для вычисления любого вещественного числа, но это не конечный метод. И 1/3, и π вычислимы посредством алгоритмов (менее или более сложных), но способ их вычисления (бесконечное деление или что-то более замысловатое) состоит из конечного числа правил.

> Хотя предмет этой статьи – как будто бы вычислимые *числа*, почти так же легко определить и исследовать вычислимые функции целой переменной, вещественной или вычислимой переменной, вычислимые предикаты и т. д. Однако в любом случае основные проблемы – одни и те же, и для подробного рассмотрения я предпочел вычислимые числа как требующие менее громоздкой техники. Вскоре я надеюсь представить отчет об отношении между вычислимыми числами, функциями и т. д. Он будет включать развитие теории функций вещественной переменной с точки зрения вычислимых чисел.

Тьюринг так никогда и не осуществил своих намерений. Гёдель тоже намеревался написать продолжение своей знаменитой статьи о неполноте и даже, опережая события, включил в ее название римскую цифру I. Гёдель не написал продолжение, потому что результаты его статьи были признаны быстрее, чем он предполагал. А Тьюринг увлекся другими вещами.

Тьюринг заканчивает первый раздел своей статьи следующим утверждением:

> Согласно моему определению, число вычислимо, если машина может записать его десятичное представление.

Заявить это в 1936 году было весьма необычно, поскольку в то время ни одна из уже построенных машин вообще не могла делать того, что требовал Тьюринг.

Тьюринг, вероятно, знал о работе английского математика Чарльза Бэббиджа (1791–1871), который создал проект Разностной машины для расчета логарифмических таблиц, а потом, примерно в 1833 году, отказался от нее ради работы над Аналитической машиной, которая больше походила на компьютер общего назначения. Бэббидж бывал также в Кембридже, а части его незаконченных машин демонстрировались в Музее науки в Кенсингтоне. Несмотря на это, идеи или терминология Бэббиджа, похоже, не оказали на Тьюринга никакого влияния.

Тьюринг мог или не мог знать о Дифференциальном анализаторе, который, начиная с 1927 года, разрабатывался Ванневаром Бушем (1890–1974) и его студентами в Массачусетском технологическом институте, но это был аналоговый компьютер, который решал дифференциальные уравнения главным образом для технических приложений. Эта машина могла быть интересна Тьюрингу с математической или технической точки зрения, но совсем не могла быть полезной в его конкретной проблеме.

Еще труднее предположить, что Тьюринг был знаком с другими, ранними компьютерными проектами середины 1930-х годов. Тьюринг определенно не знал, что студент-машиностроитель Конрад Цузе (1910–1995) в 1935 году начал строить компьютер в гостиной квартиры своих родителей в Берлине. Еще не закончился 1937 год, когда вышла работа Тьюринга, как Джордж Штибиц (1904–1995), забрав домой несколько телефонных реле со своей работы в Bell Telephone Laboratories, начал собирать двоичные сумматоры. В том же 1937 году аспирант Гарварда Говард Эйкен (1900–1973) начал исследования в области автоматизированных вычислений, приведшие к сотрудничеству между Гарвардом и IBM в создании ЭВМ Mark I[1].

В наступлении на проблемы вычислимости и Entscheidungsproblem Гильберта в это исключительное время Тьюринг был в одном строю с Алонзо Чёрчем, Эмилем Постом (1897–1954) и Стивеном Клини (1909–1994)[2], только Тьюринга можно было включить еще и в ряды тех, кто в середине 1930-х годов думал об автоматизированных вычислениях.

---

[1] Превосходное введение в эти ранние компьютеры – Paul E. Ceruzzi, *Reckoners: The Prehistory of the Digital Computer, from Relays to the Stored Program Concept, 1935–1945* (Greenwood Press, 1983).

[2] Robin Gandy, «The Confluence of Ideas in 1936» in Rolf Herken, ed., *The Universal Turing Machine: A Half-Century Survey* (Oxford University Press, 1988), 55–111; 2-nd edition (Springer-Verlag, 1995), 49–102.

Тьюринг обобщил некоторые из своих выводов, которые появятся в следующих разделах этой статьи:

> В §§9, 10 я привожу некоторые доводы с целью показать, что вычислимые числа включают все числа, которые могли бы считаться естественно вычислимыми. В частности, я показываю, что определенные большие классы чисел вычислимы. Они включают, например, вещественные части всех алгебраических чисел, вещественные части нулей функций Бесселя, числа π, *e* и т. д.

Те «числа, которые могли бы считаться естественно вычислимыми», – это числа, которые люди фактически уже вычислили и для которых существуют алгоритмы. Тьюринг даже не утруждает себя упоминанием о том, что все рациональные числа вычислимы. Это очевидно. Так же быстро он причисляет к вычислимым алгебраические числа. (Он ограничивает алгебраические числа *вещественными* частями, поскольку решения алгебраических уравнений могут иметь вещественные и мнимые части, а он ограничивается лишь вещественными числами.)

С утверждением о вычислимости алгебраических чисел Тьюринг теперь переносит обсуждение в область трансцедентных чисел. Кроме того, говорит он, вычислимы и некоторые трансцендентные числа. Бесселевы функции – это решения особого вида дифференциальных уравнений. Нули – это значения, где функции равны нулю. Когда-то они издавались в виде таблиц, поэтому должны считаться вычислимыми. (Теперь же при необходимости они обычно рассчитываются компьютерными программами.) Тьюринг не упоминает об этом, но тригонометрические и логарифмические функции обычно имеют трансцендентные значения и тоже вычислимы. Как и константы π и *e*.

То, что Тьюринг не утверждает, – так это то, что вычислимы все трансцендентные числа. Иначе вычислимые числа были бы теми же самыми, что и вещественные числа.

> Однако вычислимые числа не включают все определяемые числа, и приводится пример определяемого числа, которое невычислимо.

Это своего рода приманка. Тьюринг определит число, которое он (и ни одна из его машин) не может вычислить.

Теперь Тьюринг попадает в затруднение относительно различия между вещественными числами и вычислимыми числами:

Хотя класс вычислимых чисел довольно большой и во многом похож на класс вещественных чисел, он тем не менее перечислим.

Вычислимые числа перечислимы. Перечислимость вычислимых чисел означает, что они не то же самое, что вещественные числа, поскольку вещественные числа не перечислимы.

В §8 я исследую некоторые аргументы, которые, казалось бы, должны доказывать обратное. При надлежащем применении одного из этих аргументов получаются выводы, которые на первый взгляд похожи на выводы Гёделя†.

† Gödel, «Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, I», *Monatshefte Math. Phys.*, 38 (1931), 173–198.

Это известная теорема Гёделя о неполноте. Заметьте, что в сноске Тьюринга указано немецкое название статьи Гёделя. Английский перевод не публиковался вплоть до 1962 года.

Эти результаты имеют полезные примене-
[231]
ния. В частности, они показывают (§ 11), что гильбертова Entscheidungsproblem не может иметь решения.

Это последнее упоминание имени Гильберта на протяжении последующих 18 страниц статьи.

После того как Тьюринг узнал о доказательстве Алонзо Чёрча и установил эквивалентность двух подходов, ему нужно было добавить к статье приложение. Вместе с ним во введение был добавлен и последний абзац.

В недавней статье Алонзо Чёрча† представлена идея «эффективной вычислимости», которая эквивалентна моей «вычислимости», но определена совсем по-другому. Чёрч также пришел к похожим выводам об Entscheidungsproblem‡. Доказательство эквивалентности «вычислимости» и «эффективной вычислимости» приводится в общих чертах в приложении к данной статье.

† Alonzo Church, «An unsolvable problem of elementary number theory», *American J. of Math.*, 58 (1936), 345–363.

‡ Alonzo Church, «A note on the Entscheidungsproblem», *J. of Symbolic Logic*, 1 (1936), 40–41.

Это последнее упоминание Entscheidungsproblem на следующих почти 28 страницах статьи Тьюринга. Согласно «Оксфордскому словарю английского языка» (2-е издание), этот абзац содержит первое известное употребление слова «вычислимость», отличное от словаря 1889 года. С тех пор было издано свыше 30 книг со словом «вычислимость» в названии; первой была «Вычислимость и неразрешимость» Мартина Дэвиса, изданная McGraw-Hill в 1958 году.

И вот начинается первый из одиннадцати разделов статьи Тьюринга.

> 1. *Вычислительные машины.*
>
> Мы уже сказали, что вычислимые числа – это десятичные числа, вычисляемые конечным способом. Это требует несколько более точного определения. Но вплоть до §9 не будет ни одной попытки обосновать данное определение. Пока же я только скажу, что обоснование опирается на тот факт, что человеческая память ограничена.

Тьюринг уже сказал, что вычислимые числа – те, что могут быть записаны машиной, но теперь в определении он обосновывает «конечный способ» ограничением человеческой памяти. Это поверхностное сопоставление машины и человека характерно для статьи Тьюринга.

Когда Тьюринг сказал вначале, что вычислимое число вычисляется конечным способом, это казалось разумным, но теперь, когда он обосновывает это ограниченностью человеческого разума, он поднимает определенные вопросы о природе математического знания. Мы называем вещественные числа «вещественными», хотя подавляющее их большинство никто никогда не видел. Более того, Тьюринг покажет в статье, что это подавляющее большинство вещественных чисел даже не может быть вычислено с помощью конечных алгоритмов. Каков же смысл существования вещественных чисел? Это философский вопрос, которого Тьюринг лишь вскользь касается в исправлении к своей статье (глава 16 этой книги).

Дальше Тьюринг сравнивает человека и машину с точки зрения дискретных состояний памяти:

> Мы можем уподобить человека, вычисляющего вещественное число, машине, которая может находиться только в конечном числе состояний $q_1, q_2, ..., q_R$, которые назовем «m-конфигурациями».

Буква *m* означает *машинная*. Машина имеет конечное число конфигураций и делает еще что-то в зависимости от ее текущей конфигурации. Более современный термин – состояние, и позже Тьюринг говорит о «состояниях памяти», которые аналогичны этим состояниям машины. Например, простая стиральная машина имеет состояния «загрузка», «стирка», «ополаскивание» и «отжим». Аналогично выполнение ручного деления включает ряд различных умственных конфигураций, или состояний памяти: «теперь нужно умножить», «теперь – вычесть», «теперь – занять десяток». Машина работает, переключаясь между различными конфигурациями, нередко с повторениями.

Машина снабжена «лентой» (подобием бумаги), по которой она движется и которая разделена на части (называемые «клетками»), каждая из которых может содержать «символ».

Тьюринг называет эту ленту «подобием бумаги», потому что бумага – это то, что человек использовал бы для вычисления числа. Лента в машине Тьюринга часто изображается как бумажная, но если бы машина Тьюринга была на самом деле построена, то лента, наверное, была бы магнитной или просто блоком машинной памяти.

Люди обычно используют двумерный лист бумаги, а Тьюринг ограничивает свою машину одномерной лентой, разделенной на клетки. Символами в этих клетках могли бы быть десятичные цифры от 0 до 9, или все буквы алфавита, или 95 символов клавиатуры компьютера. (Как вы увидите, Тьюринг даже позволяет «символу» состоять из нескольких знаков.)

В этом разделе статьи для представления символов Тьюринг использует прописную букву *S* (в значении «символ») готического шрифта, поэтому выглядит она примерно так: $\mathfrak{S}$. Этот шрифт появится еще не раз.

В любой момент только одна клетка, скажем, *r*-я, содержащая символ $\mathfrak{S}(r)$, находится «в машине».

Здесь Тьюринг предполагает, что клетки ленты можно нумеровать для их идентификации. Например, $\mathfrak{S}(3451)$ должно обозначать символ в клетке с номером 3451. Если бы в этой клетке находился символ 'A', то $\mathfrak{S}(3451)$ должно было быть 'A'. Однако, строго говоря, клетки на ленте не нумеруются, и машина не обращается к отдельной клетке по ее номеру. (Иными словами, у клетки нет явного адреса.)

В любой момент времени, говорит Тьюринг, только одна клетка ленты находится «в машине» и может быть проверена ею.

Мы можем назвать эту клетку «текущей клеткой» (scanned squared). Символ в текущей клетке можно назвать «текущим символом». «Текущий символ» – единственный, который машина, так сказать, «непосредственно осознает».

Машина не может «видеть» всю ленту сразу. Она может «смотреть» только на одну клетку.

Несмотря на это, за счет изменения своей *m*-конфигурации машина фактически может помнить некоторые из символов, которые она «видела» раньше.

Машина переходит из одной *m*-конфигурации в другую в зависимости от текущего (scanned) символа. Например, если в некоторой *m*-конфигурации q_{34} текущий символ – «A», то она могла бы перейти в *m*-конфигурацию q_{17}. Если текущий символ – «B», она могла бы перейти в *m*-конфигурацию q_{123}. Таким образом, *m*-конфигурация q_{17} «знает», что последним текущим символом был «A», а *m*-конфигурация q_{123} «знает», что последним текущим символом был «B». (Это не *совсем* так; другие конфигурации тоже могли бы перейти в q_{17} и q_{123}, но, видимо, замысел машины – в том, что q_{17} и q_{123} перед выполнением своей работы знают достаточно из того, что уже произошло.)

Возможное поведение машины в каждый момент определяется *m*-конфигурацией q_n и текущим символом $\mathfrak{C}(r)$. Эту пару q_n, $\mathfrak{C}(r)$ будем называть «конфигурацией»: таким образом, конфигурация определяет возможное поведение машины.

M-конфигурации – это q_1, q_2 и т. д. *M*-конфигурацию в паре с текущим символом Тьюринг называет просто *конфигурацией*.

Тьюринг уже заявлял, что машина переходит от одной *m*-конфигурации к другой в зависимости от текущего символа. Что же еще может машина? Немного:

В некоторых конфигурациях, в которых текущая клетка пуста (то есть не содержит символа), машина записывает в текущую клетку новый символ: в других конфигурациях стирает текущий символ. Машина может также изменить текущую клетку, но только сместившись вправо или влево от нее.

Не думаю, что искажу концепцию Тьюринга, если буду называть устройство чтения и записи символов *головкой* машины. Так же, как в магнитофоне или видеокамере, головка соприкасается с лентой только в одном месте. Головка в машине Тьюринга может считать символ с ленты, стереть его или записать на нее новый символ. Она может также сдвигаться на одну клетку влево или вправо. (Хотя обычно головка неподвижна, а движется лента, лучше считать, что головка движется вдоль ленты.)

Кроме того, каждое из этих действий может изменить *m*-конфигурацию. Одни из записанных на ленте символов будут образовывать [232]
последовательность цифр – десятичное представление вычисляемого вещественного числа. Другие – лишь вспомогательными символами, чтобы «помочь запомнить». Ими будут лишь такие вспомогательные символы, которые подлежат стиранию.

Поскольку Тьюрингу нужно, чтобы его машина вычислила число, машина должна печатать числа (или цифры) а вообще – *бесконечную* последовательность цифр. Чтобы помочь себе в этом, машине может понадобиться часть ленты в качестве временной памяти.

На что похожа машина Тьюринга? Можно, конечно, представить какую-нибудь ужасного вида машину[1], но лучше взглянуть в зеркало. Перефразируя кульминацию известного научно-фантастического фильма[2], «машины Тьюринга – это люди» – только живые люди, выполняющие алгоритм очень ограниченно, но точно.

Я утверждаю, что эти действия составляют все, что необходимо для вычисления числа.

То есть для вычисления *человеком*. Если вы считаете, что этой машине недостает некоторых элементарных арифметических операций вроде сложения и вычитания, вы абсолютно правы. Сложение и вычитание не встраиваются в машину Тьюринга. Наоборот, она может их выполнить, если имеет надлежащие конфигурации.

---

[1] Одна из лучших — заканчивается сигналами, подаваемыми при открытии и закрытии биржи, — сопровождает статью Gregory J. Chaitin, «Computers, Paradoxes and the Foundations of Mathematics», *American Scientist*, Vol. 90 (March–April 2002), 168. См. онлайн-версию: http://www.cs.auckland.ac.nz/CDMTCS/chaitin/amsci.pdf .

[2] Soylent Green (1973).

Отстаивать это утверждение будет проще, когда читатель познакомится с теорией машин. Поэтому в следующем разделе я продолжу развитие теории, и, полагаю, станет понятно, что значит «машина», «лента», «текущий» и т. д.

Возможно, мы уже готовы приступить к рассмотрению некоторых настоящих машин, но Тьюринг пока не желает тешить наше любопытство. Сначала он хочет ввести некоторые определения.

2. *Определения.*

Автоматические машины.

Если на каждом шаге поведение машины (в смысле §1) *полностью* определяется конфигурацией, мы назовем машину «автоматической машиной» (или *a*-машиной).

Для некоторых целей мы можем использовать машины (машины выбора или *c*-машины), чье поведение определяется конфигурацией лишь отчасти (отсюда использование слова «возможное» в §1).

Для описания зависимости поведения машины от конфигурации (страница 91) Тьюринг пользуется выражением «возможное поведение машины». Поведение следовало бы уточнить, так как в некоторых машинах оно может слегка меняться под воздействием человека – внешнего «оператора» машины:

Когда такая машина достигает одной из таких неоднозначных конфигураций, она не может двигаться дальше, пока внешний оператор не сделает некоторый произвольный выбор. Так было бы, если бы мы пользовались машинами, имеющими дело с аксиоматическими системами. В этой статье я имею дело только с автоматическими машинами и поэтому часто буду опускать префикс «*a*-».

Разделение Тьюрингом автоматических машин и машин выбора несколько напоминает традиционное разделение программирования на пакетную обработку и интерактивное вычисление. Сегодня наша работа с вычислительной техникой настолько интерактивна, что мы забываем о множестве компьютерных программ, которые работают постоянно, не отвечая на каждое нажатие клавиши и щелчок мыши пользователя.

Хотя машины выбора могут представлять интерес, они играют почти незаметную роль в статье Тьюринга. Поведение автоматических

машин в статье Тьюринга будет полностью определяться конфигурациями машин.

Вычислительные машины.

Если *a*-машина печатает два вида символов, первые из которых (называемые цифрами) состоят только из 0 и 1 (остальные называются символами второго вида), то машина будет называться вычислительной машиной.

Еще не приступив к демонстрации типовых машин, Тьюринг решил ограничить их печатью цифр 0 и 1 – двух цифр, необходимых для представления двоичных чисел[1]. Использование двоичных чисел было разумным шагом, однако большинству читателей 1937 года он, наверное, не был столь очевиден, как нам сегодня. Клод Э. Шеннон (1916–2001), чья магистерская диссертация *Символьный анализ реле и переключающих цепей*, написанная в 1937 году в MIT, доказала эквивалентность между цепями и булевой алгеброй, безусловно, оценил бы такой выбор. Однако двоичные числа использовались в первых компьютерах явно не всеми: если Цузе использовал двоичные числа, то машины Эйкена и Штибица были десятичными. ENIAC (1943–1945) тоже был десятичным компьютером. Слово «бит» («bit» – сокращение от «binary digit») не встречалось в печати вплоть до более поздней статьи Шеннона 1948 года[2].

Тьюринг не пытается настаивать на использовании двоичных чисел в своих машинах. Их преимущество действительно становится очевидным только на странице 245 его статьи (страница 182 этой книги), но чтобы отбросить все сомнения, в следующей главе я проведу сравнение двух простых машин – двоичной и десятичной.

---

[1] Краткий обзор двоичных чисел может быть найден в Charles Petzold, *Code: The Hidden Language of Compute Hardware and Software* (Microsoft Press, 1999).

[2] Claude E. Shannon, «A Mathematical Theory of Communication», *The Bell System Technical Journal*, Vol. 27 (July, October 1948). Шеннон приписывает выдумку этого слова американскому математику Дж. В. Тьюки. Слово получило неважный отзыв от Ланселота Хогбена в *The Vocabulary of Science* (Stein and Day, 1970), 146: «Введение Тьюки слова *бит* для двоичных цифр – это не что иное, как безответственная вульгарность, чтобы быть заслуживающей похвалы». Не согласен: биты настолько распространены в нашей нынешней жизни, что крошечное слово идеально, как вещи сами по себе.

Если машина снабжена пустой лентой и приведена в движение, начиная с правильной начальной *m*-конфигурации, то подпоследовательность печатаемых ею символов первого вида будем называть *последовательностью, вычисляемой машиной*.

Машина приводится в движение при пустой ленте. Машина печатает цифры 0 и 1 (символы первого вида) и другие символы (второго вида). Цифры 0 и 1 образуют вычисляемую последовательность. Тьюринг проводит различие между вычисляемой *последовательностью* и вычисляемым *числом*.

Вещественное число, чье представление в виде двоичной десятичной дроби получается путем предварения этой последовательности десятичной запятой, называется *числом, вычисляемым машиной*.

(Наверное, мы тоже не исказим замысел Тьюринга, если будем считать, что вычисляемая машиной последовательность цифр вещественного числа, согласно нашим математическим традициям, предваряется, помимо запятой, еще и цифрой 0. – *прим. перев.*)

Это предложение читается с некоторым трудом из-за не вполне правильной терминологии. Тем не менее мы должны простить Тьюринга за эту путаницу, потому что в то время люди просто не были готовы к обсуждению двоичных чисел. Даже сегодня тот, кто свободно обращается с двоичными числами, часто не вполне владеет двоичными дробями. Не спасает даже научный режим калькулятора Windows: приводя число к двоичному виду, он просто отбрасывает дробную часть.

Слово «десятичный» предполагает числа по основанию десять. Вот несколько десятичных дробей:

0,25
0,5
0,75

Десятичная запятая отделяет целую часть (если она есть) от дробной части.

Те же три значения представляются в двоичном виде так:

0,01
0,1
0,11

Но эта запятая не десятичная. На самом деле ее нужно называть *двоичной запятой*.

Если отдельные цифры двоичных целых чисел соответствуют степеням 2, то цифры двоичных дробей соответствуют отрицательным степеням 2:

- 0,1 – это двоичный эквивалент 2^{-1} или десятичной дроби 1/2;
- 0,01 – эквивалент 2^{-2} или 1/4;
- 0,001 – 2^{-3} или 1/8;
- 0,0001 – 2^{-4} или 1/16;
- 0,00001 – 2^{-5} или 1/32

и т. д. Двоичное число 0,10101 – это

$$1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3} + 0 \cdot 2^{-4} + 1 \cdot 2^{-5},$$

или, наверное, более наглядно

$$\frac{1}{2}+\frac{0}{4}+\frac{1}{8}+\frac{0}{16}+\frac{1}{32}.$$

Десятичный эквивалент этого двоичного числа – 21/32 или 0,65625. Многие двоичные дроби, как и десятичные, имеют повторяющиеся группы цифр. Так, 1/3 в двоичном представлении:

0,01010101...

А вот – 2/3:

0,10101010...

Аналогично:

1/5 – 0,001100110011...
2/5 – 0,011001100110...
3/5 – 0,100110011001...
4/5 – 0,110011001100...

Утверждение Тьюринга правильнее звучит так: «вещественное число, чье представление в виде двоичной дроби получается путем предварения этой последовательности двоичной запятой, называется числом, вычисляемым машиной».

Или давайте скажем даже так: «вычисляемое машиной число – это двоичная дробь, полученная приписыванием к этой последовательности двоичной запятой».

Например, если некая вычислительная машина Тьюринга печатает только 0 и 1, то «вычисляемая машиной последовательность» будет:

01

«Вычисляемое машиной число» получается приписыванием к этой последовательности двоичной запятой (и цифры 0 – *прим. перев.*):

0,01

Это – двоичный эквивалент 1/4.

Поскольку всегда предполагается, что запятая в двоичном числе предшествует вычисляемой последовательности, машины Тьюринга вычисляют только двоичные числа от 0 до 1, но столь узкий диапазон прекрасно подходит для проникновения в перечислимость, которая может понадобиться нам.

Будем считать, что на каждом шаге работы машины номер текущей клетки, полная последовательность всех символов на ленте и *m*-конфигурация описывают *полную конфигурацию* на этом шаге.

При обсуждении своей машины Тьюринг употребляет слово «конфигурация» в трех значениях, и будет неплохо освежить их в памяти:

- *m-конфигурация* – это одно из состояний машины;
- *конфигурация* – это сочетание *m*-конфигурации и текущего символа;
- *полная конфигурация* – это, по сути, «снимок» всей ленты в некоторый момент времени плюс текущая *m*-конфигурация и положение головки.

Изменения машины и ленты между последовательными полными конфигурациями назовем *ходами* (*moves*) машины.

Следующие два определения не используются до некоторого места позже в статье:

[233]

Циклические и ациклические машины.

Если вычислительная машина никогда не записывает более чем конечное число символов первого вида, назовем её *циклической* (*circular*). В противном случае говорят, что она *ациклическая* (*cycle-free*).

Машина будет циклической, если она достигнет конфигурации, из которой нет хода, или если она продолжает движение и, возможно, печатает символы второго вида, но больше не может печатать символы первого вида. Значение слова «циклический» будет объяснено в §8.

Чуть раньше я привел пример машины, которая печатает 0 и 1 и больше ничего. Число символов здесь конечно, а это подпадает под определение *циклической* машины Тьюринга. Машина где-то застревает и не может больше печатать цифры. Это плохо. Тьюрингу нужно, чтобы его машины продолжали печатать цифры *всегда*.

Ациклические машины – это хорошие машины. Машина, которая печатает только 0 и 1 и больше ничего, – не ациклическая. Если машине действительно нужно вычислить двоичный эквивалент 1/4, она должна напечатать 0 и 1, а затем печать цифру 0 без конца.

Хотя Тьюринг не говорит об этом, он, похоже, подразумевает, что его вычислительные машины печатают цифры слева направо, точно так же, как мы читаем цифры после запятой в двоичном числе.

Вычислимые последовательности и числа.

Говорят, что последовательность вычислима, если она может быть вычислена ациклической машиной. Число вычислимо, если оно отличается на целое число от числа, вычисляемого ациклической машиной.

Тьюринг делает различие между последовательностями и числами. Вычислимая последовательность:

010000...

Соответствующее вычислимое число:

0,010000...

Число

1,010000...

тоже считается вычислимым, потому что оно отличается на целое число от числа, вычисляемого машиной. Так же, как

10,01000...

и отрицательные числа.

Мы избежим путаницы, говоря чаще о вычислимых последовательностях, чем о вычислимых числах.

Глава 5

Машины в работе

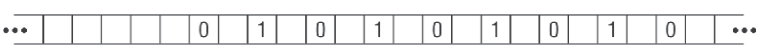
Тьюринг несомненно понимал, что введение воображаемой вычислительной машины в математическую статью было и нестандартным, и смелым. Как хороший математик он дал определение и формальное описание этих машин. Ему не нужно было приводить какие-либо примеры, но, я полагаю, он понимал, что одна лишь абстракция не устроит его читателей. Им нужно что-то конкретное. И он идет навстречу этому желанию.

3. Примеры вычислительных машин.

I. Можно создать машину для вычисления последовательности 010101... .

Машина печатает ленту, которая выглядит следующим образом:

Правильно, но неточно. Как позже объяснит Тьюринг, он предпочитает, чтобы для печати числовых последовательностей его машины использовали только чередующиеся клетки. На самом деле машина из первого примера будет печатать ленту, подобную такой:



Для обозначения *m*-конфигураций своей машины Тьюринг использует строчные буквы немецкого готического шрифта. Возможно, к ним надо будет привыкнуть, поэтому я буду обращать внимание на потенциально трудные знаки. Символы, которые Тьюринг использует для его первой машины, – 𝔟, 𝔠, 𝔨 и 𝔢. (Внимание: немецкая буква k похожа на f.)

Машина должна иметь четыре *m*-конфигурации «𝔟», «𝔠», «𝔢», «𝔢» и уметь печатать «0» и «1». Поведение машины описывается в следующей таблице, в которой «*R*» означает «машина перемещается так, чтобы просмотреть клетку непосредственно справа от текущей». Аналогично – для «*L*». «*E*» означает «текущий символ стирается», а «*P*» – «печать».

В этих таблицах за *P* всегда следует определенный печатаемый символ. Например, *P*0 означает печать 0, *P*1 – печать 1, а *Px* – печать *x*.

Эту таблицу (и все последующие таблицы того же вида) следует понимать так: в конфигурации, описанной в первых двух столбцах, последовательно выполняются операции из третьего столбца, а затем машина переходит в *m*-конфигурацию, описанную в последнем столбце.

В таблице – четыре столбца, разделенные на две пары:

Конфигурация		*Поведение*	
m-конфиг.	*символ*	*операции*	*заключ. m-конфиг.*

То, что делает машина, зависит от конфигурации, которая состоит из *m*-конфигурации и символа текущей клетки. Третий столбец содержит операции (которыми могут быть только *P*, *E*, *L* и *R*), а четвертый – следующую *m*-конфигурацию.

Обычно во втором столбце явно указывается конкретный текущий символ, например 0 или 1, но Тьюринг использует еще слово «Any», которым обозначается любой символ, и слово «None», чтобы обозначить отсутствие символа, то есть пустую клетку. (Это может несколько смутить современных программистов, которые привыкли считать пробел таким же символом, как все остальные. Под словом «Any» Тьюринг, как правило, понимает «любой непустой» символ.) Произвольный символ, *включая* пустую клетку, обрабатывается следующим способом:

Если второй столбец оставлен пустым, это значит, что поведение, заданное третьим и четвертым столбцами, относится к любому символу и к отсутствию символа.

К счастью, потенциальная неоднозначность минимальна.

Машина начинает с *m*-конфигурации 𝔟 при пустой ленте.

Машины Тьюринга всегда начинают с *m*-конфигурации 𝔟 (от *begin* – начать, или скорее 𝔟𝔢𝔤𝔦𝔫). Вот долгожданная машина:

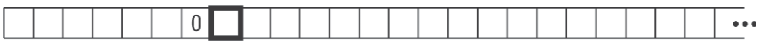
Конфигурация		*Поведение*	
m-конфиг.	*символ*	*операции*	*заключ. m-конфиг.*
𝔟	None	*P*0, *R*	𝔠
𝔠	None	*R*	𝔢
𝔢	None	*P*1, *R*	𝔣
𝔣	None	*R*	𝔟

Эти строки могут читаться примерно так: «В *m*-конфигурации 𝔟, когда текущая клетка – пустая (None), напечатать 0, переместить головку вправо и сменить *m*-конфигурацию на 𝔠».

Давайте запустим эту машину и понаблюдаем за ее работой. Начнем с *m*-конфигурации 𝔟 при пустой ленте. Хотя лента теоретически бесконечна в обе стороны, для машин, которые Тьюринг описывает в этой статье, нужна только лента, бесконечно уходящая вправо, потому что именно там печатаются цифры вычисляемой последовательности:

Положение на ленте головки чтения-записи можно обозначить поразному. Я выбрал жирную рамку вокруг текущей клетки. Изначально головка может находиться на ленте где угодно:

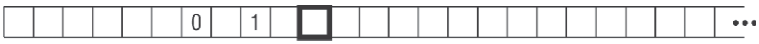
В этой клетке ничего нет. Таблица говорит нам, что в *m*-конфигурации 𝔟 и в отсутствие символа нужно напечатать 0 и сдвинуться направо:



Новой *m*-конфигурацией становится 𝔠. Если в клетке пусто, сдвигаемся вправо и попадаем в *m*-конфигурацию 𝔢:

Если в *m*-конфигурации $\mathfrak{e}$ в клетке пусто, печатаем в ней 1 и перемещаемся вправо:

Теперь машина находится в *m*-конфигурации $\mathfrak{f}$ и должна переместиться вправо:



Теперь машина вернулась в исходную *m*-конфигурацию $\mathfrak{b}$, и цикл начинается снова. Таким образом, машина печатает бесконечную последовательность пар символов 01. Заманчиво считать каждую из четырех строк таблицы *командой* (*instruction*), и действительно, позже Тьюринг применяет этот термин. Но учтите, что эти строки не *команды* для машины, а ее *описание*. Именно поэтому лучше использовать термин *состояние*. Если мы считаем эти строки командами, то подразумеваем, что стоит заменить их на что-то другое, и та же самая машина работала бы по-другому, но это будет означать, что машина исполняет данные команды, а это совсем не так. (Во всяком случае, пока.) Эта машина выполняет определенную задачу. Не имеет значения, как на самом деле работает машина; важно то, что мы можем описать работу машины стандартным способом, основанным на конфигурациях, символах и операциях.

Можно ли построить такую машину? Данную машину можно построить по-разному. Она могла бы иметь вращающееся колесо с краской и резиновыми штампиками по окружности, которое печатает поочередно нули и единицы. Построение машины Тьюринга, которая работает так, как описано, – машины, которая действительно читает символы и интерпретирует их, – потребует, наверное, более сложной компьютерной логики внутри, чем машина выглядит внешне. Обычно машины Тьюринга «строятся» путем компьютерного моделирования.

Машины Тьюринга переходят от одной *m*-конфигурации к другой в зависимости от текущего символа. Этот «условный переход» (известный в информатике) – то, что первые вычислительные машины той поры делали плохо. Конрад Цузе кодировал команды своей машины, пробивая дырки в старых 35-миллиметровых кинопленках. В его первой машине Z1 команды выполнялись последовательно. Машина Z3 могла делать переходы, но условные переходы были неуклюжими. Только когда программы стали храниться в памяти компьютеров

(«компьютер с хранимой программой»), переходы стали простыми и обычными.

В столбце *символ* этой частной таблицы – всюду «None», означающий, что конфигурация применима только к пустой клетке. Если бы этой машине пришлось прочитать клетку, в которой действительно есть символ, она не знала бы, что делать. Она могла бы остановиться. Могла бы сломаться. Могла вспыхнуть. Или отформатировать жесткий диск. Мы этого не знаем. Но что бы ни случилось, такую машину нельзя считать «ациклической» машиной. Однако пока данная машина начинает с пустой ленты, проблем не будет.

Поскольку Тьюринг определил машину для печати последовательности

01010101...

он показал, что это – вычислимая последовательность. Эту последовательность можно преобразовать в вычислимое число, дописав перед ней нуль и запятую (в оригинале десятичную точку, как принято в англоязычных странах – *прим. перев.*):

0,01010101...

Теперь ясно, что машина вычисляет двоичный эквивалент рационального числа 1/3. Поменяв местами 0 и 1, машина вычислила бы двоичное число

0,10101010...

или 2/3.

Позвольте показать вам машину, которая вычисляет число 1/4, двоичное представление которого:

0,01000000...

Эта машина отвечает соглашениям Тьюринга и использует готические буквы b, c, d, e и f для *m*-конфигураций:

Конфигурация		*Поведение*	
m-конфиг.	*символ*	*операции*	*заключ. m-конфиг.*
𝔟	None	*P*0, *R*	𝔠
𝔠	None	*R*	𝔡
𝔡	None	*P*1, *R*	𝔢
𝔢	None	*R*	𝔣
𝔣	None	*R*	𝔢

Обратите особое внимание на последние *m*-конфигурации 𝔢 и 𝔣. Они чередуются таким образом, чтобы машина заканчивала печать бесконечным рядом нулей. Бесконечная печать нулей необходима, чтобы машина соответствовала определению Тьюринга машины «ациклическая».

Должно быть ясно, что подобные вычислительные машины можно определить для вычисления *любого рационального числа*. Рациональные числа здесь не являются проблемой.

Ранее (во втором параграфе раздела 1) Тьюринг сказал: «машина может также изменить текущую клетку, но только сместившись вправо или влево от нее». Теперь ему слегка не хватает гибкости.

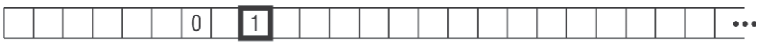
[234]

Если (вопреки описанию в §1) мы позволим символам *L*, *R* появляться в столбце операций несколько раз, мы можем значительно упростить таблицу.

m-конфиг.	*символ*	*операции*	*заключ. m-конфиг.*
𝔟	None	*P*0	𝔟
	0	*R*, *R*, *P*1	𝔟
	1	*R*, *R*, *P*0	𝔟

(Вскоре Тьюринг позволит конфигурации иметь также несколько операций *P*.) Теперь в таблице – лишь одна *m*-конфигурация, и все зависит от текущего символа. Если текущая клетка пуста (что бывает, когда машина только запущена), то машина просто печатает 0:

Головка не двигается. Машина остается в той же самой *m*-конфигурации, но теперь текущий символ – 0. Машина перемещается на две клетки вправо и печатает 1:



Теперь текущий символ – 1, поэтому машина перемещается на две клетки вправо и печатает 0:

И снова эта машина печатает 0 и 1 в чередующихся клетках.

Важный урок состоит в том, что каждую заданную последовательность можно вычислить разными машинами. Однако заданная автоматическая машина, запущенная с пустой лентой, всегда вычисляет одну и ту же последовательность. (Я говорю здесь именно об *автоматической* машине, потому что машины выбора допускают вмешательство человека-оператора и могут таким образом создавать другие последовательности, но Тьюринг почти не обсуждает машины выбора в своей статье.) Ни в одну из автоматических машин Тьюринга невозможно привнести какую-то неопределенность, беспорядок или информацию (вроде даты и времени, долготы и широты или веб-страницы) *«извне»*.

Многократное применение операций *L*, *R* и *P* в одной конфигурации может значительно упростить машины, но следует иметь в виду, что эти упрощенные таблицы всегда можно вернуть к строгой форме, допускающей для каждого состояния лишь одну из операций *L*, *R* или *P*. Сейчас это различие может показаться несущественным, но позже станет важным.

II. В качестве несколько более сложного примера мы можем создать машину для вычисления последовательности 001011011101111011111....

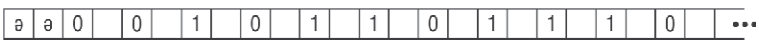
Несколько более сложного? Смотрите, что предлагает здесь Тьюринг. Последовательность из удлиняющихся цепочек единиц, разделяемых нулями. Сначала – одна единица, потом – две, потом – три и т. д. Очевидно, Тьюрингу уже надоело вычислять рациональные числа. То, чем он теперь хочет заняться, – иррациональное число, и очень похоже, что оно еще и трансцендентное.

Печатая группу единиц, эта новая машина должна как-то «помнить», сколько единиц она напечатала в предыдущей группе, а затем напечатать еще одну. Двигаясь назад и вперед, машина всегда имеет доступ к предыдущей группе, поэтому она может использовать эту информацию для построения следующей группы. Интересно изучить стратегию Тьюринга для достижения данной цели.

Тьюринг снова использует строчные готические буквы для *m*-конфигураций, в данном случае – *o, q, p, f* и *b*.

Машина может иметь пять *m*-конфигураций, т. е. «𝔬», «𝔮», «𝔭», «𝔣», «𝔟», и печатать «ә», «*x*», «0», «1». Первыми тремя символами на ленте будут «әә0»; остальные цифры будут следовать через клетку.

Здесь Тьюринг впервые говорит о печати цифр (нулей и единиц, или символов первого вида) в чередующихся клетках. Считая, что самые левые символы появляются на ленте слева, он предлагает, что лента заканчивается примерно так:



Конечно, лента никогда не «заканчивается» чем-то, потому что машина работает вечно. Она должна печатать без конца, чтобы считаться «ациклической» машиной.

Символ ə известен в фонетике и лингвистике как *schwa* (нейтральный гласный – *прим. перев.*). Тьюринг использует его в качестве того, что программисты называют *ограничителем* (стражем, меткой – *прим. перев.*). Это специальный знак, который в данном случае обозначает границу числа. Машина может двигать головку влево до самого начала ленты, пока в текущей клетке нет ə. (Почему же здесь два ə? В этом примере нужен только один, но Тьюринг позже создает машину, которая требует в качестве два ə. Наверное, он добавил второй ə в этом примере только для согласованности.)

В машине первого примера пустые клетки между цифрами 0 и 1 были не нужны. Здесь же они будут играть важную роль.

В промежуточных клетках мы никогда не печатаем ничего, кроме «*x*». Эти символы служат нам для «сохранения места» и стираются, когда мы заканчиваем работу с ними.

Тьюринг разделяет клетки ленты на две категории. Машина печатает нули и единицы в каждой второй клетке. В этих клетках никаких других символов, кроме ограничителя, не появляется. Тьюринг использует промежуточные клетки как своего рода временную рабочую область памяти. Таким образом, мы можем говорить о «числовых клетках», содержащих нули и единицы, и «нечисловых клетках», содержащих прочие символы. (Позже Тьюринг назовет их *F*-клетками для цифр (*Figures*) и *E*-клетками для удаляемых (*Erasable*) символов.)

Мы также условимся, что в последовательности цифр в чередующихся клетках не будет пустот.

Вычисляя шаг за шагом нули и единицы, машина печатает их последовательно слева направо. Каждый новый символ, который вы-

числяет машина, печатается в следующей доступной пустой числовой клетке. Ни одна числовая клетка не пропускается. Эти ограничения – свод правил (явных и подразумеваемых), которые Эмиль Пост позже назвал «машиной согласно Тьюрингу»[1] – немного более ограниченной, чем обобщенная «машина Тьюринга». Машина согласно Тьюрингу никогда не стирает числовую клетку и не переписывает существующую цифру в числовой клетке на другую. Позже эти неявные правила становятся важными.

Вот таблица машины Тьюринга для вычисления иррационального числа, которое он определил:

Конфигурация		*Поведение*	
m-конфиг.	*символ*	*операции*	*заключ. m-конфиг.*
б		*P*ə, *R*, *P*ə, *R*, *P*0, *R*, *R*, *P*0, *L*, *L*	о
о	1	*R*, *Px*, *L*, *L*, *L*	о
	0		ч
ч	Any (0 or 1)	*R*, *R*	ч
	None	*P*1, *L*	п
п	*x*	*E*, *R*	ч
	ə	*R*	ф
	None	*L*, *L*	п
ф	Any	*R*, *R*	ф
	None	*P*0, *L*, *L*	о

Как обычно, машина начинается в *m*-конфигурации б. Она печатает два ə и два нуля. Лента выглядит следующим образом:

m-конфигурация б выполняет функцию, которую программист мог бы назвать инициализацией. Машина больше никогда не попадает в *m*-конфигурацию б.

---

[1] В приложении статьи Emil Post, «Recursive Unsolvability of a Problem of Thue», *The Journal of Symbolic Logic*, Vol. 12, No. 1 (Mar. 1947), 1–11. Вся статья перепечатывается в Martin Davis, ed., *The Undecidable* (Raven Press, 1965), 293–303. Приложение перепечатывается в B. Jack Copeland, ed., *The Essential Turing* (Oxford University Press, 2004), 97–101.

Прежде чем мы получим все сведения о внутренностях этой машины, давайте получим общее представление о других *m*-конфигурациях. В нескольких конфигурациях (а именно 𝔮, 𝔭 и 𝔣̃) в столбце *операции* есть перемещение сразу на две клетки: *R*, *R* или *L*, *L*. В этих случаях машина фактически пропускает числовые клетки (для 𝔮 и 𝔣̃) или нечисловые клетки (для 𝔭).

Все *m*-конфигурации, кроме 𝔟, также замыкаются на себя в зависимости от текущего символа. Программисты часто называют такую операцию *циклом* (*loop*). Циклы выполняют повторяющиеся действия, даже такие простые, как поиск заданного символа.

m-конфигурация 𝔬 перемещается справа налево по группе единиц в числовых клетках. Справа от каждой найденной единицы она печатает *x*, а затем идет влево, чтобы проверить следующую числовую клетку. Когда она завершается, она переключается на *m*-конфигурацию 𝔮.

m-конфигурация 𝔮 двигается по числовым клеткам слева направо, пока не встретит пустую. Это – конец текущей последовательности. Затем она печатает 1, сдвигается влево (к нечисловой клетке) и переключается на 𝔭. Точно так же *m*-конфигурация 𝔣̃ двигается вправо по числовым клеткам, пока не встретит пустую. Затем она печатает 0, сдвигается на 2 клетки влево и переключается на 𝔬.

m-конфигурация 𝔭 – своего рода диспетчер. Она проводит большую часть своего времени, двигаясь влево по нечисловым клеткам в поиске символов *x*. Найдя *x*, она стирает его, сдвигается вправо и переключается на 𝔮. Если она достигает ограничителя, то сдвигается вправо и переключается на 𝔣̃.

Тьюринг использует символы *x* очень хитрым способом. Построение новой группы единиц машина начинает с печати *x* после каждой 1 в предыдущей группе. Машина печатает 1 в конце существующей последовательности, а затем печатает по одной 1 для каждого x, удлиняя тем самым группу единицу.

Хотя можно проиллюстрировать, как выглядит лента после каждой операции, для этого примера было бы лучше рассмотреть ленту по завершении каждой конфигурации.

От *m*-конфигурации 𝔟 машина переходит к 𝔬, но для текущего символа 0 она ничего не делает и тут же переходит прямо к *m*-конфигурации 𝔮. Если в *m*-конфигурации 𝔮 текущий символ – 0 или 1, головка перемещается вправо на две клетки и остается в той же самой *m*-конфигурации. Но если она встречает пустую клетку, то печатает 1 и перемещается влево. В общем, *m*-конфигурация 𝔮 перемещается

вправо по числовым клеткам, пока не встретит пустую. Тогда она печатает 1 и перемещается влево.

Следующая *m*-конфигурация – ɒ, которая проходит по всем нечисловым клеткам. Каждый раз она сдвигается влево на две клетки, пока не встретит нечисловую клетку с символом *x* или ə. В данном случае им будет ə. Она сдвигается вправо:

m-конфигурация f̄ перемещает головку по числовым клеткам. Каждый раз она сдвигается вправо на две клетки вплоть до пустой. Затем она печатает 0 и сдвигается на две клетки влево:

Это тот самый 0, который печатается между группами из единиц.

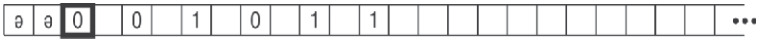
Теперь мы находимся в *m*-конфигурации ɔ. Эта *m*-конфигурация всегда начинает с самой правой единицы группы. Ее задача – напечатать *x* после каждой единицы. Она завершается на цифре 0 слева от группы единиц:

Вернулись в *m*-конфигурацию q. Она движется вправо по числовым клеткам, пока не встретит пустую. Затем она печатает 1 и сдвигается влево.

m-конфигурация ɒ движется влево по нечисловым клеткам, пока не встретит *x* или ə. Если встретился *x*, она стирает его и перемещается вправо:

Снова вернулись в *m*-конфигурацию q. Движемся вправо по числовым клеткам до пустой, печатаем 1 и сдвигаемся влево:

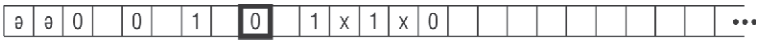
Теперь мы – в *m*-конфигурации ƥ. Движемся влево по нечисловым клеткам, пока не встретим ə. Затем делаем шаг вправо:



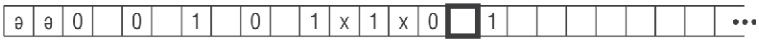
Теперь из *m*-конфигурации ƒ движемся вправо по числовым клеткам, пока не найдем пустую. Затем печатаем 0 и сдвигаемся на две клетки влево:

Кажется, это работает. Теперь у нас – группа из одной единицы и группа из двух единиц. Посмотрим, получим ли мы то, что нужно, если продолжим дальше.

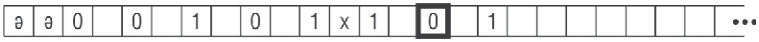
Задача *m*-конфигурации ɔ – напечатать *x* после каждой единицы в последней группе единиц.



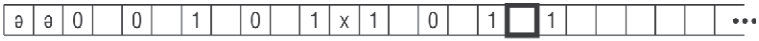
m-конфигурация ɑ перемещается вправо по числовым клеткам, пока не увидит пустую. Тогда она печатает 1 и сдвигается влево:



Обратите внимание, что в этой группе – два символа *x* и две единицы. Для каждого удаляемого *x* будет печататься новая единица. *m*-конфигурация ƥ движется влево по нечисловым клеткам, пока не найдет *x*. Она стирает его и сдвигается вправо:

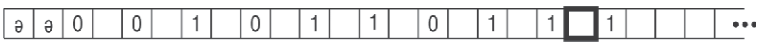


m-конфигурация ɑ движется вправо по числовым клеткам, пока не находит пустую. Затем она печатает 1 и сдвигается влево:

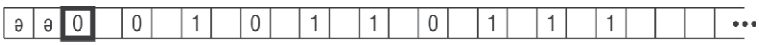


Вернулись в *m*-конфигурацию ꝑ, которая движется влево, пока не наткнется на *x*. Она стирает его и сдвигается вправо.

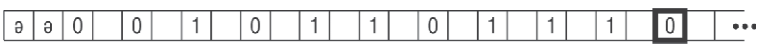
m-конфигурация q печатает еще одну 1 в конце:



Теперь *m*-конфигурация ꝑ движется влево, пока не встретит ə:



m-конфигурация f̄ движется вправо по числовым клеткам, пока не доберется до конца, и печатает 0:



Теперь машина успешно напечатала группу из трех единиц и еще один 0.

Как Тьюринг пришел к методу, применяемому этой машиной? Я подозреваю, что он пытался вычислять последовательность вручную, сопротивляясь искушению использовать числа. Наверное, он отслеживал группы единиц, используя небольшие галочки над цифрами. Эти галочки стали символами *x*, которые машина печатает в нечисловых клетках.

Изображений ленты в статье Тьюринга нет. Он не заинтересован в таком слишком «реалистичном» наглядном представлении машины или ее действий. Напротив, его цель в другом – способ записи поведения машины.

В разделе 2 своей статьи (страница 97 этой книги) Тьюринг говорит: «Будем считать, что на каждом шаге работы машины номер текущей клетки, полная последовательность всех символов на ленте и *m*-конфигурация описывают *полную конфигурацию* на этом шаге». Хотя слова Тьюринга «номер текущей клетки» кажутся несколько странными, так как клетки явно не нумеруются, бесконечная только в одну сторону лента имеет неявную нумерацию.

Тьюринг собирается показать метод для записи поведения машины, используя эти самые полные конфигурации, то есть снимки ленты вместе с текущей *m*-конфигурацией и текущей клеткой.

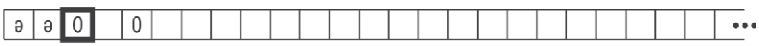
Для иллюстрации работы этой машины ниже приводится таблица нескольких первых полных конфигураций. Эти полные конфигурации изображаются путем выписывания последовательности [235]
символов, которые располагаются на ленте, вместе с *m*-конфигурацией, записываемой под текущим символом. Последовательные полные конфигурации отделяются двоеточиями.

То, что идет в статье дальше, представляет собой четыре элемента «таблицы», каждый из двух строк, на первый взгляд больше похожие на абсолютную тарабарщину. Вот первые из этих четырех элементов:

```
 :ə ə 0   0 :ə ə 0   0 :ə ə 0   0 :ə ə 0   0   :ə ə 0   0  1 :
ɓ    ɔ        q             q             q            ƥ
```

Обратите внимание на двоеточия! Между каждой парой двоеточий – очередной снимок ленты. Некоторые промежутки между нулями и (дальше) между нулями и единицами чуть больше обычного. Этот увеличенный промежуток представляет собой пустую клетку. Вместе с *m*-конфигурациями под лентой они образуют первые шесть полных конфигураций этой машины, показывая все напечатанные на ленте символы на текущий момент.

Первая ɓ обозначает начальную конфигурацию. Изначально лента пуста. Эта конфигурация печатает последовательность между первыми двумя двоеточиями, уже приведенную ранее:



Чтобы указать положение головки на следующем текущем символе, Тьюринг вместо жирного квадрата помещает под следующим текущим символом следующую *m*-конфигурацию:

```
: ə  ə  0   0 :
       ɔ
```

Если текущий символ – 0, *m*-конфигурация ɔ ничего не делает, поэтому следующий снимок ленты такой же, только теперь *m*-конфигурацией является q:

```
: ə  ə  0   0 :
       q
```

Если *m*-конфигурация q видит 0, головка смещается на две клетки вправо, и следующая конфигурация – тоже q:

```
: ə ə 0  0 :
       q
```

Текущая клетка – снова 0. Головка перемещается на две клетки вправо, но *m*-конфигурация – все еще q:

```
: ə ə 0  0    :
         q
```

Заметьте, как удлиняется лента, когда головка уходит за последний напечатанный символ. Теперь текущая клетка – пустая, поэтому машина печатает 1, перемещается на одну клетку влево и переходит в конфигурацию p:

```
: ə ə 0  0  1:
        p
```

Несмотря на меньшую, чем у ленты, наглядность, система обозначений Тьюринга более информативна, особенно тем, что указывает очередную *m*-конфигурацию в текущем положении головки. Такая последовательность полных конфигураций показывает полную историю действий машины. Достаточно взглянуть на любую из этих полных конфигураций, чтобы сопоставить состоянию машины *m*-конфигурацию и текущий символ и представить следующую полную конфигурацию.

Следующая приводимая Тьюрингом последовательность показывает *m*-конфигурацию p, которая ищет слева символ ə, а затем переключается на конфигурацию f, которая ищет пустую клетку справа:

```
ə ə 0  0  1 : ə ə 0  0  1 : ə ə 0  0  1 : ə ə 0  0  1 :
    p          p             f             f
```

Следующий элемент: оставаясь в *m*-конфигурации f, машина ищет пустую числовую клетку (заметьте, как опять увеличивается расстояние между двоеточиями), печатает 0, сдвигается на две клетки влево и переключается на конфигурацию o.

```
ə ə 0  0  1 : ə ə 0  0  1  : ə ə 0  0  1  0 :
        f             f           o
```

Заметив 1, *m*-конфигурация ɒ делает шаг вправо, печатает x и затем сдвигается на три клетки влево:

ə ə 0 0 1 *x* 0 : ... .
ɒ

Вот и всё, что показывает Тьюринг, но если такое представление истории ленты недостаточно лаконично для вас, Тьюринг предлагает альтернативу:

Эту таблицу можно было бы также записать в виде

ɓ : ə ə ɒ 0 0 : ə ə q 0 0 : ... , (C)

где левее текущего символа оставлен промежуток и в него вписана *m*-конфигурация.

Тьюринг пометил этот формат символом C (от «Configuration»). Он будет ссылаться на него в разделе 6. Приведенная ранее полная конфигурация

: ə ə 0 0:
ɒ

теперь становится такой:

: ə ə ɒ 0 0:

Вот и понятна хотя бы одна причина, почему Тьюринг использовал готические буквы для обозначения *m*-конфигураций: в подобном формате не так-то просто отличить *m*-конфигурации от печатаемых машиной символов. Последовательность символов между каждой парой двоеточий больше не является точной копией ленты, потому что требуется дополнительное место для очередной *m*-конфигурации. Даже Тьюринг признает, что это немного неудобно.

Эту форму не просто соблюдать, но позже мы будем использовать ее в теоретических целях.

На самом деле, в еще более измененном виде, она станет *основной*. Тьюринг уже готовится к главному представлению: он представит Универсальную вычислительную машину – называемую сегодня всеми Универсальной машиной Тьюринга, или УМТ (UTM), – которая является функциональным (если не полностью техническим) эквивалентом современного компьютера.

Постарайтесь понять, чем *хорош* этот последний формат: вся история действий машины выстроена в один поток символов – формат, очень любимый многими программистами. При чтении или записи файлов либо передаче цифровых данных идеальный подход – читать или писать поток символов один за другим от начала до конца без скачков вперед или назад.

Отметим также, что Тьюринг поместил очередную *m*-конфигурацию *перед* очередным текущим символом. Эти пара элементов была определена им как *конфигурация*, и она же появляется в полной конфигурации в том же самом порядке, что и в первых двух столбцах таблицы машины. Можно взять эту пару из *m*-конфигурации и символа и, «пробежав» по колонкам таблицы *m-конфиг.* и *символ*, найти совпадение. (Конечно, лучше, когда машина содержит в столбце *символ* фактические символы, а не «Any», «None» или пустоту.) Тьюринг действительно автоматизирует этот процесс поиска при построении своей Универсальной машины.

Дальше Тьюринг объясняет свой выбор печатать числовую последовательность через одну клетку:

> Соглашение о записи символов только через одну клетку очень полезно: я всегда буду делать это. Я буду называть одну последовательность чередующихся клеток *F*-клетками, а другую последовательность *E*-клетками. Символы в *E*-клетках будут подлежать стиранию. Символы в *F*-клетках образуют непрерывную последовательность. Без пропусков, пока не достигнет конца.

Раньше я называл их числовыми и нечисловыми клетками. Вы можете запомнить, что есть что, по словам *Figures* (*цифры*) (нули и единицы) и *Erasable* (*стираемые*). Комментарий «без пропусков, пока не достигнет конца» относится только к *F*-клеткам. Цифры вычисляемой последовательности всегда печатаются последовательно слева направо без пропусков *F*-клеток и перезаписи цифр в *F*-клетках. Эти правила обязательны для Универсальной машины Тьюринга.

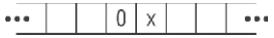
E-клетки – разновидность рабочей памяти, возможно, эквивалентной в некотором смысле человеческой памяти.

> Нет нужды иметь более одной *E*-клетки между каждой парой *F*-клеток: кажущаяся потребность в большем количестве *E*-клеток может быть удовлетворена наличием достаточно богатого разнообразия символов, которые могли бы печататься в *E*-клетках.

Во второй машине Тьюринга применялся метод идентификации знаков с помощью печати символов *x* в *E*-клетках. Это – общий метод, который он будет часто эксплуатировать, таким образом он дает ему название.

Если символ β находится в *F*-клетке *S* и символ α находится в *E*-клетке справа от *S*, то будем говорить, что *S* и β *помечены* символом α. Процесс печати этого α будет называться пометкой β (или *S*) символом α.

Говорят, что этот 0 (в *F*-клетке) *помечен x*:



Оказывается, эти метки очень удобны и являются одним из лучших изобретений Тьюринга.

Однако метки не так уж и нужны. Можно определить машины, которые используют только два символа или способны лишь отличать пустую клетку от помеченной. Такой подход изучался математиком Эмилем Постом в его интересной статье[1], где независимо от Тьюринга описана похожая конструкция. У Поста есть «работник» с набором последовательно расположенных «ящиков». Работник может:

(a) *пометить ящик, в котором он находится (если тот пуст)*;
(b) *стереть метку на ящике, в котором он находится (если тот помечен)*;
(c) *переместиться в ящик справа*;
(d) *переместиться в ящик слева*;
(e) *определить, помечен или нет, тот ящик, в котором он находится.*

Пост фактически не показывает действия своего работника на реальных примерах. Работа с клетками или ящиками, которые могут быть только помеченными или непомеченными, очевидно, существенно более трудоемка, чем метод Тьюринга.

---

[1] Emil L. Post, «Finite Combinatory Processes. Formulation I», *The Journal of Symbolic Logic*, Vol. I, No. 3 (Sep. 1936), 103–105. Перепечатана в Мартин Дэвис (Martin Davis), ed., *The Undecidable,* 289–291. Хотя статья Поста была издана до Тьюринга, статья Поста была получена журналом *The Journal of Symbolic Logic* 7 октября 1936 года; *Proceedings of the London Mathematical Society* получила статью Тьюринга 28 мая 1936 года. (Русск. перевод в кн. Успенский В. А. Машины Поста. – Сер.: «Популярные лекции по математике».– М.: Наука, 1979).

Глава 6

Сложение и умножение

Еще в начале мая 1935 года Алан Тьюринг решил поступать в Принстонский университет и обратился за стипендией на поездку[1]. Когда год спустя он узнал, что профессор математики Принстона Алонзо Чёрч тоже опубликовал статью об Entscheidungsproblem (проблеме разрешимости), он «совершенно точно решил»[2], что ему нужно ехать именно туда.

Помог Макс Ньюмэн. В том же письме, где Ньюмэн сообщал Чёрчу о работе Тьюринга (страница 83), он также просил о помощи в получении Тьюрингом стипендии:

> Стоит отметить, что работа Тьюринга совершенно самостоятельна: он работал без какого-либо надзора или чьей-либо критики. Это делает ее настолько важной, что он должен как можно скорее войти в контакт с ведущими в этом направлении учеными, чтобы не превратиться в законченного отшельника[3].

Склонность работать в одиночку, без постороннего влияния действительно была одной из больших проблем Тьюринга. Ранее в своей жизни Тьюринг повторно изобрел теорию биномов и разработал свою собственную систему обозначений для исчисления. Наверное, это было хорошо, что, атакуя проблему разрешимости, он не был знаком с более ранней работой Чёрча и его коллег, иначе он не пришел бы к столь интересному решению. Однако вообще важно знать, что происходит в остальном мире, а в области математической логики Принстон был именно таким местом. Тьюрингу не удалось получить проктеровскую стипендию, на которую он претендовал, но оказалось возможным получить стипендию в своем колледже Кингс.

---

[1] Andrew Hodges, *Alan Turing*: The Enigma (Simon & Schuster, 1983), 95.
[2] Hodges, *Alan Turing*, 113.
[3] Там же.

Интеллектуальная атмосфера вокруг города Принстона, Нью-Джерси, стала в последнее время более яркой с образованием Института перспективных исследований (Institute for Advanced Study, IAS). IAS был создан на пожертвования Луиса Бэмбергера (Louis Bamberger) в размере 5 млн долл., который сначала создал сеть универмагов Бэмбергера, а затем продал ее Мэйсиз (Macy's) накануне обвала фондовой биржи в 1929 году.

С самого начала Институт перспективных исследований планировался как место, стимулирующее научные и исторические исследования. Первое время Школа математики IAS делила одно здание с Отделением математики Принстонского университета, так что два учебных заведения в значительной степени подпитывали друг друга. IAS вскоре стал меккой для талантливых ученых и математиков, часть которых спасалась бегством от нарастающей опасности в Европе. Самым известным из них был Альберт Эйнштейн, который приехал в IAS в 1933 году и оставался там до конца жизни.

По прибытии в Принстон в сентябре 1936 года Тьюрингу очень хотелось встретиться с Куртом Гёделем. Годом раньше Гёдель тоже был в IAS и позже вернулся туда, но с Тьюрингом он так никогда и не встретился.

В IAS был Джон фон Нейман, которого Тьюринг встречал в Кембридже, из Кембриджа был и Г. Х. Харди. В IAS были также Рихард Курант и Герман Вейль, сбежавшие из Гёттингена несколькими годами ранее.

В Принстонском университете Тюринг пробыл два года, на втором году обучения получал проктеровскую стипендию (2000$). Научным руководителем диссертации Тьюринга стал Чёрч. Под его руководством Тьюринг написал диссертацию[1] и 21 июня 1938 года получил степень доктора философии. Спустя месяц он вернулся в Англию, отклонив предложение Джона фон Неймана стать его ассистентом в IAS с зарплатой 1500$ в год.

Весной 1939 года Алан Тьюринг вернулся в Кембридж, чтобы читать курс по основаниям математики. Четыре года назад Тьюринг занимался основаниями математики с Максом Ньюмэном и узнал о проблеме разрешимости. Теперь же у Тьюринга была возможность

---

[1] Alan Turing, «Systems of Logic Based on Ordinals», *Proceedings of the London Mathematical Society*, 2nd Series, Volume 45 (1939), 161–228. Переиздано в Alan Turing, *Collected Works of A. M. Turing: Mathematical Logic* (Elsevier, 2001), 161–228, and B. Jack Copeland, ed., *The Essential Turing* (Oxford University Press, 2004) 146–204.

задавать на заключительном экзамене вопрос о недоказуемости проблемы разрешимости на основании его собственной работы о вычислимых числах[1].

В своей статье Тьюринг просит читателей поверить в то, что его машины действительно могут вычислять нетривиальные числовые последовательности. На самом деле мы до сих пор не видели ничего, что можно было бы назвать *вычислением*. В первом примере Тьюринга машина якобы печатала двоичный эквивалент 1/3, но делала она это, просто тупо чередуя нули и единицы. Конечно, это не деление 1 на 3. Ни одна машина не осуществляет общий процесс для вычисления произвольного рационального числа путем деления числителя на знаменатель.

Даже программисты, работающие с машинным кодом низкого уровня, привыкли к вычислительной технике, выполняющей основные математические операции сложения и вычитания. По этой причине нас может настораживать – и даже немного пугать – машина, в которой даже сложение должно выполняться путем определения конфигураций и операций.

Давайте развеем наши опасения, создав машины, превосходящие тривиальные. И убедимся в том, что машины Тьюринга действительно могут складывать и умножать (а значит, еще и вычитать, делить, возводить в степень и, возможно, даже писать стихи).

Первый пример – небольшая машина Тьюринга, которая последовательно вычисляет все положительные целые числа. Эта машина не соблюдает соглашения Тьюринга, потому что записывает очередное число поверх предыдущего. При печати результатов она не пропускает ни одной клетки и заменяет каждый результат следующим, еще большим числом. Кроме того, учитывая, что эти числа – целые, я строил машину для печати цифр на ленте так, как мы обычно записываем целые числа, – слева направо, начиная со старших цифр. Несмотря на несоблюдение соглашений Тьюринга, эта машина *действительно* показывает, как увеличивается число при добавлении к нему 1, что является по меньшей мере одним из базовых умений современного компьютера.

В своих примерах готическим буквам я предпочел содержательные слова, выделяемые жирным шрифтом, а иногда в последующих примерах – целые словосочетания, разделяемые черточками. Как и Тьюринг, я использую слово «none» для обозначения пустой клетки. В отличие от Тьюринга, я использую слово «else» для указания на то, что конфигурация должна применяться ко всем прочим явно не ука-

---

[1] Hodges, *Alan Turing*, 152.

занным символам. Эта конкретная машина начинает работу в конфигурации **begin** и имеет всего лишь три *m*-конфигурации:

m-конфиг.	*символ*	*операции*	*заключит. m-конфиг.*
begin	none	P0	**increment**
increment	0	P1	rewind
	1	P0,L	increment
	none	P1	**rewind**
rewind	none	L	increment
	else	R	rewind

m-конфигурация **begin** просто печатает единственный 0 и затем переключается на **increment.** *m*-конфигурация **increment** читает цифру. Если она – 0, **increment** меняет ее на 1 и заканчивает увеличение всего числа. Если она – 1, то **increment** меняет ее на 0 и сдвигает головку влево для переноса 1 в старший разряд числа и увеличения его на 1. *m*-конфигурация **rewind** перемещает головку вправо до самого младшего разряда числа для подготовки его к следующему увеличению.

Как только вы начинаете описывать машины, выполняющие арифметические действия, становится понятно, почему так удобны двоичные числа. Вот эквивалентная машина, которая генерирует все положительные целые числа не в двоичной системе счисления, а в десятичной:

m-конфиг.	*символ*	*операции*	*заключит. m-конфиг.*
begin	none	P0	**increment**
increment	0	P1	**rewind**
	1	P2	**rewind**
	2	P3	**rewind**
	3	P4	**rewind**
	4	P5	**rewind**
	5	P6	**rewind**
	6	P7	**rewind**
	7	P8	**rewind**
	8	P9	**rewind**
	9	P0, L	**increment**
	none	P1	**rewind**
rewind	none	L	**increment**
	else	R	**rewind**

Проблема здесь в том, что машина должна явно проверить каждую десятичную цифру. Двоичная система счисления проще, потому что у нее меньше вариантов. Таблицы двоичного сложения и умножения совсем маленькие:

+	**0**	**1**
0	0	0
1	1	10

и

×	**0**	**1**
0	0	0
1	0	1

Я использую эти правила сложения и умножения во втором примере этой главы. Эта машина придерживается соглашений Тьюринга и вычисляет квадратный корень из 2 в двоичной системе. На самом деле, с учетом того, что запятая в двоичном числе должна предшествовать всем его цифрам, машина вычисляет

$$\frac{\sqrt{2}}{2}$$

или в десятичной системе – 0,70710678... Ради ясности и простоты при описании машины я буду считать, что она вычисляет $\sqrt{2}$.

Реализуемый машиной алгоритм вычисляет один двоичный знак за один цикл. Предположим, что машина работала некоторое время и уже определила первые четыре цифры. Первые четыре цифры $\sqrt{2}$ в двоичном коде – 1,011, что равносильно $1^3/_8$, или десятичному числу 1,375. Какой будет следующая цифра? Стратегия машины – полагать, что следующей цифрой всегда будет 1. Для проверки этого предположения умножим 1,0111 на себя:

```
    1,0111
  × 1,0111
    10111
   10111
  10111
 00000
10111
10,00010001
```

Произведение превышает 2, поэтому предположение было неправильным. Значит, пятая цифра – это 0, и поэтому первые пять цифр – 1,0110. Таким же образом определим шестую цифру. Предположим, что шестая цифра – 1, и умножим 1,01101 на себя:

```
    1,01101
  × 1,01101
    101101
   000000
  101101
 101101
000000
101101
1,1111101001
```

Теперь результат меньше, чем 2, поэтому предположение было правильным. Теперь у нас шесть цифр – 1,01101, а в десятичной записи – $1^{3}/_{32}$, или 1,40625.

Очевидно, что машина «корень из двух» требует умножения. В общем случае умножение двух многозначных чисел требует, чтобы каждая цифра одного числа умножалась на каждую цифру другого числа. Если у одного числа n цифр, а у другого m цифр, то общее количество умножений цифра на цифру будет $(n \times m)$.

При выполнении умножения вручную мы обычно умножаем одно из чисел целиком на отдельные цифры другого числа, получая n или m частичных произведений, которые затем складываем. Предлагаемая мной машина умножает несколько иначе – с сохранением промежуточного итога умножения. Каждое произведение бита на бит добавляется к этому промежуточному итогу. Хитрость данного частичного сложения заключается в том, что в большинстве случаев очередное произведение бита на бит добавляется *не* к самому младшему (крайнему) разряду промежуточного итога, а к одному из более старших разрядов.

Например, рассмотрим умножение числа 1,01101 на себя. Каждый из шести битов должен быть умножен на себя и на остальные пять битов, что потребует 36 побитовых умножений. Само умножение тривиально: если 1 умножается на 1, то результат – 1; иначе – 0. Местоположение этого результата в промежуточном итоге зависит от положения битов в числе. Если третий справа бит умножается на четвертый справа бит, то результат добавляется к шестому справа разряду промежуточного итога. (Это становится понятней, если нумеровать биты с нуля: третий справа бит имеет номер 2; четвертый справа бит – номер 3; их сумма равна 5, а это – номер двоичного разряда, к которому добавляется произведение.)

При определении двоичного квадратного корня из 2 мы всегда умножаем n-битное число на себя. Если результат имеет $(2n - 1)$ бит,

значит, произведение меньше 2 и предположение о том, что новая последняя цифра – именно 1, было правильным. Если результат имеет $2n$ бит, произведение превышает 2, поэтому новая последняя цифра должна быть 0. Машина будет использовать этот факт, для того чтобы определить, является ли каждая новая цифра 0 или 1.

Машина, которую я представлю, придерживается соглашений Тьюринга в том смысле, что всё, что она будет печатать в *F*-клетках, – это цифры квадратного корня из 2 в порядке их вычисления. Все остальное, включая хранение промежуточного итога умножения, будет делаться в *E*-клетках.

Машина начинает работу с *m*-конфигурации **begin**. В качестве граничной метки машина использует вместо ə символ @. (Стоит лишь сказать, что это более доступный в сегодняшних компьютерах символ.) Машина начинает с печати граничной метки и цифры 1:

m-конфиг.	*символ*	*операции*	*заключит. m-конфиг.*
begin	none	P@, R, P1	new

Таким образом, единственное начальное предположение, которое делает машина, – это то, что квадратный корень из 2 не меньше 1, но меньше 2.

Машина всегда возвращается к *m*-конфигурации **new**, когда готова вычислить следующую цифру. Конфигурация перемещает головку к самой левой цифре:

new	@	R	**mark-digits**
	else	L	**new**

Остальную часть машины будет проще понять, если мы разберем ее действия, после того как она уже вычислила несколько цифр. Вот лента с первыми тремя уже вычисленными цифрами, представляющими собой двоичный эквивалент числа 1,25. Машина будет печатать четвертую цифру (которую я буду называть «неизвестной») в помеченной вопросительным знаком клетке:

Этот знак вопроса нужен лишь для нашего удобства; на самом деле он не появляется на ленте и не используется машиной!

При подготовке к умножению машина маркирует цифры числа. (Напомню, что Тьюринг определил «маркировку» как печать симво-

ла справа от цифры.) Машина использует разнообразные *x*-маркеры наподобие Примера II (страница 105) машины Тьюринга. *m*-конфигурация **mark-digit** помечает все известные цифры маркером *x*, а неизвестную – маркером *z* (который я вскоре объясню), а также помечает младший разряд промежуточного итога маркером *r*:

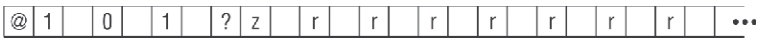
mark-digits	0	R, Px, R	**mark-digits**
	1	R, Px, R	**mark-digits**
	none	R, Pz, R, R, Pr	**find-x**

Теперь лента выглядит так:

Символ *r* – самый младший разряд промежуточного итога, который нужно понимать как 0. Следующий блок печатает еще два маркера *r* для каждого *x*, стирая при этом маркеры *x*.

find-x	x	E	**first-r**
	@	N	**find-digits**
	else	L, L	**find-x**
first-r	r	R, R	**last-r**
	else	R, R	**first-r**
last-r	r	R, R	**last-r**
	none	Pr, R, R, Pr	**find-x**

Теперь на ленте 7 цифр промежуточного итога, представляющих собой начальное значение 0000000:



Разряды промежуточного итога следуют в порядке, обратном вычисляемому числу. Самый младший разряд промежуточного итога находится слева. Заданных семи цифр промежуточного итога достаточно, если верно предположение о том, что неизвестная цифра – 1. Если потребуется восьмая цифра, то неизвестная цифра – 0.

Число, которое машина должна умножить на себя, состоит из уже вычисленного числа (101 в этом примере) и новой цифры, которая предполагается равной 1, так что этим числом будет 1011. Для отслеживания, какие цифры умножаются друг на друга, машина помечает

цифры маркерами *x*, *y* и *z*. Всегда при умножении только одна цифра помечена маркером *x* и только одна – маркером *y*, и именно они в этот момент перемножаются. Если случается так, что маркеры *x* и *y* совпадают (накладываются друг на друга – *прим. перев.*), то вместо них используется маркер *z*, а помеченная им цифра умножается на себя.

Именно поэтому неизвестная цифра (изначально 1) сразу помечается маркером *z*. Самое первое умножение – это умножение неизвестной цифры на саму себя; тем не менее для анализа последующих конфигураций будет полезно помнить, что при умножении любая из цифр может быть помечена маркером *x* и любая другая – маркером *y*, но только одна – маркером *z*.

Теперь мы готовы к первому побитовому умножению. Машина перемножает либо две цифры, помеченные маркерами *x* и *y*, либо одну цифру, помеченную маркером *z*, на саму себя. *m*-конфигурация **find-digits** сначала возвращается к ограничителю, а затем переходит к **find-1st-digit**, чтобы найти крайнюю левую цифру, помеченную *x*, *y* или *z*.

find-digits	@	R, R	**find-1st-digit**
	else	L, L	**find-digits**
find-lst-digit	x	L	**found-1st-digit**
	y	L	**found-1st-digit**
	z	L	**found-2nd-digit**
	none	R, R	**find-lst-digit**

Если **find-1st-digit** обнаруживает *x*, *y* или *z*, она устанавливает головку на цифре. В зависимости от символа машина переходит к **found-1st-digit** или **found-2nd-digit**.

Если первая помеченная цифра – 0, вторая уже не нужна, так как в любом случае произведение будет 0. Таким образом, мы можем добавить 0 к текущей сумме, перейдя к **add-zero**:

found-1st-digit	0	R	**add-zero**
	1	R, R, R	**find-2st-digit**

Если первая цифра – 1, то нужно найти вторую. Машина ищет вторую цифру, помеченную *x* или *y*:

find-2st-digit	x	L	**found-2st-digit**
	y	L	**found-2st-digit**
	none	R, R	**find-1st-digit**

Вторая цифра определяет, что должно быть добавлено к текущей сумме:

found-1st-digit	0	R	**add-zero**
	1	R	**add-zero**
	none	R	**add-one**

Заметьте, что пустая *F*-клетка – это неизвестная цифра, которая по предположению равна 1. В нашем примере неизвестная цифра помечена маркером *z*, поэтому для добавления 1 к промежуточному итогу будет использоваться **add-one**.

Обычно добавление 0 к промежуточному итогу не должно влиять на него; однако эта машина должна обработать промежуточный итог независимо от того, что к нему добавляется.

При описании инициализации промежуточного итога в правых *E*-клетках я сказал, что буква *r* обозначает 0. Буквы *s* и *t* тоже обозначают 0, а буквы *u*, *v* и *w* представляют 1. Это множество букв требуется для отслеживания в промежуточном итоге той позиции, куда добавляется произведение битов.

m-конфигурация **add-zero** заменяет первый найденный *r* на *s* или первый найденный *u* на *v*:

add-zero	r	Ps	add-finished
	u	Pv	add-finished
	else	R, R	add-zero

Замена *r* на *s* (означающих 0) и *u* на *v* (означающих 1) гарантирует, что очередная цифра будет добавляться к следующей позиции промежуточного итога.

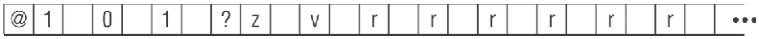
Добавление 1 к промежуточному итогу сложнее. Первый *r* (означающий 0) заменяется на *v* (означающий 1) или же первый *u* (означающий 1) заменяется на *s* (означающий 0). В последнем случае необходим еще и перенос:

add-one	r	Pv	**add-finished**
	u	Ps, R, R	**carry**
	else	R, R	**add-one**

Если перенос приводит к записи цифры в пустую клетку, то промежуточный итог превысил 2, и поэтому новой конфигурацией становится **new-digit-is-zero**:

carry	r	Pu	**add-finished**
	none	Pu	**new-digit-is-zero**
	u	Pr, R, R	**carry**

После первого перемножения битов и прибавления к промежуточному итогу лента выглядит так:



Заметьте, что первый *r* был заменен на *v* (означающий 1).

Теперь нужно передвинуть маркеры *x*, *y* и *z*, чтобы задать следующую пару перемножаемых битов. Маркер *x* обычно перемещается на один знак влево. (Маркер z, как вы помните, означает просто совпадение маркеров *x* и *y*, поэтому маркер *z* становится маркером *y*, а маркер *x* печатается на один знак левее.) Но когда маркер *x* достигает конца (за самым старшим битом), маркер *y* сдвигается на один знак влево, а маркер *x* возвращается назад к самой правой (младшей) цифре. Когда маркер *y* достигает конца, умножение завершается.

Сначала головка перемещается к ограничителю:

add-finished	@	R, R	**erase-old-x**
	none	L, L	**add-finished**

Если **erase-old-x** находит *x*, она стирает его; если она находит *z*, он заменяется на *y*. В любом случае головка сдвигается к следующей слева *E*-клетке:

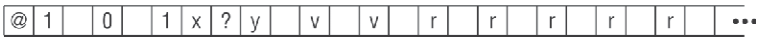
erase-old-x	x	E, L, L	**print-new-x**
	z	Py, L, L	**print-new-x**
	else	R, R	**erase-old-x**

Теперь можно печатать следующий маркер *x*:

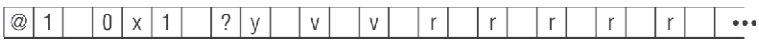
print-new-x	@	R, R	**erase-old-x**
	y	Pz	**find-digits**
	none	Px	**find-digits**

Лента нашего примера теперь готова вернуться к **find-digits** для следующего перемножения битов:

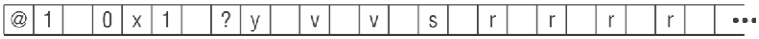
Результатом этого перемножения будет еще одна 1, добавляемая к промежуточному итогу, но на сей раз она должна добавляться одной позицией дальше, так как она всегда добавляется к крайним левым *r* или *u*:



Затем машина сдвигает маркер *x* на одну позицию влево:

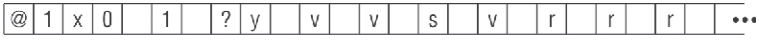


Это умножение дает 0, который добавляется к промежуточному итогу. Общее значение не изменяется, но крайний левый *r* меняется на *s*:



Маркер *x* снова сдвигается влево:

Еще одно перемножение битов приводит к замене самого левого *r* на *v*:



Теперь *x* достиг ограничителя. Этот случай обрабатывается конфигурациями **erase-old-y** и **print-new-y**:

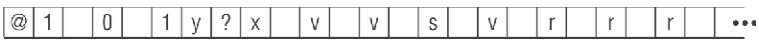
erase-old-y	y	E, L, L	**print-new-y**
	else	R, R	**erase-old-y**
print-new-y	@	R	**new-digit-is-zero**
	else	Py, R	**reset-new-x**

Отметим, что если ограничителя достигает маркер *y*, то умножение полностью заканчивается с промежуточным итогом, не выходящим за пределы выделенной для него области. В этом случае мы знаем, что неизвестная цифра – 1.

В противном случае маркер *x* должен быть вновь установлен на самый младший бит числа, который является неизвестной цифрой:

reset-new-x	none	R, Pu	**flag-result-digits**
	else	R, R	**reset-new-x**

Теперь маркеры *x* и *y* располагаются на ленте примерно так:



Многое еще предстоит сделать. Следующее произведение битов должно быть добавлено ко второй цифре промежуточного итога. Для достижения этого первый *s* или *v* в промежуточном итоге заменяется соответственно на *t* или *w*:

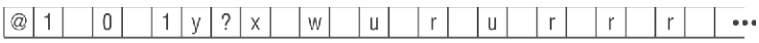
flag-result-digits	s	Pt, R, R	**unflag-result-digits**
	v	Pw, R, R	**unflag-result-digits**
	else	R, R	**flag-result-digits**

Остальные маркеры s и *v* заменяются соответственно на *r* и u:

unflag-result-digits	s	Pr, R, R	**unflag-result-digits**
	v	Pu, R, R	**unflag-result-digits**
	else	N	**find-digits**

Этот процесс гарантирует, что следующее произведение битов добавляется в нужное место промежуточного итога.

Вот теперь лента готова к следующему умножению битов, результат которого будет добавлен к первому *r* или *u* промежуточного итога.



Умножение завершается одним из двух уже известных вам способов. Если машина пытается перенести бит из промежуточного итога в пустую клетку, значит, результат превысил 2, неизвестная цифра – 0, а новой конфигурацией становится **new-digit-is-zero**. В противном случае, если маркер *y* достигает ограничителя, умножение полностью завершается с промежуточным итогом, не превышающим 2, и вступает в силу **new-digit-is-one**.

Эти два блока очень похожи. Во-первых, машина возвращается к ограничителю:

new-digit-is-zero	@	R	**print-zero-digit**
	else	L	**new-digit-is-zero**

Теперь машина может найти пустую клетку и напечатать в ней 0. Двигаясь по цифрам, она может стереть все оставшиеся маркеры:

print-zero-digit	0	R, E, R	**print-zero-digit**
	1	R, E, R	**print-zero-digit**
	none	P0, R, R, R	**cleanup**

Точно так же *m*-конфигурация **new-digit-is-one** печатает в качестве новой цифры 1 и тоже входит в режим **cleanup** (очистки):

new-digit-is-one	@	R	**print-one-digit**
	else	L	**new-digit-is-one**

print-one-digit	0	R, E, R	**print-one-digit**
	1	R, E, R	**print-one-digit**
	none	P1, R, R, R	**cleanup**

Напечатав новую цифру, *m*-конфигурация **cleanup** удаляет промежуточный итог, а затем переходит в **new** для следующей цифры.

cleanup	none	N	**new**
	else	E, R, R	**cleanup**

Теперь на ленте есть новая четвертая цифра, и она подготовлена к пятой:

Очевидно, машина Тьюринга – неудобная среда программирования. В большинстве языков программирования есть функция sqrt (или что-то в этом роде), которая вычисляет квадратный корень не только из 2, но и из любого другого числа.

Конечно, эти функции вычисления квадратного корня нередко имеют ограниченную точность. В большинстве современных компьютеров числа с плавающей запятой хранятся в формате, который соответствует стандарту Института инженеров по электротехнике и радиоэлектронике (IEEE). Число с плавающей запятой двойной точности округляется до 52 битов или примерно до 15–16 десятичных цифр. Сравнительно недавно (когда стали доступны специализированные наборы математических функций с большей точностью), если была нужна большая, чем эта, точность, приходилось рассчитывать в основном на самого себя. Действуя, как я только что, вы могли бы

проявить себя в наращивании возможностей машины Тьюринга для выполнения вычислений с произвольным числом цифр.

На реальном компьютере удобно, по крайней мере, складывать и умножать. Столкнувшись с работой по реализации на машине Тьюринга функции несколько иного типа, можно было бы принять во внимание сборку набора обычных машинных таблиц, которые можно использовать в качестве строительных блоков для создания более сложных таблиц.

Именно это Тьюринг и делает дальше, хотя его истинная цель – универсальная машина, которая может моделировать любую другую машину.

Глава 7

Они же – подпрограммы

Каждый программист знает, что почти во всех программных проектах часто встречаются определенные типы задач. Иногда задачи – одни и те же; чаще – с некоторыми вариациями. Даже в машине square-root-of-2 некоторые *m*-конфигурации очень похожи. Взгляните, например, на эти три:

new	@	R	**mark-digits**
	else	L	**new**
new-digit-is-zero	@	R	**print-zero-digit**
	else	L	**new-digit-is-zero**
new-digit-is-one	@	R	**print-one-digit**
	else	L	**new-digit-is-one**

Во всех этих *m*-конфигурациях головка в цикле перемещается влево, пока не встретится ограничитель @. Затем головка сдвигается на шаг вправо (на самую левую цифру), и машина переключается на другую *m*-конфигурацию.

Было бы удобно заранее определить, что машине потребуются некоторые схожие *m*-конфигурации, и предопределить специальные *m*-конфигурации исключительно для этих рутинных операций. Это помогло бы прояснить определенные стратегии, используемые в программировании машины Тьюринга, и упростить окончательную работу.

Назовем *m*-конфигурацию, возвращающую головку к ограничителю, **goto-sentinel**. Потом, когда при записи состояний конкретной машины нам будет нужно установить головку на символе справа от ограничителя, мы просто зададим **goto-sentinel** и больше не будем задумываться о том, как это делается. Это не только чуть сократит описание машины, но и (теоретически) поможет каждому, кто должен разобраться в машине и понять ее.

Можно определить **goto-sentinel** как

goto-sentinel	{	@	R	**?????**
		else	L	**goto-sentinel**

и сразу заметить проблему, обозначенную обращающей на себя внимание группой вопросительных знаков. Найдя ограничитель, машина должна перейти в какую-то другую *m*-конфигурацию, но она неизвестна, пока не возникнет реальная потребность в применении **goto-sentinel** в машине. В самом общем случае, чтобы сохранить гибкость **goto-sentinel**, нам нужен некоторый способ задать заключительную *m*-конфигурацию.

Решение – в том, чтобы определить **goto-sentinel** как математическую функцию, аргумент которой – заключительная *m*-конфигурация:

goto-sentinel(A)	{	@	R	**A**
		else	L	**goto-sentinel(A)**

Теперь *m*-конфигурации **new**, **new-digit-is-zero** и **new-digit-is-one** можно исключить. В начале работы машины square-root-of-2 вместо переходов от **begin** к **new** и от **new** к **mark-digits** мы можем задать:

begin	none	P@, R, P1	**goto-sentinel(mark-digits)**

Вместо того чтобы определять **carry** с переходом к **new-digit-is-zero**

carry	{	r	Pu	**add-finished**
		none	Pu	**new-digit-is-zero**
		u	Pr, R, R	**carry**

можно сослаться на **goto-sentinel**, чтобы вернуться к ограничителю, а затем переключиться на **print-zero-digit**:

carry	{	r	Pu	**add-finished**
		none	Pu	**goto-sentinel(print-zero-digit)**
		u	Pr, R, R	**carry**

Говоря о **print-zero-digit**, разве не видно, что она отличается от **print-one-digit** только цифрой, которую печатает? Можно с пользой для себя определить обобщенную функцию **print-digit**. Ее аргумент – печатаемый символ:

print-digit(a)	0	R, E, R	**print-digit(a)**
	1	R, E, R	**print-digit(a)**
	none	Pa, R, R, R	**cleanup**

Заметьте, что операция «Pa» в последней строке означает, что печатаемый символ – аргумент **print-digit**. Теперь *m*-конфигурация **carry** становится такой:

carry	r	Pu	**add-finished**
	none	Pu	**goto-sentinel(print- digit(0))**
	u	Pr, R, R	**carry**

m-конфигурация **print-new-y** (отвечавшая за обнаружение **new-digit-is-one**) теперь становится такой:

print-new-y	@	R	**goto-sentinel(print- digit(1))**
	else	Py, R	**reset-new-x**

Сегодняшние программисты сразу узнают этот прием. Хотя различные языки программирования обеспечивают подобную возможность в виде *процедур, функций* или *методов*, наиболее общий для этого термин – *подпрограмма*. Десятки лет подпрограмма была универсальным конструктивным элементом компьютерных программ.

Программистов, читающих эту книгу, хотелось бы немного предостеречь от *слишком* буквального применения своих знаний о подпрограммах к конфигурациям с аргументами. Эти конфигурации существуют прежде всего для того, чтобы прояснить конструкцию машин Тьюринга и сделать их проще для написания. Не существует понятия «вызова» какой-либо из этих конфигураций или «возврата» из них.

Прежде чем остановиться на термине «*m*-функция», Тьюринг называл эти конфигурации с аргументами «скелетными таблицами» («skeleton tables»). Таблицу машины, которая использует скелетные таблицы, он называет «сокращенной таблицей» («abbreviated table»).

4. Сокращенные таблицы.

Существуют определенные типы операций, используемые почти всеми машинами, и в некоторых машинах они применяются во множестве сочетаний. Эти операции включают копирование последовательностей символов, сравнение последовательностей, стирание всех символов заданного вида и т. д. Там, где появляются такие операции, за счет применения «скелетных таблиц» мы можем значительно сократить таблицы *m*-конфигураций. В скелетных таблицах используются прописные немецкие буквы и строчные греческие буквы.

На удивление, немецкие прописные буквы еще труднее для чтения, чем строчные. К счастью, Тьюринг не идет дальше буквы E, но было бы полезно познакомиться с ними поближе:

Заметьте, в частности, что буква A больше похожа на U, и обратите внимание на едва уловимое различие между C и E. Греческие буквы, которые Тьюринг использует в этом разделе, – наклонные α, β и γ.

> Они имеют природу «переменных». Заменив везде каждую прописную немецкую букву на *m*-конфигурацию и каждую строчную греческую букву – символом, мы получим таблицу для *m*-конфигурации. [236]

Там, где я в своем примере для представления *m*-конфигурации использовал прописную латинскую букву, Тьюринг использует прописную немецкую букву. Там, где я для представления символа использовал строчную латинскую букву, Тьюринг использует строчную греческую букву. В примерах Тьюринга часто встречается несколько аргументов.

В наше время подпрограммы (такие как sqrt) хранятся в файлах, называемых *библиотеками*, которые позволяют программистам использовать их, задавая только их имена. Можно даже сказать, что целые операционные системы – такие как Unix, Microsoft Windows или Apple Mac OS – состоят прежде всего из подпрограмм, доступных работающим в них приложениям.

Однако для Тьюринга скелетные таблицы существуют исключительно для того, чтобы (с его точки зрения) проще создавать большие машины и (с нашей точки зрения) проще читать и понимать их.

> Скелетные таблицы должны считаться лишь сокращениями: они не являются необходимыми. Пока читатель понимает, как получить полные таблицы из скелетных таблиц, нет необходимости давать в связи с ними какие-либо точные определения.

Скелетные таблицы не являются необходимыми, говорит он, и это – правда. Если бы скелетные таблицы преподносились исключительно как интересная тема и ограничивались только этим разделом

статьи, их можно было бы легко опустить. Однако Тьюринг строит платформу для своей Универсальной машины, которая предполагает *широкое* применение скелетных таблиц, представленных в этом разделе. Без этих таблиц Универсальная машина была бы намного длиннее и сложнее, чем есть.

По этой причине некоторая осведомленность о конечных намерениях Тьюринга может помочь чуть больше понять эти таблицы. Как будет сказано в разделе 7, Универсальная машина интерпретирует ленту, которая содержит вычислительную машину, закодированную последовательностью символов. В самой левой позиции находится ограничитель ə. На ленте чередуются *F*-клетки и *E*-клетки. Как обычно, *E*-клетки – стираемые. *F*-клетки в Универсальной машине содержат, главным образом, буквы, а не цифры. При этом машина всегда печатает *F*-клетки последовательно слева направо и не стирает предыдущих символов. По этой причине две пустые клетки подряд означают, что правее них *F*-клеток нет.

Давайте рассмотрим пример:

m-конфиг.	*Символ*	*Поведение*	*Заключ. m-конфиг.*	
$\mathfrak{f}(\mathfrak{C}, \mathfrak{B}, \alpha)$	ə	L	$\mathfrak{f}_1(\mathfrak{C}, \mathfrak{B}, \alpha)$	Из *m*-конфигурации $\mathfrak{f}(\mathfrak{C}, \mathfrak{B}, \alpha)$ машина ищет символ вида α, который является самым первым слева («первый α»), после чего *m*-конфигурацией становится $\mathfrak{C}$. Если α нет, то *m*-конфигурацией становится $\mathfrak{B}$.
	not ə	L	$\mathfrak{f}(\mathfrak{C}, \mathfrak{B}, \alpha)$	
$\mathfrak{f}_1(\mathfrak{C}, \mathfrak{B}, \alpha)$	α		$\mathfrak{C}$	
	not α	R	$\mathfrak{f}_1(\mathfrak{C}, \mathfrak{B}, \alpha)$	
	None	R	$\mathfrak{f}_2(\mathfrak{C}, \mathfrak{B}, \alpha)$	
$\mathfrak{f}_2(\mathfrak{C}, \mathfrak{B}, \alpha)$	α		$\mathfrak{C}$	
	not α	R	$\mathfrak{f}_1(\mathfrak{C}, \mathfrak{B}, \alpha)$	
	None	R	$\mathfrak{B}$	

Конечно, он мог бы начать с примера попроще, но достоинство этого – в демонстрации всех особенностей. Справа от таблицы находятся пояснения Тьюринга. (Определяя Универсальную машину, Тьюринг тоже помещает свои пояснения справа от таблиц.)

Хотя Тьюринг фактически определяет функцию $\mathfrak{f}$, она требует двух других функций $\mathfrak{f}_1$ и $\mathfrak{f}_2$. Все они имеют три одинаковых аргумента: две *m*-конфигурации и один символ. *m*-конфигурация $\mathfrak{f}$ перемещает головку влево, пока та не встретит α. После чего *m*-конфигурацией становится $\mathfrak{f}_1$. Эта *m*-конфигурация движется вправо, пока не встретит α. (Отметим, что α – третий аргумент $\mathfrak{f}$.) Встретив α, она переходит к *m*-конфигурации $\mathfrak{C}$, первому аргументу $\mathfrak{f}$. *m*-конфигурации $\mathfrak{f}_1$ и $\mathfrak{f}_2$

очень похожи. Вместе они, по сути дела, ищут пару пустых клеток. Как только $\mathfrak{f}_1$ находит пустую клетку, она переключается на $\mathfrak{f}_2$. Если следующая клетка – непустая, $\mathfrak{f}_2$ возвращается к $\mathfrak{f}_1$. И только когда $\mathfrak{f}_2$ встречает пустую клетку, которая должна быть второй пустой подряд, она останавливается и переходит к *m*-конфигурации $\mathfrak{B}$, второму аргументу $\mathfrak{f}$. В этом случае символ α не найден.

Итак, $\mathfrak{f}$ означает *поиск*. Если она находит α, то перейдет в *m*-конфигурацию $\mathfrak{C}$, а головка будет находиться на первом (самом левом) α. Если она не может найти α, то переходит в *m*-конфигурацию $\mathfrak{B}$.

На самом деле в этой таблице есть небольшое недоразумение. В двух *m*-конфигурациях $\mathfrak{f}_1$ и $\mathfrak{f}_2$ условие «not α», казалось бы, должно означать «любую непустую клетку, отличную от α», потому что второй конфигурации требуется «None» или пустая клетка; однако у первой *m*-конфигурации нет условия «None», и оно добавлено для совместимости. Вариант «None» должен быть тем же, что и «not α»[1].

В таблице полной машины эта скелетная таблица должна появляться в столбце «*заключ. m-конфиг.*» примерно так:

m-конфиг.	*символ*	*операции*	*заключ. m-конфиг.*
			$\mathfrak{f}(\mathfrak{q}, \mathfrak{r}, x)$

m-конфигурации $\mathfrak{q}$ и $\mathfrak{r}$ должны быть определены в другом месте машины, а x должен быть символом, используемым машиной.

Если мы всюду заменим $\mathfrak{C}$ на $\mathfrak{q}$ (например), $\mathfrak{B}$ на $\mathfrak{r}$ и α на x, мы получим полную таблицу для *m*-конфигурации $\mathfrak{f}(\mathfrak{q}, \mathfrak{r}, x)$.

В контексте полной машины эта скелетная таблица фактически разворачивается в следующую таблицу:

m-конфиг.	*символ*	*операции*	*заключ. m-конфиг.*
$\mathfrak{f}$	ə	L	$\mathfrak{f}_1$
	not ə	L	$\mathfrak{f}$
$\mathfrak{f}_1$	x		$\mathfrak{q}$
	not x	R	$\mathfrak{f}_1$
	None	R	$\mathfrak{f}_2$

[1] Это одно из нескольких исправлений из сноски в приложении к статье Эмиля Поста «Recursive Unsolvability of a Problem of Thue», *The Journal of Symbolic Logic*, Vol. 12, No. 1 (Mar. 1947), 1–11. Статья целиком перепечатана в Martin Davis, ed., *The Undecidable* (Raven Press, 1965), 293–303. Приложение (со сноской, включенной в текст) перепечатано в B. Jack Copeland, ed., *The Essential Turing* (Oxford University Press, 2004), 97–101.

$$\mathfrak{f}_2 \quad \begin{cases} x & \\ \text{not } x & R \\ \text{None} & R \end{cases} \quad \begin{matrix} \mathfrak{q} \\ \mathfrak{f}_1 \\ \mathfrak{r} \end{matrix}$$

Поскольку функция $\mathfrak{f}$ в одной и той же машине может использоваться неоднократно, расширенные версии *m*-конфигураций $\mathfrak{f}$, $\mathfrak{f}_1$ и $\mathfrak{f}_2$ должны нуждаться в различных именах, каждый раз, когда они используются.

$\mathfrak{f}$ называют «функцией *m*-конфигурации», или «*m*-функцией».

Это название гораздо лучше, чем «скелетная таблица». Надеюсь, не будет путаницы, если я в большинстве случаев буду называть их просто *функциями*.

Единственное, что допустимо для подстановки в *m*-функцию, – это *m*-конфигурации и символы машины. Они должны быть перечислены более или менее явно: они могут включать такие выражения, как $\mathfrak{p}(\mathfrak{e}, x)$; на самом деле – должны, если есть хоть какие-то *m*-функции.

Если была определена *m*-функция $\mathfrak{p}$ и машина ссылается на нее в столбце «*заключ. m-конфиг*», то следует считать, что $\mathfrak{p}$ – *m*-конфигурация машины.

Здесь Тьюринг немного волнуется, потому что аргументами *m*-функций могут быть другие *m*-функции. Иными словами, *m*-функции могут быть *вложенными*. (Не волнуйтесь: примеров будет множество.) Проблема возникает из-за неявного признания бесконечной рекурсии, когда функция обращается либо к себе, либо к другой функции, которая, в свою очередь, обращается к первой. Если допускается бесконечная рекурсия, машина может прийти к бесконечному числу *m*-конфигураций, а это противоречит исходному определению Тьюрингом вычислительной машины.

Если бы мы не настаивали на этом явном перечислении, а просто заявили, что машина имеет определенные *m*-конфигурации (перечисленные) и все *m*-конфигурации, получаемые подстановкой *m*-конфигураций в определенные *m*-функции, мы получили бы бесконечное число *m*-конфигураций; например, мы могли бы сказать,

что машина должна иметь *m*-конфигурацию $\mathfrak{q}$ и все *m*-конфигурации, получаемые подстановкой *m*-конфигурации $\mathfrak{C}$ в $\mathfrak{p}(\mathfrak{C})$. Тогда она должна иметь в качестве конфигураций $\mathfrak{q}$, $\mathfrak{p}(\mathfrak{q})$, $\mathfrak{p}(\mathfrak{p}(\mathfrak{q}))$, $\mathfrak{p}(\mathfrak{p}(\mathfrak{p}(\mathfrak{q})))$...

Мы должны быть уверены, что после замены в машине всех *m*-функций число *m*-конфигураций будет все еще конечным.

Тогда наше правило интерпретации таково. Мы придаем *m*-конфигурациям машины имена, выраженные, главным образом, через *m*-функции.

И снова Тьюринг предвосхищает свою Универсальную машину, которая действительно будет выражена, главным образом, через определенные в этом разделе *m*-функции.

Мы ввели также скелетные таблицы. Все, что нам нужно, – это полная таблица *m*-конфигураций машины. Она получается многократной подстановкой в скелетные таблицы.

Возможно, в этом месте он уже *слишком* навязчив. Обычно в явном перечислении всех *m*-конфигураций машины нет необходимости. На самом деле нам лишь нужно знать, что их количество конечно.

Дополнительные примеры. [237]
(В пояснениях символ «→» употребляется в значении «машина переходит в *m*-конфигурацию...».)

Под «пояснениями» Тьюринг подразумевает зачастую непонятные описания, которые появляются справа от скелетных таблиц. Столбцы этих таблиц слегка «гуляют», а заголовков у них нет. Некоторые таблицы содержат только начальные и заключительные *m*-конфигурации. Остальные содержат столбцы с текущими символами и операции, которые нужно различать по их содержимому.

В следующем примере Тьюринг приводит *m*-функцию, которая является аргументом другой *m*-функции:

$\mathfrak{e}(\mathfrak{C}, \mathfrak{B}, \alpha)$		$\mathfrak{f}(\mathfrak{e}_1(\mathfrak{C}, \mathfrak{B}, \alpha), \mathfrak{B}, \alpha)$	Из $\mathfrak{e}(\mathfrak{C}, \mathfrak{B}, \alpha)$ стирается первый α и $\rightarrow \mathfrak{C}$. Если α нет $\rightarrow \mathfrak{B}$
$\mathfrak{e}_1(\mathfrak{C}, \mathfrak{B}, \alpha)$	E	$\mathfrak{C}$	

Символ $\mathfrak{e}$ означает «стирание». Эта функция, начав с $\mathfrak{f}$, ищет первое (самое левое) вхождение α и останавливает на нем головку. Обрати-

те внимание, что первый аргумент $\mathfrak{f}$ – функция $\mathfrak{e}_1$. Это означает, что, найдя символ α, $\mathfrak{f}$ затем перейдет в $\mathfrak{e}_1$, которая просто сотрет символ и перейдет в m-конфигурацию $\mathfrak{C}$. Если $\mathfrak{f}$ не находит символа α, она переходит в $\mathfrak{B}$.

Если вы не просто верите Тьюрингу, что это работает, а действительно хотите разобраться в этом, у вас может возникнуть вопрос, зачем $\mathfrak{e}_1$ нужно так много аргументов. Они не нужны. Ее можно определить проще – $\mathfrak{e}_1(\mathfrak{C})$.

Программисты, будьте начеку: чтобы правильно понять вложенные m-функции, ваших знаний может быть слишком много. Противьтесь почти неодолимому искушению считать, что $\mathfrak{e}_1$ должна быть «вычислена» некоторым образом до того, как будет передана в $\mathfrak{f}$. Наоборот, считайте первый аргумент $\mathfrak{f}$ *указателем* конечного пункта $\mathfrak{f}$ после обнаружения символа α.

Тьюринг определяет вторую версию функции $\mathfrak{e}$ с двумя аргументами вместо трех:

$\mathfrak{e}(\mathfrak{B}, \alpha)$	$\mathfrak{e}(\mathfrak{e}(\mathfrak{B}, \alpha), \mathfrak{B}, \alpha)$	Из $\mathfrak{e}(\mathfrak{B}, \alpha)$ стираются все символы α и $\rightarrow \mathfrak{B}$

Определение двух различных функций с одним и тем же именем, но разным числом аргументов – довольно передовой прием программирования (называемый перегрузкой функции), который не предусматривался во многих старых языках программирования.

В этой версии для стирания первого символа α применяется $\mathfrak{e}$ с тремя аргументами, но заметьте, что первый из трех аргументов – $\mathfrak{e}$ с двумя аргументами! Когда $\mathfrak{e}$ с тремя аргументами успешно обнаружила и стерла первый α, она затем переходит к $\mathfrak{e}$ с двумя аргументами, которая снова применяет $\mathfrak{e}$ с тремя аргументами для стирания очередного α. Это продолжается до тех пор, пока все символы α не будут стерты.

Очень умно. Фактически Тьюринг применил здесь вложение и рекурсию для описания реализации повторяющихся действий.

Тем не менее использование $\mathfrak{e}$ с двумя аргументами в качестве аргумента $\mathfrak{e}$ с тремя аргументами для реализации $\mathfrak{e}$ с двумя аргументами, кажется, должно вызывать страшное видение бесконечной вложенности m-конфигураций.

Последний пример кажется чуть труднее для понимания, чем многие другие. Давайте предположим, что в списке m-конфигураций некоторой машины есть $\mathfrak{e}(\mathfrak{b}, x)$ (= $\mathfrak{q}$, например).

M-функция $\mathfrak{e}$ может играть какую-то роль в машине, только когда появляется в столбце *заключ. m-конфиг.*, как, например, $\mathfrak{e}(\mathfrak{b}, x)$, где $\mathfrak{b}$ – *m*-конфигурация, используемая в машине. Теперь мы можем сказать, что $\mathfrak{e}(\mathfrak{b}, x)$ – еще одна *m*-конфигурация машины, и – пока $\mathfrak{q}$ не использована для обозначения какой-то другой *m*-конфигурации в машине – можем ссылаться на эту новую *m*-конфигурацию так же, как на $\mathfrak{q}$.

При использовании $\mathfrak{e}(\mathfrak{b}, x)$ в столбце *заключ. m-конфиг.* машины мы, в сущности, имеем еще одно дополнительное состояние машины, которое Тьюринг представляет двумя разными способами:

		Ее таблица
	$\mathfrak{e}(\mathfrak{b}, x)$	$\mathfrak{e}(\mathfrak{e}(\mathfrak{b}, x), \mathfrak{b}, x)$
или	$\mathfrak{q}$	$\mathfrak{e}(\mathfrak{q}, \mathfrak{b}, x)$.

(Точка в конце последней строки появилась потому, что она считается частью предложения, начинающегося с «Таблица является».) Из этой таблицы следует, что *m*-конфигурация $\mathfrak{e}(\mathfrak{q}, \mathfrak{b}, x)$ – еще одна *m*-конфигурация машины, как и $\mathfrak{e}_1(\mathfrak{q}, \mathfrak{b}, x)$, что показывает следующая развертка:

Или подробнее:

$\mathfrak{q}$		$\mathfrak{e}(\mathfrak{q}, \mathfrak{b}, x)$
$\mathfrak{e}(\mathfrak{q}, \mathfrak{b}, x)$		$\mathfrak{f}(\mathfrak{e}_1(\mathfrak{q}, \mathfrak{b}, x), \mathfrak{b}, x)$
$\mathfrak{e}_1(\mathfrak{q}, \mathfrak{b}, x)$	*E*	$\mathfrak{q}$.

(И снова точка вслед за $\mathfrak{q}$ в последней строке – из-за того, что эта таблица считается частью предложения.) Заметьте, что $\mathfrak{e}_1$, стерев символ, возвращается к $\mathfrak{q}$ – собственной *m*-конфигурации машины, поэтому бесконечного порождения *m*-конфигураций здесь нет.

Здесь мы могли бы заменить $\mathfrak{e}_1(\mathfrak{q}, \mathfrak{b}, x)$ на $\mathfrak{q}'$ и затем задать таблицу для $\mathfrak{f}$ (с правильными подстановками) и в конечном счете прийти к таблице, в которой *m*-функции не появляются.

Точно так же, как Тьюринг использовал $\mathfrak{q}$ для представления конфигурации $\mathfrak{e}(\mathfrak{b}, x)$, он может использовать $\mathfrak{q}'$ для представления конфигурации $\mathfrak{e}_1(\mathfrak{q}, \mathfrak{b}, x)$ и дополнительные конфигурации для представления $\mathfrak{e}_1$ и $\mathfrak{f}$.

Теперь, когда мы поняли смысл этих функций (*да, конечно*), Тьюринг неуклонно множит их число. Понимаю, что трудно тотчас увидеть, как все они соединятся вместе. Для создания своей Уни-

версальной машины Тьюрингу нужно несколько типовых функций, полезных для обработки отдельных символов и строк символов. Вы уже представляете себе функции поиска и стирания. Кроме них, ему нужны вырезание, копирование и вставка, а также несколько стандартных процедур печати.

Функция $\mathfrak{pe}$ означает «печатать в конце». Она печатает в первой пустой *F*-клетке символ, обозначенный β.

$\mathfrak{pe}(\mathfrak{C}, \beta)$		$\mathfrak{f}(\mathfrak{pe}_1(\mathfrak{C}, \beta), \mathfrak{C}, \mathfrak{ə})$	Из $\mathfrak{pe}(\mathfrak{C}, \beta)$ машина печатает β в конце последовательности символов и $\rightarrow \mathfrak{C}$.
$\mathfrak{pe}_1(\mathfrak{C}, \beta)$	Any *R*, *R* None $P\beta$	$\mathfrak{pe}_1(\mathfrak{C}, \beta)$ $\mathfrak{C}$	

В этой функции скрыты некоторые неявные предположения. Функция $\mathfrak{f}$ обычно ищет самое левое вхождение своего третьего аргумента, но здесь этот аргумент – тот самый символ $\mathfrak{ə}$, который $\mathfrak{f}$ ищет, чтобы добраться до левого края последовательности. Поэтому функция $\mathfrak{pe}$ предполагает наличие двух подряд символов $\mathfrak{ə}$, как во втором примере машины Тьюринга на странице 105. Сначала *m*-функция $\mathfrak{f}$ находит самый правый из двух $\mathfrak{ə}$ (один – в *E*-клетке), а затем сдвигает головку влево, чтобы стать слева от находящегося в *F*-клетке $\mathfrak{ə}$. Потом функция $\mathfrak{pe}_1$ перемещается вправо по *F*-клеткам, пока не найдет пустую. Она печатает β, который для большинства вычислительных машин будет или 0, или 1.

Следующие примеры остроумны. Сначала Тьюринг определяет функции $\mathfrak{l}$ (от *l*eft – влево) и $\mathfrak{r}$ (от *r*ight – вправо), а затем использует их вместе с $\mathfrak{f}$ для создания еще двух функций $\mathfrak{f}'$ и $\mathfrak{f}''$, которые перемещают головку влево или вправо после обнаружения заданного символа.

$\mathfrak{l}(\mathfrak{C})$	*L*	$\mathfrak{C}$	Из $\mathfrak{f}'(\mathfrak{C}, \mathfrak{B}, \alpha)$ делается то же самое, что из $\mathfrak{f}(\mathfrak{C}, \mathfrak{B}, \alpha)$, но перед этим делается шаг влево и $\rightarrow \mathfrak{C}$.
$\mathfrak{r}(\mathfrak{C})$	*R*	$\mathfrak{C}$	
$\mathfrak{f}'(\mathfrak{C}, \mathfrak{B}, \alpha)$		$\mathfrak{f}(\mathfrak{l}(\mathfrak{C}), \mathfrak{B}, \alpha)$	
$\mathfrak{f}''(\mathfrak{C}, \mathfrak{B}, \alpha)$		$\mathfrak{f}(\mathfrak{r}(\mathfrak{C}), \mathfrak{B}, \alpha)$	

Я бы назвал их *fl* и *fr*, а не $\mathfrak{f}'$ и $\mathfrak{f}''$, но это я.

Универсальной машине нужно будет перемещать символы на ленте с места на место. Функция $\mathfrak{C}$ выполняет «копирование». Символ α, скорее всего, должен быть меткой. Функция получает символ в *F*-клетке слева от этой метки и применяет $\mathfrak{pe}$ для копирования в первую пустую *F*-клетку в конце.

$\mathfrak{c}'(\mathfrak{C}, \mathfrak{B}, \alpha)$		$\mathfrak{f}'(\mathfrak{c}_1(\mathfrak{C}), \mathfrak{B}, \alpha)$	$\mathfrak{c}(\mathfrak{C}, \mathfrak{B}, \alpha)$. Машина пишет в конце первый символ, отмеченный α, и $\rightarrow \mathfrak{C}$.
$\mathfrak{c}_1(\mathfrak{C})$	β	$\mathfrak{p}\mathfrak{e}(\mathfrak{C}, \beta)$	

Заметьте, что функция применяет $\mathfrak{f}'$ для поиска символа α так, чтобы головка стала от него слева, где находится помеченная им цифра.

Функция $\mathfrak{c}_1$ имеет необычный синтаксис: вторым аргументом $\mathfrak{p}\mathfrak{e}$ становится текущий символ. Тьюринг говорит:

> [238]
> Последняя строка означает все множество строк, получаемых из нее заменой β любым символом, который может встретиться на ленте данной машины.

Например, если бы функция $\mathfrak{c}$ применялась только для копирования нулей и единиц, то $\mathfrak{c}_1$ на самом деле должна определяться как

$$\mathfrak{c}_1(\mathfrak{C}) \quad \begin{cases} 0 & \mathfrak{p}\mathfrak{e}(\mathfrak{C}, 0) \\ 1 & \mathfrak{p}\mathfrak{e}(\mathfrak{C}, 1) \end{cases}$$

Функция $\mathfrak{c}\mathfrak{e}$ означает «копировать и стирать» («**c**opy and **e**rase»). Она имеет версии с двумя и с тремя аргументами.

$\mathfrak{c}\mathfrak{e}(\mathfrak{C}, \mathfrak{B}, \alpha)$	$\mathfrak{c}(e(\mathfrak{C}, \mathfrak{B}, \alpha), \mathfrak{B}, \alpha)$	$\mathfrak{c}\mathfrak{e}(\mathfrak{B}, \alpha)$. Машина копирует с конца все символы, помеченные α, и стирает символы α; $\rightarrow \mathfrak{B}$.
$\mathfrak{c}\mathfrak{e}(\mathfrak{B}, \alpha)$	$\mathfrak{c}\mathfrak{e}(\mathfrak{c}\mathfrak{e}(\mathfrak{B}, \alpha), \mathfrak{B}, \alpha)$	

$\mathfrak{c}\mathfrak{e}$ с тремя аргументами сначала применяет $\mathfrak{c}$ для копирования самой левой помеченной α цифры, а затем применяет e, чтобы стереть метку. Версия $\mathfrak{c}\mathfrak{e}$ с двумя аргументами использует версию с тремя аргументами, чтобы скопировать первую цифру и стереть метку, а затем возвращается к версии с двумя аргументами. На самом деле все символы, помеченные α, копируются в конец ленты в первые доступные *F*-клетки. (Второй пример Тьюринга на странице 107 мог бы применить эту функцию для копирования группы единиц в конец ленты.)

Теперь, наверное, самое время поднять крайне неприятный вопрос *эффективности*. Тьюринг определяет функции, которые выглядят изящными и компактными, но фактически скрывают огромное количество действий. Для выполнения каждого копирования со стиранием функция $\mathfrak{c}$ применяет $\mathfrak{f}$ для поиска метки (и не забудьте, что $\mathfrak{f}$ каждый раз возвращается к ограничителю), а затем переходит к функции

e, которая снова применяет $\mathfrak{f}$ для поиска той же самой метки, чтобы ее стереть. Более эффективную схему имеет $\mathfrak{ce}$, стирающая перед копированием символа метку, как только та найдена. (В честь печально известной неэффективности машин Тьюринга термином «*тьюринговская трясина*» (tar-pit – дословно смоляная яма, *прим. пер.*) окрестили излишне универсальные компьютерные программы, которые гораздо дольше машут крыльями, чем летят.)

Но Тьюринга не интересуют мирские проблемы эффективности. В конце концов, машина – воображаемая. Если нужно, она может работать с частотой миллион «зетагерц», и никто не поймет, сколько бесполезных действий она выполняет.

Функция $\mathfrak{re}$ – «замена» («replace»). Предполагается, что аргументы α и β – метки. Функция находит самую левую метку α и заменяет ее на β. (Понятно, что α и β – метки, так как Тьюринг не допускает замены уже помеченных в *F*-клетках цифр.)

$\mathfrak{re}(\mathfrak{C}, \mathfrak{B}, \alpha, \beta)$ $\mathfrak{re}_1(\mathfrak{C}, \mathfrak{B}, \alpha, \beta)$	*E, Pβ*	$\mathfrak{f}(\mathfrak{re}_1(\mathfrak{C}, \mathfrak{B}, \alpha, \beta), \mathfrak{B}, \alpha)$ $\mathfrak{C}$	$\mathfrak{re}(\mathfrak{C}, \mathfrak{B}, \alpha, \beta)$. Машина заменяет первую α на β и $\rightarrow \mathfrak{C} \rightarrow \mathfrak{B}$, если α нет.

Версии с тремя аргументами заменяют все метки α на β:

$\mathfrak{re}(\mathfrak{B}, \alpha, \beta)$	$\mathfrak{re}(\mathfrak{re}(\mathfrak{B}, \alpha, \beta), \mathfrak{B}, \alpha, \beta)$	$\mathfrak{re}(\mathfrak{B}, \alpha, \beta)$. Машина заменяет все α на β; $\rightarrow \mathfrak{B}$.

Для единообразия первая строка пояснения справа должна иметь отступ.

Если вы уловили суть методологии Тьюринга и принцип именования, вы понимаете, что функция $\mathfrak{cr}$ – «копировать и заменить» («copy and replace»):

$\mathfrak{cr}(\mathfrak{C}, \mathfrak{B}, \alpha)$ $\mathfrak{cr}(\mathfrak{B}, \alpha)$	$\mathfrak{c}(\mathfrak{re}(\mathfrak{C}, \mathfrak{B}, \alpha, \alpha), \mathfrak{B}, \alpha)$ $\mathfrak{cr}(\mathfrak{cr}(\mathfrak{B}, \alpha), \mathfrak{ce}(\mathfrak{C}, \mathfrak{B}, \alpha, \alpha), \alpha)$	$\mathfrak{cr}(\mathfrak{B}, \alpha)$ отличается от $\mathfrak{ce}(\mathfrak{B}, \alpha)$ только тем, что символы α не стираются. *m*-конфигурация $\mathfrak{cr}(\mathfrak{B}, \alpha)$ возникает, когда на ленте нет символов «α».

Эти функции больше нигде не используются в статье Тьюринга.

Универсальной машине нужна возможность «найти и заменить», и Тьюринг далее приводит полстраницы функций, начинающиеся с символов $\mathfrak{cp}$ (**c**om**p**are – сравнить) и $\mathfrak{cpe}$ (**c**ompare and **e**rase – сравнить и удалить). Заключительные *m*-конфигурации этих функций настолько длины, что пояснения Тьюринга оказываются под таблицами, а не справа от них. (В первой строке есть опечатка. В колонке *заключ. m-конфиг.* нижний индекс 1 у $\mathfrak{C}$ должен быть запятой. Кроме того, в колонке *заключ. m-конфиг.* встречаются точки, которые не служат никаким целям.)

$\mathfrak{cp}(\mathfrak{C}, \mathfrak{A}, \mathfrak{E}, \alpha, \beta)$		$\mathfrak{f}'(\mathfrak{cp}_1(C_1\ \mathfrak{A}, \beta), \mathfrak{f}(\mathfrak{A}, \mathfrak{E}, \beta), \alpha)$
$\mathfrak{cp}_1(\mathfrak{B}, \beta)$	γ	$\mathfrak{f}'(\mathfrak{cp}_2(\mathfrak{C}, \mathfrak{A}, \gamma), \mathfrak{A}, \beta)$
$\mathfrak{cp}_2(\mathfrak{C}, \mathfrak{A}, \gamma)$	$\begin{cases}\gamma \\ \text{not } \gamma\end{cases}$	$\mathfrak{C}$ $\mathfrak{A}$

Сравниваются первый символ, помеченный α, и первый, помеченный β. Если нет ни α, ни β, $\rightarrow \mathfrak{E}$. Если есть оба и символы подобны $\rightarrow \mathfrak{C}$. Иначе $\rightarrow \mathfrak{A}$.

$\mathfrak{cpe}(\mathfrak{C}, \mathfrak{A}, \mathfrak{E}, \alpha, \beta)$ $\quad$ $\mathfrak{cp}(\mathfrak{e}(\mathfrak{e}(\mathfrak{C}, \mathfrak{C}, \beta), \mathfrak{C}, \alpha), \mathfrak{A}, \mathfrak{E}, \alpha, \beta)$

$\mathfrak{cpe}(\mathfrak{C}, \mathfrak{A}, \mathfrak{E}, \alpha, \beta)$ отличается от $\mathfrak{cp}(\mathfrak{C}, \mathfrak{A}, \mathfrak{E}, \alpha, \beta)$ тем, что в случае подобия символов α и β стираются.

$\mathfrak{cpe}(\mathfrak{A}, \mathfrak{E}, \alpha, \beta)$ $\quad$ $\mathfrak{cpe}(\mathfrak{cpe}(\mathfrak{A}, \mathfrak{E}, \alpha, \beta), \mathfrak{A}, \mathfrak{E}, \alpha, \beta)$.

$\mathfrak{cpe}(\mathfrak{A}, \mathfrak{E}, \alpha, \beta)$. Последовательность символов, помеченных α, сравнивается с последовательностью, помеченной β. $\rightarrow \mathfrak{C}$, если они подобны. Иначе $\rightarrow \mathfrak{A}$. Некоторые из символов α и β удаляются.

Под «подобием» Тьюринг понимает «совпадение».

Теперь Тьюринг исчерпал свой ресурс мнемонических имен функций, поскольку следующую он называет просто $\mathfrak{q}$, увы, так же, как вскоре он будет называть главным образом *m*-конфигурации. Еще хуже то, что позже он назовет эту функцию $\mathfrak{g}$.

Думаю, Тьюринг хотел назвать эту функцию *g*, а не $\mathfrak{q}$. Так же, как функция $\mathfrak{f}$ ищет первое (то есть самое левое) вхождение символа, эта функция ищет последнее (самое правое) вхождение символа. Это дает основание считать, что родственные функции $\mathfrak{f}$ и *g* должны обозначаться последовательными буквами. По этой причине, несмотря на то что в следующей таблице описывается функция $\mathfrak{q}$, я буду называть ее $\mathfrak{g}$.

Версия $\mathfrak{g}$ с одним аргументом движется вправо, пока не находит две подряд пустые клетки. Считается, что это самый правый конец ленты. Версия $\mathfrak{g}$ с двумя аргументами сначала использует $\mathfrak{g}$ с одним аргументом, а затем движется влево в поиске символа α.

$\mathfrak{q}(\mathfrak{C})$	Any	R	$\mathfrak{q}(\mathfrak{C})$	[239]
	None	R	$\mathfrak{q}_1(\mathfrak{C})$	$\mathfrak{q}_1(\mathfrak{C}, \alpha)$. Машина ищет последний символ вида α. $\rightarrow \mathfrak{C}$.
$\mathfrak{q}_1(\mathfrak{C})$	Any	R	$\mathfrak{q}(\mathfrak{C})$	
	None		$\mathfrak{C}$	
$\mathfrak{q}(\mathfrak{C}, \alpha)$			$\mathfrak{q}(\mathfrak{q}_1(\mathfrak{C}, \alpha))$	
$\mathfrak{q}_1(\mathfrak{C}, \alpha)$	α		$\mathfrak{C}$	
	not α	L	$\mathfrak{q}_1(\mathfrak{C}, \alpha)$	

Тьюринг заканчивает этот раздел несколькими различными функциями со знакомыми именами.

Вспомните функцию $\mathfrak{pe}$, которая печатала символ в последней F-клетке. Функция $\mathfrak{pe}_2$ печатает два символа в последних двух F-клетках:

$\mathfrak{pe}_2(\mathfrak{C}, \alpha, \beta)$	$\mathfrak{pe}(\mathfrak{pe}(\mathfrak{C}, \beta), \alpha)$	$\mathfrak{pe}_2(\mathfrak{C}, \alpha, \beta)$. Машина печатает $\alpha\ \beta$ в конце.

Подобно функции $\mathfrak{ce}$, которая копировала помеченные α символы в конец, функция $\mathfrak{ce}_2$ копирует символы, помеченные α и β, тогда как $\mathfrak{ce}_3$ копирует символы, помеченные α, β и γ.

$ce_2(\mathfrak{B}, \alpha, \beta)$	$\mathfrak{ce}(\mathfrak{ce}(\mathfrak{B}, \beta), \alpha)$	$\mathfrak{ce}_3$ ($\mathfrak{B}$, α, β, γ). Сначала машина копирует в конец символы, помеченные α, затем – помеченные β, и, наконец –помеченные γ, символы α, β, γ она стирает.
$\mathfrak{ce}_3(\mathfrak{B}, \alpha, \beta, \gamma)$	$\mathfrak{ce}(\mathfrak{ce}_2(\mathfrak{B}, \beta, \gamma), \alpha)$	

Копирования выполняются последовательно: сначала копируются все символы, помеченные α, затем – символы, помеченные β, и т. д. Позже Тьюринг применит функцию $\mathfrak{ce}_5$ с шестью аргументами, которую нигде не описывал, но действия ее должны быть очевидны.

Наконец, функция $\mathfrak{e}$ с единственным аргументом стирает все метки.

$\mathfrak{e}(\mathfrak{C})$	ə	R	$\mathfrak{e}(\mathfrak{C})$	Из $\mathfrak{e}(\mathfrak{C})$ стираются метки всех помеченных символов. $\rightarrow \mathfrak{C}$.
	Not ə	L	$\mathfrak{e}(\mathfrak{C})$	
$\mathfrak{e}_1(\mathfrak{C})$	Any	R, E, R	$e_1(\mathfrak{C})$	
	None		$\mathfrak{C}$	

Программисты определенного возраста могут помнить книгу изобретателя языка программирования Паскаль Никлауса Вирта (Niklaus Wirth) (род. 1934) с прекрасным названием *Алгоритмы + Структуры данных = Программы* (Prentice-Hall, 1975)[1]. Как следует из названия, компьютерной программе нужны как код (алгоритм), так и некоторые данные для их обработки кодом. Здесь Тьюринг привел множество алгоритмов, которые нужны его Универсальной вычислительной машине, но еще не описал, как преобразовать произвольную вычислительную машину в данные для обработки. Об этом – дальше.

[1] Одно их последних изданий этой книги: Вирт Н. *Алгоритмы и структуры данных. Новая версия для Оберона* / пер. с англ. Ф. В. Ткачева. – М.: ДМК Пресс, 2010 (*прим. пер.)*.

Глава 8

Всё есть число

В наш цифровой век стало привычным представлять любую информацию числами. Текст, рисунки, фотографии, звук, музыка, кинофильмы – все проходит через жернова оцифровки и отправляется на хранение в наши компьютеры и другие устройства в виде сложнейших конструкций из нулей и единиц.

Однако в 1930-е годы числа были лишь числами, и если кто-то превращал текст в числа, то делал это в целях дезинформации и интриг.

Второй учебный год Алана Тьюринга в Принстоне осенью 1937-го начинался в атмосфере нарастающей тревоги, что Англия и Германия скоро окажутся в состоянии войны. Он, конечно, работал над своей докторской диссертацией, но проявлял интерес и к криптологии – математической науке о создании секретных кодов или шифров (криптография) и взламывании чужих кодов (криптанализ)[1]. Тьюринг считал, что в военное время лучше всего кодировать слова сообщения двоичными цифрами, умножая их затем на большие числа. Тогда расшифровка сообщения без знания этого большого числа свелась бы к сложной задаче разложения на множители. Эта идея Тьюринга была довольно дальновидной, поскольку сегодня так работает большинство программ шифрования.

В отличие от большинства математиков, Тьюрингу нравилось делать вещи своими руками. Для реализации автоматической кодовой машины он начал строить устройство двоичного умножения, используя электромагнитные реле, которые были основными конструктивными элементами компьютеров до того, как свою достаточную надежность доказали вакуумные лампы. Тьюринг даже собирал в механической мастерской свои собственные реле и сам делал обмотку электромагнитов.

В армии и флоте Германии уже использовалось совсем другое шифровальное устройство. «*Энигма*» (*Enigma*) была изобретена не-

---

[1] Andrew Hodges, *Alan Turing: The Enigma* (Simon & Schuster, 1983), 138.

мецким инженером-электриком Артуром Шербиусом (1878–1929). В 1918 году после неудачной попытки Шербиуса убедить ВМФ Германии использовать его машину, в 1923 году она стала продаваться в коммерческих целях. В итоге вскоре после этого ВМФ, вслед за остальной частью немецких вооруженных сил, заинтересовался ею[1].

«Энигма» имела простую клавиатуру с 26-ю клавишами, расположенными так же, как на пишущей машинке, только без цифр, знаков препинания и клавиши SHIFT. Над клавиатурой было 26 лампочек, расположенных в том же порядке. Сообщения шифровались при наборе символов на клавиатуре. При нажатии любой буквы высвечивалась другая буква. Высвеченные буквы вручную переписывались и затем отправлялись получателю. (Шифрованное сообщение можно было передать в руки или отправить почтой; позже шифровки передавали по радио, используя азбуку Морзе.) У получателя сообщения была своя машина «Энигма», и он должен был набрать шифровку на клавиатуре. При этом загорающиеся лампочки побуквенно воспроизводили исходный текст.

Клавиши клавиатуры соединялись электрической цепью с лампочками через ряд роторов. Каждый ротор был небольшим диском, на обеих сторонах которого было по 26 контактов для каждой буквы алфавита. Контакты в роторе соединялись симметрично: если контакт A на одной стороне соединялся с контактом T на другой, то T на первой стороне соединялся с A на второй. Эта симметрия позволяла использовать машину как для шифровки, так и для расшифровки.

Обычная «Энигма» имела три соединенных ротора, каждый из которых был «прошит» по-разному и мог быть установлен в одно из 26 положений. Три ротора на шифрующей и дешифрующей машинах должны были устанавливаться одинаково. Трехбуквенные ключи для установки роторов могли меняться, скажем, ежедневно согласно списку, известному только операторам «Энигмы».

Пока ничто из того, что я написал об «Энигме», не говорит о ее способности к чему-то большему, чем шифрование простой заменой одной буквы на другую, которое легко взламывается даже криптоаналитиком-любителем. Такой шифр даже проще многих шифров-подстановок, поскольку он симметричен: если D заменяется на S, то S заменяется на D.

---

[1] David Kahn, *Seizing the Enigma: The Race to Break the German U-Boat Codes, 1939–1943* (Houghton-Mifflin, 1991), ch. 3.

Но вот в чем была изюминка: когда пользователь «Энигмы» нажимал клавишу на клавиатуре, роторы *вращались*. При каждом нажатии клавиши первый ротор проворачивался вперед на одну позицию. Если, например, набиралась строка из 26 букв А, то каждая следующая буква А шифровалась по-другому, так как ротор проходил через 26 позиций. Совершив полный оборот, первый ротор проворачивал второй ротор вперед на одну позицию. Следующая строка из 26 букв А теперь кодировалась бы другой последовательностью букв. Совершив свой полный оборот, второй ротор цеплял зубец третьего ротора. Неподвижный четвертый ротор посылал электрический сигнал назад по роторам в обратном порядке. Шаблон шифра мог повторяться только после 17 576 (это 26^3) нажатий клавиш.

Дальше – больше: роторы были сменными. Основная машина поставлялась с пятью различными роторами, которые могли использоваться в любом из трех разъемов для роторов. Еще одно дополнение включало коммутационную панель, которая добавляла еще один уровень шифрования букв.

В 1932 году три польских математика начали разрабатывать методы декодирования сообщений «Энигмы»[1]. Они решили, что им нужно построить устройство, автоматически моделирующее «Энигму». Первые «бомбы» (как называли эти устройства) заработали в 1938 году и перебирали возможные варианты установки роторов. Одним из этих математиков был Мариан Реджевски (1905–1980), который провел год в Гёттингене после его окончания. Он писал, что машины «за неимением лучшего названия»[2] называли бомбами, возможно, что название было подсказано тиканьем, которое они издавали, или особым сортом любимого математиками мороженого с фруктами и орехами[3].

Для взлома шифров британское правительство обычно приглашало ученых-языковедов, справедливо полагая, что они лучше других подготовлены для расшифровки сложных языков. С приближением войны становилось понятно, что для анализа изощренных шифраторов вроде «Энигмы» Государственной школе кодирования и шифро-

---

[1] Marian Rejewski, «How Polish Mathematicians Deciphered the Enigma», *Annals of the History of Computing*, Vol. 3, No. 3 (July 1981), 213–234. См. также Elisabeth Rakus-Andersson, «The Polish Brains Behind the Breaking of the Enigma Code Before and During the Second World War», in Christof Teuscher, ed., *Alan Turing: Life and Legacy of a Great Thinker* (Springer, 2004), 419–439.

[2] Реджевски, «How Polish Mathematicians Deciphered the Enigma», 226.

[3] Kahn, Seizing the Enigma, 73.

вания (Government Code and Cypher School, GC&CS) нужны были еще и математики.

Когда летом 1938 года Алан Тьюринг вернулся из Принстона в Англию, его пригласили прочитать курс в главном управлении GC&CS. Возможно, правительство поддерживало связь с ним еще до 1936 года[1]. В 1939 году GC&CS приобрела большое имение под названием Блетчли-Парк (Bletchley Park) с особняком в викторианском стиле в 50 милях северо-восточнее Лондона. В некотором смысле Блетчли-Парк был интеллектуальным центром Англии – местом, где железная дорога между Оксфордом и Кембриджем соединялась с дорогой на юг к Лондону.

1 сентября 1939 года Германия вторглась в Польшу. Два дня спустя Великобритания объявила войну Германии, а 4 сентября Алан Тьюринг прибыл на службу в Блетчли-Парк. Перехватом и расшифровкой секретной корреспонденции там занималось в общей сложности около десяти тысяч человек. Чтобы разместить всех, по всей территории были построены бараки. Тьюринг руководил бараком № 8, отвечавшим за расшифровку кодов немецкого военного флота. Немцы использовали эти коды для связи с подводными лодками – главной угрозой конвоям, курсировавшим между США и Великобританией в Атлантике.

В начале 1939 года британцы встречались с польскими математиками для изучения «Энигмы» и «бомб». Приступив к работе в Блетчли-Парке, Тьюринг вскоре начал перепроектировать и совершенствовать устройства, называемые теперь «бомбами», но на французский лад – *bombe*. Первая «Бомба» Тьюринга (как их порой называли) заработала в 1940 году. Она весила тонну и могла имитировать одновременную работу тридцати «Энигм»[2].

Прежде чем нанести удар по шифровке «Бомбой» Тьюринга, нужно было сократить число вариантов перебора. Криптоаналитики искали «ключи» – общие слова или фразы, часто встречавшиеся в шифровках. Они позволяли выяснить исходное положение первого ротора. Большую ценность представляли случаи, когда одно и то же сообщение передавалось с использованием двух шифров: они были известны как «поцелуи». В другом методе применялась плотная белая бумага

---

[1] Hodges, *Alan Turing*, 148.

[2] Stephen Budiansky, *Battle of Wits: The Complete Story of Codebreaking in World War II* (Free Press, 2000), 155. См. также Jack Gray and Keith Thrower, *How the Turing Bombe Smashed the Enigma Code* Speedwell, 2001).

различной ширины с напечатанными на ней строками из букв алфавита наподобие перфокарт, применявшихся позже в компьютерах. Аналитики пробивали в бумаге отверстия, соответствующие буквам шифровок. Все шифровки одного дня (составленные при одних и тех же настройках «Энигмы») можно было потом сравнивать наложением бумажных полос. Поскольку бумага, используемая для этого, поставлялась из соседнего города под названием Банбери (Banbury), этот процесс называли «банберизмом».

Эти разнообразные технические приемы были усовершенствованы до такой степени, что к середине 1941 года успехи, достигнутые в расшифровке текстов «Энигмы», заметно сократили военно-морские потери[1]. Многие из работавших в Блетчли-Парке заслуживают признания за этот успех, однако именно работа Алана Тьюринга сыграла существенную роль.

Даже в необычном сообществе математиков и языковедов в Блетчли-Парке Тьюринг заработал определенную репутацию чудака:

> Каждый год в первую неделю июня его [Тьюринга] поражала неприятная сенная лихорадка, и он приезжал на службу на велосипеде, надев противогаз, дабы защититься от пыльцы. Его велосипед был неисправен: цепь срывалась через одинаковые промежутки времени. Вместо того чтобы починить его, он считал обороты педалей, вовремя слезал с велосипеда и руками подправлял цепь[2].

Весной 1941 года Алан Тьюринг сделал предложение Джоан Кларк, одной из редких женщин в Блетчли-Парке, которую не отправили на бессмысленную канцелярскую работу. Джоан Кларк изучала математику в Кембридже, когда была принята на работу по взлому шифров. Спустя несколько дней после предложения Тьюринг признался ей, что у него были «гомосексуальные склонности»[3], но встречи продолжались еще несколько месяцев, пока он не почувствовал, что должен их прекратить.

В ноябре 1942 года Тьюринг был направлен в Вашингтон округа Колумбия с заданием помочь скоординировать деятельность Англии и Соединенных Штатов по взлому шифров. После этого назначения он провел первые два месяца 1943 года в Bell Labs, расположенной в то время на Западной улице (West Street) Нью-Йорка. Там он встретился с Гарри Найквистом (Harry Nyquist) (1889–1976), пер-

---

[1] Hodges, *Alan Turing*, 218–219.
[2] I. J. Good, «Early Work on Computers at Bletchley», *Annals of the History of Computing*, Vol. 1, No. 1 (July 1979), 41.
[3] Hodges, *Alan Turing*, 206.

вопроходцем в теории цифровой дискретизации, и Клодом Элвудом Шэнноном (Claude Elwood Shannon) (1916–2001), чья статья «Математическая теория связи» (1948) заложила основы теории информации и явила миру слово «бит».

Основной интерес для Тьюринга в Bell Labs представляло устройство шифрования речи, которое предназначалось для обеспечения надежной телефонной связи через Атлантику. Звуковые волны делились на различные частотные диапазоны, оцифровывались, а затем шифровались посредством сложения по модулю определенного значения (как, например, значения 60 при сложении секунд и минут). На стороне приемника числа расшифровывались, и затем воспроизводилась речь.

В исследовании Найквиста и работе Шеннона, а также в устройстве шифрования речи можно увидеть ростки идей, которые позже приведут к технологиям оцифровки изображений в файлах JPEG и звука в файлах MP3, однако для их осуществления потребовались десятилетия. С другой стороны, первые компьютеры выдавали не более чем числа. Даже самая первая Разностная машина Бэббиджа была задумана исключительно для печати точных таблиц логарифмов. В этой связи неудивительно, что машины Тьюринга тоже генерируют числа, а не реализуют, например, обобщенные функции.

Тьюринг подошел в статье к тому, чтобы взять курс на более необычное использование чисел для кодирования других видов информации. В следующем разделе статьи Тьюринга показывается, как можно представить числами не фотографии или песни, а сами машины.

Да, все есть число. Даже машины Тьюринга – это числа.

5. *Перечисление вычислимых последовательностей.*

Вычислимая последовательность γ определяется описанием машины, которая вычисляет γ. Таким образом, последовательность 001011011101111... определяется таблицей на с. 234, и фактически любая вычислимая последовательность может быть описана такой таблицей.

Это – пример II машины на странице 107 этой книги.

Будет полезно придать этим таблицам некоторую стандартную форму.

На самом деле Тьюринг начинал со стандартной формы, которую описал в разделе 1 (страница 91 этой книги). Он требовал, чтобы отдельная операция могла заставить машину напечатать или стереть символ и сместиться на одну клетку влево или вправо. После демонстрации одной машины в таком формате (пример I на странице 101, который Тьюринг тоже скоро вспомнит) Тьюринг быстро отказался от своих собственных правил. Он разрешил в одной операции печатать несколько символов и перемещаться на несколько клеток. Это было сделано исключительно для того, чтобы машинные таблицы не растягивались на несколько страниц. Теперь он хотел бы вернуться к своим исходным требованиям.

Первым делом предположим, что таблица задается в том же виде, что и первая таблица, например I на с. 233. То есть что элемент в столбце операций всегда имеет одну из форм *E: E, R: E, L: Pα: Pα, R: Pα, L: R: L:* или вообще пуст.

Для разделения девяти различных вариантов Тьюринг использует двоеточия. Эти варианты – результат сочетания трех типов печати (стереть, напечатать символ или ничего) с тремя типами движения (влево, вправо или на месте).

Таблица всегда может быть приведена к такому виду путем введения дополнительных *m*-конфигураций.

Например, таблица из примера II (страница 107) начиналась с конфигурации 𝔟:

Конфигурация		*Поведение*	
m-конфиг.	*символ*	*операции*	*заключит. m-конфиг.*
𝔟		*P*ə, *R*, *P*0, *R*, *P*0, *R*, *R*, *P*0, *L*, *L*	𝔬

Чтобы отвечать исходным (и восстановленным) требованиям Тьюринга, эта единственная конфигурация должна быть разбита на шесть простых. Для дополнительных конфигураций я буду использовать немецкие строчные буквы *c*, *d*, *e*, *g* и *h* (*f* уже использована в исходной таблице).

Конфигурация		*Поведение*	
m-конфиг.	*символ*	*операции*	*заключит. m-конфиг.*
𝔟		*P*ə, *R*	𝔠
𝔠		*P*ə, *R*	𝔡
𝔡		*P*0, *R*	𝔢
𝔢		*R*	𝔤
𝔤		*P*0, *L*	𝔥
𝔥		*L*	𝔬

Теперь каждая операция состоит исключительно из операции печати (или нет), за которой могут следовать сдвиг влево или вправо на одну клетку.

Давайте теперь дадим номера *m*-конфигурациям, назвав их $q_1, \ldots, q_R$, как в §1. Начальной *m*-конфигурацией всегда должна быть q_1.

Если случится, что в машине 237 различных *m*-конфигураций, они должны теперь обозначаться от q_1 до q_{237}.

В измененном начале примера II первые шесть *m*-конфигураций можно переименовать от q_1 до q_6. Начальной *m*-конфигурацией, которую Тьюринг всегда называл 𝔟, становится q_1. Теперь таблица выглядит так:

Конфигурация		*Поведение*	
m-конфиг.	*символ*	*операции*	*заключит. m-конфиг.*
q_1		*P*ə, *R*	q_2
q_2		*P*ə, *R*	q_3
q_3		*P*0, *R*	q_4
q_4		*R*	q_5
q_5		*P*0, *L*	q_6
q_6		*L*	q_7

[240]

М ы также дадим номера символам $S_1, \ldots, S_m$ и, в частности, пусто = S_0, 0 = S_1, 1 = S_2.

Несколько смущает то, что символу 0 назначен индекс 1, а символу 1 – индекс 2, но мы должны смириться с этим. Машина примера II должна печатать также ə и *x*, поэтому для нее должны быть определены следующие обозначения:

- S_0 означает пусто,
- S_1 означает 0,
- S_2 означает 1,
- S_3 означает ə, и
- S_4 означает x.

Машине, вычисляющей квадратный корень из 2, нужны символы вплоть до S_{14}.

Первые шесть конфигураций машины примера II теперь таковы:

Конфигурация		*Поведение*	
т-конфиг.	*символ*	*операции*	*заключит. т-конфиг.*
q_1		PS_3, R	q_2
q_2		PS_3, R	q_3
q_3		PS_1, R	q_4
q_4		R	q_5
q_5		PS_1, L	q_6
q_6		L	q_7

Результат унификации именования – эти строки, содержащие очень простые шаблоны. В итоге Тьюринг выделяет три стандартных вида:

Строки таблицы теперь имеют вид

т-конфиг.	*Символ*	*Операции*	*Заключит. т-конфиг.*	
q_i	S_j	PS_k, L	q_m	(N_1)
q_i	S_j	PS_k, R	q_m	(N_2)
q_i	S_j	PS_k	q_m	(N_3)

Эти три стандартных вида Тьюринг пометил справа как N_1, N_2 и N_3. Все три что-то печатают; разница только в том, что головка сдвигается влево (L), вправо (R) или стоит на месте.

А где же стирание? Поскольку Тьюринг определил S_0 как пустой символ, стирание может выполняться просто как печать S_0:

Строки вида

$q_i \quad S_j \quad E, R \quad q_m$

должны записываться как

$q_i \quad S_j \quad PS_0, R \quad q_m$

Операции сдвига вправо или влево без печати можно считать той же операцией, но с печатью просматриваемого символа:

а строки вида

$q_i \quad S_j \quad R \quad q_m$

должны записываться как

$q_i \quad S_j \quad PS_j, R \quad q_m$

Таким образом, мы привели каждую строку таблицы к строке одного из видов (N_1), (N_2), (N_3).

Для иллюстрации процесса стандартизации таблицы я использовал первую конфигурацию таблицы примера II, но в колонке *символ* этой конфигурации ничего нет, потому что данная конфигурация делает одно и то же независимо от символа. Машина начинает с пустой ленты, поэтому мы знаем, что читаемый ею символ – пустой. Первая конфигурация таблицы примера II, преобразованная к стандартной форме, становится такой:

Конфигурация		*Поведение*	
т-конфиг.	*символ*	*операции*	*заключит. т-конфиг.*
q_1	S_0	PS_3, R	q_2
q_2	S_0	PS_3, R	q_3
q_3	S_0	PS_1, R	q_4
q_4	S_0	R	q_5
q_5	S_0	PS_1, L	q_6
q_6	S_0	L	q_7

Это достаточно просто, но давайте рассмотрим вторую *m*-конфигурацию машины примера II:

ɒ 1 *R, Px, L, L, L* ɒ
0 ɋ

M-конфигурация $\mathfrak{o}$ станет нумерованной конфигурацией q_7. Если просматриваемый символ – 1, головка должна сдвинуться один раз вправо, а затем трижды влево. Эти три сдвига влево потребуют еще трех *m*-конфигураций q_8, q_9 и q_{10}. Тогда *m*-конфигурация $\mathfrak{q}$ станет q_{11}. Вот *m*-конфигурация q_7:

Конфигурация		*Поведение*	
m-конфиг.	*символ*	*операции*	*заключит. m-конфиг.*
q_7	S_2	PS_2, R	q_8
q_7	S_1	PS_1	q_{11}

В обоих случаях машина печатает просматриваемый символ. Вот *m*-конфигурации q_8, q_9 и q_{10}:

q_8	S_0	PS_4, R	q_8
q_9	S_2	PS_2, L	q_{12}
q_{10}	S_0	PS_0, L	q_7

Проблема – в колонке *символ*. Чтобы заполнить ее правильно, нам на самом деле нужно знать, с чем столкнется машина. В q_8 машина видит пустую клетку и печатает *x*. А что она увидит, сдвинувшись влево? Как раз символ 1, который просматривался в q_7, но в других случаях это не так очевидно. При таком подходе слова «Any», «Not» или «Else» не работают, и в каждом из таких случаев для каждого используемого машиной символа придется добавлять отдельные конфигурации.

Это неприятно, но число символов всегда конечно, поэтому это, безусловно, придется делать. Допустим, мы привели все конфигурации некоторый машины к стандартным формам, которые Тьюринг обозначил (*N*1), (*N*2) и (*N*3). Покончив с этим и избавившись от исходной таблицы, не потеряли ли мы какую-нибудь информацию? Да, немного потеряли. Мы знаем, что S_0 – пусто, S_1 – 0 и S_2 – 1, но мы уже не знаем, что именно обозначается символами S_3, S_4 и т. д. Однако это не важно. Машины используют эти символы в своих целях. Важно только то, что они уникальны. Нас, в сущности, интересуют лишь нули и единицы, которые печатает машина, а не то, что она использует временно.

Каждую конфигурацию таблицы можно представить комбинацией *m*-конфигураций, символов, а также *L* и *R*.

Давайте из каждой строки вида (N_1) образуем выражение вида $q_i S_j S_k L q_m$;

Такое представление иногда называют *пятеркой*, потому что оно состоит из пяти элементов. Несмотря на свою загадочную природу, она легко читается: «если в m-конфигурации q_i просматривается символ S_j, то напечатать символ S_k, сдвинуться влево и изменить m-конфигурацию на q_m». Аналогично для N_2 и N_3:

из каждой строки вида (N_2) образуем выражение вида $q_iS_jS_kRq_m$; а из каждой строки вида (N_3) образуем выражение вида $q_iS_jS_kNq_m$.

Обратите внимание: когда головка стоит на месте, используется буква *N* (от *N*o move – без движения).

Выпишем из таблицы машины все образованные таким образом выражения и разделим их точками с запятой. Таким образом, мы получаем полное описание машины.

Пример Тьюринг приведет совсем скоро. Каждая конфигурация – это пятерка, а машина в целом теперь представляет собой поток пятерок. (Довольно интересно, что пятерки не требуют какого-то определенного порядка. Они похожи на язык программирования, в котором каждый оператор начинается с метки и заканчивается *goto*.)

Следующая замена радикальна. Она избавляет описание машины от всех нижних индексов и превращает его в поток прописных букв:

В этом описании мы заменим q_i буквой «*D*», за которой следует буква «*A*», повторенная *i* раз, а S_j – на «*D*», за которой следует «*C*», повторенная *j* раз.

Например, q_1 заменяется на *DA*, а q_5 заменяется на *DAAAAA*. (Не забывайте, что первая конфигурация – q_1. Не q_0.) Аналогично символ S_0 (пусто) теперь обозначается *D*, S_1 (символ 0) – *DC*, а S_2 (символ 1) – *DCC*. Остальные символы – S_3 и далее – заменяются на *DCCC* и т. д.

Это новое описание машины можно назвать *стандартным описанием* (S.D). Оно составлено только из букв «*A*», «*C*», «*D*», «*L*», «*R*», «*N*» и «;».

Буквами *L*, *R* и *N* обозначаются перемещения. Конфигурации разделяются точками с запятой.

Наконец, если мы меняем «*A*» на «1», «*C*» на «2», «*D*» на «3», «*L*» на «4», «*R*» на «5», «*N*» на «6» и «;» на «7», у нас будет описание машины в виде арабских цифр.

Это шаг очень важен. Тьюринг стандартизировал свои машины до такой степени, что может теперь однозначно определить машину целым числом, и в этом целом числе закодированы все состояния машины. Несомненно, Тьюринг был вдохновлен подходом Гёделя, взявшегося в своей Теореме о неполноте преобразовать любое математическое выражение в уникальный номер.

Целое число, представленное этими цифрами, можно назвать *описательным номером* (далее *описатель – прим. пер.*) (D.N) машины. D.N однозначно определяет S.D [241] и структуру машины. Машину, чей D.N равен *n*, можно обозначить как $\mathcal{M}(n)$.

Теперь Тьюринг применил другой шрифт. Он будет использовать этот рукописный шрифт для обозначения машин в целом.

Каждой вычислимой последовательности соответствует хотя бы один описательный номер, тогда как ни одному описательному номеру не соответствует более одной вычислимой последовательности.

Так как порядок следования пятерок не имеет значения, они могут кодироваться, не оказывая влияния на последовательность, которую вычисляет машина. Отсюда ясно, что с каждой вычислимой последовательностью связано множество описателей, но каждый описатель определяет машину, которая генерирует только одну вычислимую последовательность (во всяком случае, когда работа начинается с пустой ленты).

Без лишнего пафоса Тьюринг заканчивает выводом, о котором он упоминал в самом начале статьи:

Следовательно, вычислимые последовательности и числа перечислимы.

Вы можете пронумеровать вычислимые последовательности, перечислив все возможные описатели, так как они – просто целые числа. Отсюда неявно следует, что вычислимые числа – это лишь перечислимое подмножество вещественных чисел. Поскольку вычислимые числа перечислимы, а вещественные числа нет, существует множество невычислимых вещественных чисел. Но эта тема будет исследована глубже в следующих разделах.

Давайте найдем описатель машины I из §3.

Эта машина изначально определялась следующей таблицей:

Конфигурация		*Поведение*	
т-конфиг.	*символ*	*операции*	*заключит. т-конфиг.*
𝔟	None	$P0, R$	𝔠
𝔠	None	R	𝔢
𝔢	None	$P1, R$	𝔣
𝔣	None	R	𝔟

Если мы переименуем *т*-конфигурации, получим таблицу:

q_1	S_0	PS_1, R	q_2
q_2	S_0	PS_0, R	q_3
q_3	S_0	PS_2, R	q_3
q_4	S_0	PS_0, R	q_1

Это очень простое преобразование.

Прочие таблицы можно было бы получить, добавляя бесполезные строки, такие как

q_1	S_1	PS_1, R	q_2

Таким образом, *прочие таблицы, порождающие ту же самую вычислимую последовательность*, можно было бы получить, добавляя строки, которые не играют никакой роли. Если машина начинает работу с пустой ленты и всегда перемещается вправо после печати в клетке, она никогда не встретит цифру 0.

Наша первая стандартная форма была бы

$$q_1 S_0 S_1 R q_2; q_2 S_0 S_0 R q_3; q_3 S_0 S_2 R q_4; q_4 S_0 S_0 R q_1; .$$

Здесь показывается только таблица из четырех строк, а конфигурации разделяются точками с запятой. Преобразование этого к стандартному описанию (S.D) требует замены q_i на D с последующими A в количестве i (один или больше) и замены S_j на D с последующими C в количестве j (нуль или больше).

Стандартное описание будет таким

DADDCRDAA; DAADDRDAAA;
DAAADDCCRDAAAA; DAAAADDRDA;

Стандартное описание может быть трудным для чтения, но оно часто используется, поэтому нужно постараться привыкнуть к нему. Чтобы разбить его на составляющие, нужно обращать внимание на каждую букву *D*. Каждая буква *D* представляет либо конфигурацию, либо символ.

- Если за *D* следуют одна или более букв *A*, то это – конфигурация. Номер конфигурации – количество *A*.
- Если вслед за *D нет A*, то это – символ. В этом случае за *D* могут следовать буквы *C*. *D* сама по себе – это пусто, *DC* – 0, *DCC* – 1, а большее количество *C* означает прочие символы.

Тьюринг не использует описатель так же часто, как стандартное описание. Описатель – это скорее абстракция; Тьюринг не выполняет никаких вычислений с числом, представляющим описатель. В качестве примера Тьюринг показывает, что для создания описателя можно заменить *A* на 1, *C* на 2, *D* на 3, *R* на 5, а точку с запятой – на 7:

Описатель – это

3133253117311335311173111332253111173111133531731323253117

а также

3133253117311335311173111332253111173111133531731323253117

Второй из этих номеров – тот же, что и первый, но с дополнительными цифрами в конце (31323253117), соответствующими «бесполезной» конфигурации $q_1S_1S_1Rq_2$, которую определил Тьюринг. Суть вот в чем: эти два описателя определяют две разные машины, но обе они вычисляют абсолютно одно и то же число, которое (как вы помните) есть двоичное представление 1/3. Машина с переставленными конфигурациями по-прежнему вычисляет то же самое число, но у нее другой описатель.

Эти номера огромны! Тьюринга явно не заботит, насколько они велики. Для представления q_{35}, например, он мог бы придумать какой-нибудь способ вставить число 35 в описатель, но нет. Для представления q_{35} в стандартном описании используется:

DAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA,

а описатель включает цифры

3111111111111111111111111111111111,

причем не один раз, а минимум дважды!

Достигнутое здесь весьма интересно. Рассмотрим машину Тьюринга, которая вычисляет π. Обычно мы изображаем число π бесконечной последовательностью:

π = 3.14159265358979323846264338327 95...

Теперь же мы можем представить π *конечным* целым числом – описателем машины Тьюринга, вычисляющей эти цифры. Какое из представлений π лучше? Первые 32 цифры с последующим многоточием? Или описатель машины Тьюринга, которая может сгенерировать столько цифр, сколько позволит нам наше терпение? В каком-то смысле описатель – более фундаментальное числовое представление π, поскольку описывает алгоритм вычисления числа.

Сведя каждую машину к описателю, Тьюринг на самом деле дал возможность создавать машины лишь перечислением положительных целых чисел. Не каждое положительное целое число – допустимый описатель машины Тьюринга, и многие допустимые описатели не описывают ациклические машины, но это перечисление, конечно, включает все ациклические машины Тьюринга, каждая из которых соответствует вычислимому числу. Следовательно, вычислимые числа перечислимы.

Это очень важный результат, хотя, наверное, тревожный, ибо это значит, что все – а из того, что мы знаем о пространстве вещественных чисел, *практически все* – вещественные числа невычислимы.

Это открытие вместе с некоторыми математическими парадоксами и исследованиями в квантовой гравитации побудили математика Грегори Чейтина (Grerory Chaitin) задаться вопросом «Насколько вещественны вещественные числа?»[1]. Подтверждения существования вещественных чисел действительно скудны.

---

[1] Gregory J. Chaitin, «How Real are Real Numbers?», *International Journal of Bifurcation and Chaos*, Vol. 16 (2006), 1841–1848. Перепечано в Gregory J. Chaitin, *Thinking About Gödel and Turing: Essays on Complexity, 1970–2007* (World Scientific, 2007), 267–280.

Современным программистам легко представить компьютерные программы в виде чисел, потому что исполняемый файл программы – это просто массив байтов. Обычно мы не считаем эти байты единым числом, но могли бы, конечно. Например, исполняемый файл WinWord.exe Microsoft Word 2003 имеет размер 12 047 560 байтов. Это более 96 миллионов битов или 29 миллионов десятичных цифр, поэтому номер, представляющий WinWord.exe, находится где-то в районе $10^{29000000}$. Это, конечно, большое число. В книге, где примерно 50 строк на странице и 50 цифр в строке, это число растянулось бы более чем на 11 000 страниц. Это целое число гораздо больше, чем знаменитый гугол (googol) (10^{100}), но все же – конечное. WinWord.exe – одна из многих возможных исполняемых программ (подобно всевозможным машинам Тьюринга), которая попала в перечень целых чисел, наряду с любой другой программой обработки текста, даже той, что еще не написана.

Тьюринг заканчивает этот раздел определением, которое понадобится позже.

Число, которое является описателем ациклической машины, будем назвать *приемлемым* числом. В §8 показывается, что не может быть общего процесса для определения того, что данный номер приемлем или нет.

Легко определить, является ли некоторое целое число правильным описателем, но Тьюринг теперь утверждает, что не существует общего процесса для определения того, что данный описатель представляет ациклическую машину и печатает предполагаемую бесконечную последовательность нулей и единиц. Не существует общего процесса для определения того, что машина может встретить символ, которого не ждет, зациклится, печатая пробелы, сломается, сгорит, переполнится или попадает «пальцем в небо».

Глава 9

Универсальная машина

Машина, которую Тьюринг описывает в следующем разделе своей статьи, известна сегодня как Универсальная машина Тьюринга, названная так потому, что это – единственная необходимая нам машина. Представленные ранее отдельные вычислительные машины не обязательно должны иметь аналогичные реализации или взаимозаменяемые части. А вот Универсальная машина может моделировать другие машины, если ей предоставить их описания. Как сказали бы сегодня, Универсальная машина *программируема*.

> 6. *Универсальная вычислительная машина*.
>
> Можно создать одну-единственную машину, которую можно использовать для вычисления любой вычислимой последовательности. Если этой машине $\mathcal{U}$ предоставить ленту, в начале которой запи- [242]
> сано S.D некоторой вычислительной машины $\mathcal{M}$, то $\mathcal{U}$ вычислит ту же самую последовательность, что и $\mathcal{M}$. В этом разделе я объясняю в общих чертах поведение машины. Следующий раздел посвящается построению полной таблицы для $\mathcal{U}$.

Здесь снова – тот же рукописный шрифт. Тьюринг использует $\mathcal{M}$ для произвольной машины и $\mathcal{U}$ для Универсальной машины.

Говоря о компьютерных программах, обычно упоминают *вход* и *выход*. Программа считывает вход и записывает выход. У описанных до сих пор машин, в сущности, нет входа, так как они начинают с пустой ленты. Они генерируют выход в виде последовательности нулей и единиц, возможно чередующихся с какими-то другими символами, используемыми в качестве временных меток или временной памяти. Универсальная машина $\mathcal{U}$, наоборот, требует определенный вход, а именно ленту со стандартным описанием (далее описанием, когда речь идет об описании машины – *прим. пер.*) $\mathcal{M}$ – последовательно-

стью букв *A*, *C*, *D*, *L*, *N* и *R*, описывающей все конфигурации $\mathcal{M}$. Машина $\mathcal{U}$ считывает и интерпретирует это описание и печатает тот же самый результат, который должна напечатать $\mathcal{M}$.

Но это не совсем так: результат $\mathcal{U}$ *не* будет тем же, что результат $\mathcal{M}$. В общем случае невозможно, чтобы $\mathcal{U}$ полностью воспроизводила $\mathcal{M}$. Машина $\mathcal{M}$ может начать с пустой ленты, тогда как машина $\mathcal{U}$ получает не пустую ленту, а ленту с записанным на ней описанием $\mathcal{M}$. Что случится, если $\mathcal{M}$, вместо того чтобы точно следовать соглашениям Тьюринга, будет записывать результат в обоих направлениях? Любая попытка точно воспроизвести $\mathcal{M}$ могла бы легко закончиться записью поверх описания.

Тьюринг говорит, что $\mathcal{U}$ предоставляется лента, «в начале которой» – описание машины $\mathcal{M}$. У ленты, которая бесконечна в обоих направлениях, нет «начала». Тьюринг неявно ограничивает выход $\mathcal{U}$ той частью ленты, которая расположена после описания.

Если мы ограничимся рассмотрением машин, которые печатают только в одном направлении (так или иначе таково соглашение Тьюринга), можем ли мы написать Универсальную машину, которая считывает расположенное в начале ленты описание машины, а затем точно повторяет ее выход в бесконечно пустой области ленты после описания?

Похоже, и это тоже невозможно. Понятно, что Универсальной машине нужна своя собственная рабочая область, поэтому ее выход будет отличаться от выхода машины, которую она пытается повторить. Даже если потребовать, чтобы Универсальная машина повторяла только *F*-клетки машины $\mathcal{M}$, то такая Универсальная машина, наверное, была бы значительно сложнее той, что описывает Тьюринг.

Тьюринг не гарантирует, что его Универсальная машина в точности повторяет выход машины, которую она воспроизводит. Он говорит только, что «$\mathcal{U}$ вычислит ту же самую последовательность, что и $\mathcal{M}$». На самом деле *вдобавок* к этой последовательности $\mathcal{U}$ печатает на выходе массу вспомогательных вещей.

Тьюринг идет к построению Универсальной машины довольно необычным путем.

> Давайте сначала предположим, что у нас есть машина $\mathcal{M}'$, которая запишет в *F*-клетки последовательные полные конфигурации $\mathcal{M}$.

Как вы помните, полная конфигурация – это «снимок» ленты по завершении операции вместе с положением головки и следующей

m-конфигурацией. Последовательные полные конфигурации отражают всю историю операций машины.

Их можно представить так же, как на с. 235, используя второе описание (C), где все символы идут одной строкой.

Такое представление в виде сплошного потока информации показано на странице 112 настоящей книги:

ƀ : ə ə ɒ 0 0 : ə ə q 0 0 : ...

В этой системе обозначений последовательные полные конфигурации разделяются двоеточиями. В каждой полной конфигурации очередному текущему (в оригинале *scanned*, сканируемому – *прим. пер.*) символу предшествует буква немецкого алфавита, обозначающая следующую *m*-конфигурацию.

Или лучше б было преобразовать это описание (как в §5) заменой каждой *m*-конфигурации на «*D*» с последующими «*A*», повторенными нужное число раз, и заменой каждого символа на «*D*» с последующими «*C*», повторенными нужное число раз. Число букв «*A*» и «*C*» должно соответствовать числам, выбранным в §5, так чтобы, в частности, «0» был заменен на «*DC*», «1» – на «*DCC*», а пробелы – на «*D*».

Тьюринг придумал это стандартное описание (как он его назвал) для кодирования состояний машины. Теперь он предлагает использовать его для представления полных конфигураций.

Такие подстановки нужно делать после того, как полные конфигурации собраны вместе, как в (C). Если мы делаем подстановку сразу, возникают трудности.

Думаю, Тьюринг имеет здесь в виду, что *m*-конфигурации и символы будут теперь представляться несколькими символами (например, 1 становится *DCC*), поэтому необходимо заботиться о том, чтобы при переходе к следующей *m*-конфигурации не разрушался код символа.

В каждой полной конфигурации все пустые клетки должны быть заменены на «*D*» так, чтобы полная конфигурация не представляла собой конечную последовательность символов.

Буква *D* представляет пустую клетку. Тьюрингу не нужно, чтобы в полных конфигурациях появлялись какие-либо разрывы. Ему нужно, чтобы каждая полная конфигурация была сплошной последовательностью букв. Фраза Тьюринга «так, чтобы полная конфигурация не представляла собой конечную последовательность символов» не вполне понятна. Полагаю, вместо «не» (not) должно быть «теперь» (now). Определенно ему не нужна бесконечная последовательность букв *D* для представления пустой ленты. Каждая полная конфигурация конечна.

Если в описании машины II из §3 мы заменяем «ɒ» на «*DAA*», «ə» на «*DCCC*», «ɋ» на «*DAAA*», то последовательность (C) становится следующей:

$$DA : DCCCDCCCDAADCDDC : DCCCDCCCDAAADCDDC : \ldots \quad (C_1)$$

(Это – последовательность символов в *F*-клетках.)

Тьюринг не приводит *все* подстановки, которые он делает. Он также заменяет ɓ на *DA*, пустые клетки на *D*, а нули на *DC*.

Комментарий в скобках относится к выходу машины $\mathcal{M}'$, которую предлагает Тьюринг. Обычная машина $\mathcal{M}$ печатает нули и единицы в *F*-клетках и использует *E*-клетки для других символов, чтобы помочь себе в вычислении нулей и единиц. Машина $\mathcal{M}'$ печатает последовательные полные конфигурации $\mathcal{M}$ в *F*-клетках и использует *E*-клетки, чтобы помочь себе в построении этих последовательных полных конфигураций.

Представленные таким образом полные конфигурации трудно читать. Как я уже говорил, этому можно помочь, если обращать внимание на каждую букву *D*, представляющую либо конфигурацию, либо символ.

- Если *D* сопровождается одной или более буквами *A*, то это – конфигурация. Номер конфигурации – это количество букв *A*.
- Если *D* не сопровождается *A*, то это – символ. В подобном случае *D* может сопровождаться буквами *C*. *D* сама по себе – это пустая клетка, *DC* – это 0, *DCC* – это 1, а большее количество *C* обозначает другие символы.

Нетрудно видеть, что если можно построить $\mathcal{M}$, то можно построить и $\mathcal{M}'$. Поведение $\mathcal{M}'$ могло бы зависеть от правил поведения (т. е. S.D) $\mathcal{M}$, записанных где-то в ней самой (т. е. внутри $\mathcal{M}'$); каждый шаг мог бы выполняться путем обращения к этим правилам.

Идея о том, чтобы $\mathcal{M}'$ имела описание $\mathcal{M}$, «записанное где-то в ней самой», – совершенно новая. Где оно записано? Как к нему обратиться? Тьюринг развивает идею машины $\mathcal{M}'$ таким образом, что это уводит его от цели, хотя в том, что $\mathcal{M}'$ может быть построена, действительно есть смысл.

> Мы только должны считать, что правила можно вынуть из машины и заменить другими, и мы имеем нечто очень похожее на универсальную машину.

Вот теперь все становится немного понятнее. В начале этого раздела Тьюринг сказал, что $\mathcal{U}$ снабжена лентой, содержащей описание $\mathcal{M}$. Это и означает, что ленту «можно вынуть из нее (машины) и заменить другой». Машине $\mathcal{U}$ можно задать ленту с описанием независимо от того, какую машину она должна эмулировать.

В принципе, $\mathcal{U}$ теперь представляется ну если не *совершенно* простой, то гораздо менее сложной. $\mathcal{U}$ запускается с лентой, на которой напечатано описание $\mathcal{M}$. Она отвечает за печать последовательных полных конфигураций $\mathcal{M}$. Описание и полные конфигурации кодируются одинаково: каждая полная конфигурация содержит последовательность символов, главным образом символов, печатаемых на ленте. Каждая полная конфигурация включает также D с последующими одной или более A, означающими следующую m-конфигурацию, предшествующую текущему символу, например:

DAAADCC

Эта последовательность букв, появляющихся в полной конфигурации, означает, что очередной m-конфигурацией будет q_3, а следующим текущим символом – 1. Где-то в описании $\mathcal{M}$ есть точно такая же последовательность букв. (Если ее нет, то что-то не так, а $\mathcal{M}$ не является машиной без циклов.) Все, что $\mathcal{U}$ должна сделать для определения следующей конфигурации, – найти совпадение. Найдя такую же конфигурацию, $\mathcal{U}$ сразу получает доступ к ее операции – символу, который должен печататься, коду перемещения головки и следующей m-конфигурации. После этого $\mathcal{U}$ должна создать новую полную конфигурацию на основе последней полной конфигурации, включив в нее напечатанный символ и следующую m-конфигурацию.

Понять устройство Универсальной машины можно было бы проще, если считать, что начальная полная конфигурация машины три-

виальна, а каждый шаг от одной полной конфигурации к следующей приводит к мелким изменениям вроде сравнения и копирования символов. Но Тьюринг уже определил арсенал *m*-функций, которые выполняют подобные рутинные операции.

Тем не менее пока он говорит не об $\mathcal{U}$, а о $\mathcal{M}'$, которая лишь печатает полные конфигурации $\mathcal{M}$.

Недостает одного: сейчас машина $\mathcal{M}'$ не печатает никаких цифр.

Это так. За всем этим мы забыли, что $\mathcal{M}'$ (или $\mathcal{U}$) печатает в *F*-клетках лишь последовательные полные конфигурации $\mathcal{M}$ из букв *A*, *C*, *D* и двоеточий и, возможно, использует *E*-клетки в качестве рабочей памяти. Настоящая цель этой игры – печать нулей и единиц.

Мы можем исправить это, печатая между каждой последовательной парой полных конфигураций цифры, которые появляются в новой конфигурации, но не в старой. Тогда (C_1) становится такой:

$$DDA : 0 : 0 : DCCCDCCCDAADCDDC : DCCC \ldots . \qquad (C_2)$$

Не вполне очевидно, что *E*-клетки предоставляют достаточно места для необходимой «черновой работы», но это действительно так.

Лишняя *D* в начале строки (C_2) – это типографская ошибка. Единственное отличие (C_2) от начала (C_1) – два нуля и два двоеточия. Они – результат первой операции, поэтому они печатаются после первой полной конфигурации.

Тьюрингу нужно, чтобы $\mathcal{M}'$ (и $\mathcal{U}$) печатала те же самые цифры 0 и 1, что и $\mathcal{M}$, потому что тогда можно сказать, что $\mathcal{M}'$ вычисляет ту же последовательность, что и $\mathcal{M}$. Разница только в том, что эти цифры теперь прячут в выход между последовательными полными конфигурациями машины.

Именно поэтому Тьюринг требует, чтобы его машины печатали вычисляемые числа последовательно и не изменяли число, как только оно напечатано. Без этого требования числа, напечатанные $\mathcal{M}'$ (и $\mathcal{U}$), были бы в полном беспорядке.

Тьюринг говорит, что $\mathcal{M}'$ должна печатать все цифры (0 и 1), «которые появляются в новой конфигурации, но не в старой». Когда машина приводится к стандартной форме (печать только одного символа и одно движение головки за операцию), часто случается, что машина по

ходу дела видит символ 0 или 1 где-то в другом месте. В этих случаях машина должна перепечатать 0 или 1. Машина $\mathcal{M}'$ должна пренебречь случаями, когда $\mathcal{M}$ перепечатывает 0 или 1. $\mathcal{M}'$ (и следовательно, Универсальная машина) должна печатать 0 или 1 *только тогда, когда текущая клетка – пустая*.

Тьюринг завершает этот раздел предположением, что полные конфигурации могут быть представлены в числовой форме, но это то, что он никогда не использует:

> Последовательности символов между двоеточиями в таких выражениях, как (C_1), могут использоваться как стандартные описания полных конфигураций. Если буквы заменить цифрами, как в §5, мы [243]
> получим числовое описание полной конфигурации, которую можно назвать ее описательным номером (далее описателем – *прим. пер.*).

Теперь давайте забудем все о $\mathcal{M}'$ и начнем рассматривать $\mathcal{U}$.

Хорошо известно, что описание Универсальной машины Тьюринга содержит несколько ошибок. (Весьма удивительно, как мало ошибок оно содержит, учитывая, что у Тьюринга не было возможности моделировать ее на настоящем компьютере.) Мой анализ Универсальной машины обязан исправлениям Эмиля Поста[1] и разбору Дональда Дэвиса[2].

Поскольку Универсальная машина очень важна для доводов Тьюринга в оставшейся части его статьи, он доказывает существование такой машины, создавая ее фактически полностью со всеми подробностями. Но стоит только понять ее основной механизм, как можно обнаружить, что эти подробности довольно скучны. Никто вас не накажет, если вы не усвоите всех символов и функций в описании Тьюринга.

---

[1] В приложении к статье Эмиля Поста «Recursive Unsolvability of a Problem of Thue», *The Journal of Symbolic Logic*, Vol. 12, No. 1 (Mar. 1947), 1–11. Полная статья перепечатана в Martin Davis, ed., *The Undecidable* (Raven Press, 1965), 293–303. Приложение перепечатано в B. Jack Copeland, ed., *The Essential Turing* (Oxford University Press, 2004), 97–101.

[2] Donald W. Davies, «Corrections to Turing's Universal Computing Machine» in C. Jack Copeland, ed., *The Essential Turing*, 103–124. Любой заинтересованный в программировании модели Универсальной машины захочет изучить статью Дэвиса.

7. *Подробное описание универсальной машины.*

Ниже приведена таблица поведения этой универсальной машины. Все возможные *m*-конфигурации машины появляются в первом и последнем столбцах таблицы вместе со всеми теми, которые появляются при полном выписывании несокращенных таблиц тех, что появляются в таблице в виде *m*-функций. Например, $\mathfrak{e}(\mathfrak{anf})$ появляется в таблице и является *m*-функцией.

m-конфигурация $\mathfrak{anf}$ является частью Универсальной машины Тьюринга. Одна из конфигураций в конце машины имеет $\mathfrak{e}(\mathfrak{anf})$ в столбце «*заключ. m-конфиг.*». Скелетная таблица для $\mathfrak{e}$ появляется на странице 239 статьи Тьюринга (и на странице 147 этой книги):

$\mathfrak{e}(\mathfrak{C})$	ə	*R*	$\mathfrak{e}_1(\mathfrak{C})$
	Not ə	*L*	$\mathfrak{e}(\mathfrak{C})$
$\mathfrak{e}_1(\mathfrak{C})$	Any	*R, E, R*	$\mathfrak{e}_1(\mathfrak{C})$
	None		$\mathfrak{C}$

Тьюринг теперь показывает несокращенную таблицу, где вместо $\mathfrak{C}$ подставлена $\mathfrak{anf}$:

Ее несокращенная таблица (смотри с. 239)

$\mathfrak{e}(\mathfrak{anf})$	ə	*R*	$\mathfrak{e}_1(\mathfrak{anf})$
	Not ə	*L*	$\mathfrak{e}(\mathfrak{anf})$
$\mathfrak{e}_1(\mathfrak{anf})$	Any	*R, E, R*	$\mathfrak{e}_1(\mathfrak{anf})$
	None		$\mathfrak{anf}$

Следовательно, $\mathfrak{e}_1(\mathfrak{anf})$ является *m*-конфигурацией $\mathcal{U}$.

Тьюринг начинает с описания ленты, на которой записано описание некоторой машины. Это та лента, которую Универсальная машина будет читать и интерпретировать.

Когда $\mathcal{U}$ готова начать работу, на ленту, по которой она движется, занесены символ ə в *F*-клетке и еще один ə в следующей *E*-клетке; следом, только в *F*-клетках, идет S.D машины, за которым следует двойное двоеточие «::» (отдельный символ в *F*-клетке). S.D состоит из множества команд, разделенных точкой с запятой.

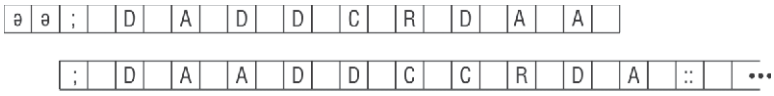
Между прочим, это первое в этой статье употребление Тьюрингом слова «команды» («instructions»). Это слово здесь к месту, так как конфигурации машин теперь играют другую роль: они должны стать командами Универсальной машины.

Раньше (в разделе 5 на странице 162 этой книги) у Тьюринга точкой с запятой заканчивалась каждая конфигурация, несмотря на это, Универсальная машина требует, чтобы с точки с запятой *начиналась* каждая команда. Это – только один из нескольких небольших «ляпов» в описании Универсальной машины.

Для иллюстрации работы $\mathcal{U}$ давайте предоставим ей простую машину $\mathcal{M}$. Это – упрощенная форма машины, которая поочередно печатает цифры 0 и 1:

m-конфиг.	*символ*	*операции*	*заключит. m-конфиг.*
q_1	S_0	PS_1, R	q_2
q_2	S_0	PS_0, R	q_1

Эта упрощенная машина имеет только две конфигурации вместо четырех и не пропускает ни одной клетки. Вот лента, подготовленная согласно указаниям Тьюринга, но с точками с запятой, предшествующими каждой команде. Лента так длинна, что я разбил ее на две строки:



Двойное двоеточие отделяет команды $\mathcal{M}$ от последующих ее полных конфигураций, которые будет печатать $\mathcal{U}$. Тьюринг напоминает нам, как эти команды кодируются:

> Каждая команда состоит из пяти следующих друг за другом частей
>
> (i) «*D*» с последующей последовательностью букв «*A*». Описывает соответствующую *m*-конфигурацию.

Чтобы обозначить *m*-конфигурацию, вслед за *D* должна идти хотя бы одна буква *A*; это значит, что конфигурации начинаются с q_1, а не с q_0.

(ii) «*D*» с последующей последовательностью букв «*C*». Описывает текущий символ.

Для символов: *D* сама по себе означает пустую клетку; *D* с одной *C* означает 0, а с двумя *C* означает 1.

(iii) «*D*» с последующей другой последовательностью букв «*C*». Описывает символ, на который должен быть заменен текущий символ.

(iv) «*L*», «*R*» или «*N*», описывающие, должна ли машина двигаться влево, вправо или никуда.

(v) «*D*» с последующей последовательностью букв «*A*». Описывает заключительную *m*-конфигурацию.

Универсальная машина должна печатать полные конфигурации, состоящие из букв *A*, *C* и *D*, а также вычисляемую последовательность, состоящую из цифр 0 и 1. Универсальная машина использует строчные буквы в качестве меток в *E*-клетках. В итоге:

Машина $\mathcal{U}$ должна быть способна печатать «*A*», «*C*», «*D*», «0», «1», «*u*», «*v*», «*w*», «*x*», «*y*», «*z*».

Тьюринг забыл включить в этот список двоеточие, которое разделяет последовательные полные конфигурации.

S.D формируется из «;», «*A*», «*C*», «*D*», «*L*», «*R*», «*N*».

Затем Тьюринг представляет одну последнюю функцию, которая нужна Универсальной машине.

Вспомогательная скелетная таблица. [244]

$$\mathrm{con}(\mathfrak{C}, \alpha) \quad \begin{cases} \text{Not } A & R, R & \mathrm{con}(\mathfrak{C}, \alpha) \\ A & L, P\alpha & \mathrm{con}_1(\mathfrak{C}, \alpha) \end{cases}$$

$$\mathrm{con}_1(\mathfrak{C}, \alpha) \quad \begin{cases} A & R, P\alpha & \mathrm{con}_1(\mathfrak{C}, \alpha) \\ D & R, P\alpha, R & \mathrm{con}_2(\mathfrak{C}, \alpha) \end{cases}$$

$\mathrm{con}(\mathfrak{C}, \alpha)$. Начиная с *F*-клетки, скажем S, последовательность C символов, описывающих конфигурацию, ближайшую справа к S, помечается буквами $\alpha. \rightarrow \mathfrak{C}$.

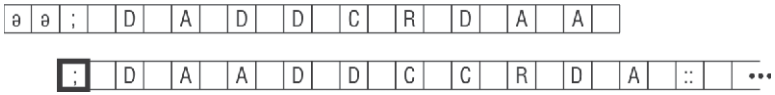
$\mathfrak{con}_2(\mathfrak{C}, \alpha)$	C	$R, P\alpha, R$	$\mathfrak{con}_2(\mathfrak{C}, \alpha)$	$\mathfrak{con}(\mathfrak{C},)$. В заключительной конфигурации машина просматривает клетку четырьмя клетками правее клетки с C. C остается непомеченной.
	Not C	R, R	$\mathfrak{C}$	

M-функция $\mathfrak{con}$ означает «конфигурация», и в ней пропущена строка[1]:

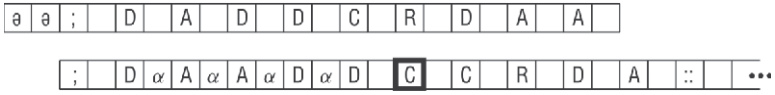
$\mathfrak{con}_1(\mathfrak{C}, \alpha)$	*None*	$PD, R, P\alpha, R, R, R$	$\mathfrak{C}$

Мы вскоре увидим, как эта пропущенная строка сыграет свою роль.

Задача функции $\mathfrak{con}$ – пометить конфигурацию с данным символом, заданным вторым аргументом. Предположим, что головка находится на точке с запятой, предшествующей команде:



Функция $\mathfrak{con}$ перемещает головку сразу на две клетки вправо, пока не встретит *A*. Слева от *A* она печатает α. Функция $\mathfrak{con}_1$ продолжает печатать метки справа от каждой *A*, пока не встретит *D*. Она печатает метку справа и от этой *D*, а затем переходит к $\mathfrak{con}_2$. Функция $\mathfrak{con}_2$ печатает метки справа от каждого *C* (если они есть). В этом примере в конфигурации нет *C*, так как текущая клетка – пустая, поэтому результат таков:



Пояснительные абзацы в скелетной таблице для $\mathfrak{con}$ слегка сбивают с толку, так как Тьюринг использует букву *C* и для обозначения всей последовательности символов, определяющих конфигурацию,

[1] Подсказано Постом, «Recursive Unsolvability of a Problem of Thue», 7.

и как элемент стандартного описания. Первое предложение второго абзаца «В заключительной конфигурации...» означает, что головка остается четырьмя клетками правее последней клетки конфигурации (то есть клетки текущего символа). Утверждение «*C* остается непомеченной» в значении «конфигурация остается непомеченной» имеет место, только когда второй аргумент $\mathfrak{con}$ пуст.

Описание Универсальной машины занимает в статье Тьюринга всего две страницы. Ранее Тьюринг так умело определил свои *m*-функции, что во многих случаях *m*-конфигурации $\mathcal{U}$ просто отсылают к определенным функциям. Как обычно, машина начинается с *m*-конфигурации $\mathfrak{b}$:

Таблица для $\mathcal{U}$.

$\mathfrak{b}$		$\mathfrak{f}(\mathfrak{b}_1, \mathfrak{b}_1, ::)$	$\mathfrak{b}$. Машина печатает *:DA*
$\mathfrak{b}_1$	*R, R, P:, R, R, PD, R, R, PA*	$\mathfrak{anf}$	в *F*-клетке после :: → $\mathfrak{anf}$

M-функция $\mathfrak{f}$ находит двойное двоеточие, которое отделяет команды от полных конфигураций. Как вы помните, каждая полная конфигурация показывает на ленте все символы вместе с *m*-конфигурацией, предшествующей текущей клетке. Когда машина начинает работу, ее первая *m*-конфигурация q_1 имеет описание *DA*. Это то, что напечатает $\mathfrak{b}_1$, начав с двоеточия, которое будет разделять каждую полную конфигурацию:

$\mathfrak{d}$	$\mathfrak{d}$	;		D		A		D		D		C		R		D		A		A	

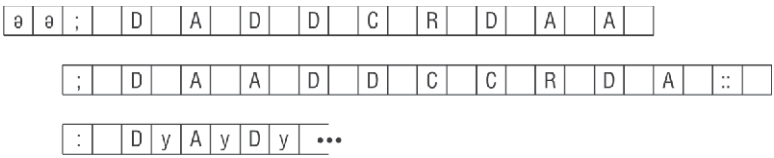
;		D		A		A		D		D		C		C		R		D		A		::	

:		D		A		•••

Следующая *m*-конфигурация $\mathcal{U}$ – это $\mathfrak{anf}$, которая по Дональду Дэвису означает *anfang*, с немецкого – *начало*. Функция $\mathfrak{g}$ в первой строке была ошибочно обозначена в таблицах функций как $\mathfrak{q}$. Она ищет последнее вхождение своего второго аргумента:

$\mathfrak{anf}$	$\mathfrak{g}(\mathfrak{anf}_1, :)$	$\mathfrak{anf}$. Машина помечает меткой y конфигурацию в последней полной конфигурации. → $\mathfrak{fom}$
$\mathfrak{anf}_1$	$\mathfrak{con}(\mathfrak{fom}, y)$	

После того как $\mathfrak{g}$ находит двоеточие (которое предшествует текущей полной конфигурации), $\mathfrak{con}$ помечает *m*-конфигурацию буквой *y*. Дополнительная строка, которую я добавил к $\mathfrak{con}_1$, также вступает в игру: она печатает *D* (пустую клетку), а также помечает ее:

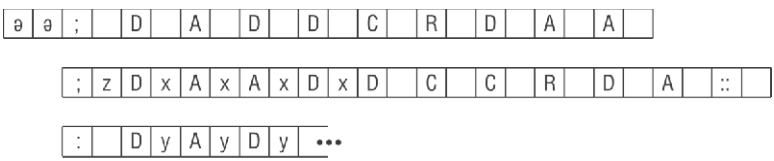


Всякий раз, как $\mathfrak{con}$ помечает в полной конфигурации *m*-конфигурацию и переходит к пустой ячейке, чтобы найти букву *D*, представляющую текущий символ, $\mathfrak{con}_1$ печатает *D*. Поэтому лента постепенно удлиняется по мере необходимости.

Теперь машина должна найти команду, чья конфигурация совпадает с символами полной конфигурации, помеченными *y*. Конечно, таких команд много, но их легко найти, так как каждой предшествует точка с запятой. Эти команды просматриваются с последней по направлению к началу. *m*-конфигурация $\mathfrak{fom}$ напоминает $\mathfrak{fom}$, но на самом деле это – *kom*, наверное, одно из нескольких сокращений, обозначающих *compare* (*сравнить*).

$\mathfrak{fom}$	;	*R, Pz, L*	con($\mathfrak{fmp}$, *x*)	$\mathfrak{fom}$. Машина находит последнюю точку с запятой, не помеченную *z*. Она помечает эту точку с запятой *z*, а последующую конфигурацию – *x*.
	z	*L, L*	$\mathfrak{fom}$	
	not *z* nor ;	*L*	$\mathfrak{fom}$	

Сначала $\mathfrak{fom}$ находит последнюю (самую правую) команду, печатает *z* после точки с запятой, а затем с помощью $\mathfrak{con}$ помечает последующую конфигурацию.



Метка *z* означает, что эта команда была проверена. При последующих попытках обнаружить совпадение $\mathfrak{fom}$ пропускает все точки с запятой, уже помеченные *z*.

m-конфигурация $\mathfrak{fmp}$ (еще одно сокращение для *compare*?) использует $\mathfrak{cpe}$ для сравнения конфигурации, помеченной *x* (это *m*-конфигурация и текущий символ команды), с конфигурацией, помеченной *y* (это текущая *m*-конфигурация и текущий символ, указанный в полной конфигурации):

$\mathfrak{fmp}$	$\mathfrak{cpe}(\mathfrak{e}(\mathfrak{fom}, x, y), \mathfrak{sim}, x, y)$	$\mathfrak{fmp}$. Машина сравнивает последовательности, помеченные *x* и *y*. Она стирает все символы *x* и *y*. → $\mathfrak{sim}$, если они не совпали. Иначе → $\mathfrak{fom}$.

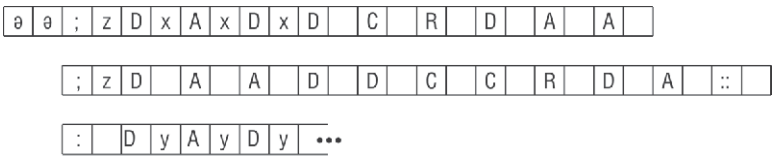
Сравнивая буквы, функция $\mathfrak{cpe}$ стирает метки, которыми они помечены. При совпадении все метки *x* и *y* стираются, и мы переходим к $\mathfrak{sim}$ (от *similar* – *подобное*).

Если конфигурации, помеченные *x* и *y*, не совпадают (что невозможно в нашем примере), то вступает в силу первый аргумент $\mathfrak{cpe}$ – функция $\mathfrak{e}$ (стереть), которая стирает все оставшиеся метки *x* и *y* и в итоге переходит к $\mathfrak{fom}$, чтобы проверить следующую команду.

Небольшая проблема функции $\mathfrak{fmp}$ – в том, что Тьюринг никогда не определял версию $\mathfrak{e}$, у которой один аргумент – *m*-конфигурация, а два других – символы. Кроме того, она не может вернуться к $\mathfrak{fom}$, потому что некоторые или все метки *y* стерты функцией $\mathfrak{cpe}$. На самом деле она должна вернуться к $\mathfrak{anf}$, чтобы заново пометить конфигурацию. Дональд Дэвис полагает, что эта команда в действительности должна быть такой:

$$\mathfrak{fmp} \qquad \mathfrak{cp}(\mathfrak{e}(\mathfrak{e}(\mathfrak{anf}, x), y), \mathfrak{sim}, x, y)$$

В нашем примере $\mathfrak{anf}_1$ вновь пометит *m*-конфигурацию и текущий символ в полной конфигурации, а $\mathfrak{fom}$ пометит следующую команду (обратным просмотром команд):



На сей раз функция $\mathfrak{cpe}$, вызванная $\mathfrak{fmp}$, обнаружит совпадение и перейдет к $\mathfrak{sim}$. Все метки *x* и *y* исчезнут, но метки *z* останутся. Самая левая метка *z* предшествует команде, которую должна выполнить $\mathcal{U}$. А пока Тьюринг подводит итог:

> $\mathfrak{anf}$. Получив длинное представление, ищется последняя команда, относящаяся к последней конфигурации. Она может быть опознана впоследствии как команда после последней точки с запятой, отмеченная *z*. → $\mathfrak{sim}$.

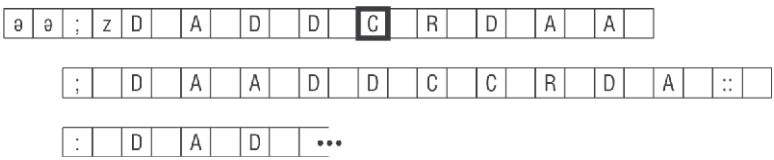
Фактически это *первая* (крайняя левая) точка с запятой, помеченная *z*, но последняя проверенная команда. *M*-конфигурация $\mathfrak{sim}$ начинает с применения $\mathfrak{f}'$ для поиска этого маркера и позиции непосредственно с точки с запятой, предшествующей команде. Как вы помните, команда состоит из пяти элементов: *m*-конфигурация, текущий символ, символ для печати, *L*, *N* или *R* и заключительная *m*-конфигурация.

[245]

$\mathfrak{sim}$			$\mathfrak{f}'(\mathfrak{sim}_1, \mathfrak{sim}_1, z)$
$\mathfrak{sim}_1$			$\mathfrak{con}(\mathfrak{sim}_2,)$
	A		$\mathfrak{sim}_3$
$\mathfrak{sim}_2$	not *A*	*R*, *Pu*, *R*, *R*, *R*	$\mathfrak{sim}_2$
	not *A*	*L*, *Py*	$\mathfrak{e}(\mathfrak{mf}, z)$
$\mathfrak{sim}_3$	*A*	*L*, *Py*, *R*, *R*, *R*	$\mathfrak{sim}_3$

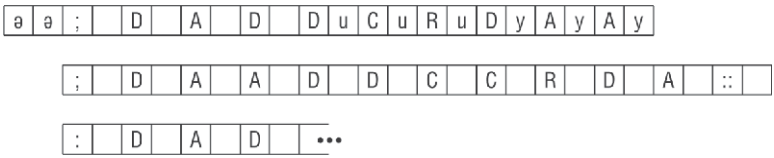
$\mathfrak{sim}$. Машина размечает команды. Часть команды, которая относится к выполняемым действиям, будет помечена *u*, а заключительная *m*-конфигурация – символом *y*. Символы *z* стираются.

M-конфигурация $\mathfrak{sim}_1$ обращается к функции $\mathfrak{con}$ с пустым вторым аргументом. В сущности, она пропускает *m*-конфигурацию и текущий символ, встав на второй символ операции печати.



Вторая строка *m*-конфигурации $\mathfrak{sim}_2$ неверна: Эмиль Пост полагает, что она должна сдвинуть головку *влево* до печати *u*. Две *m*-кон-

фигурации $\mathfrak{sim}_2$ и $\mathfrak{sim}_3$ помечают операцию (символ, который будет напечатан, и код смещения головки) и следующую *m*-конфигурацию. Функция $\mathfrak{e}$ стирает метку *z*, прежде чем перейти к $\mathfrak{mf}$.



Теперь *m*-конфигурация $\mathfrak{mf}$ (которая похожа на *mf*, но на самом деле – это *mk*, наверное, означает *mark* – *пометить*) помечает последнюю полную конфигурацию. Первым аргументом функции $\mathfrak{g}$ (которая в таблицах функций записана по ошибке как $\mathfrak{q}$) должна быть $\mathfrak{mf}_1$, а не $\mathfrak{mf}$.

$\mathfrak{mf}$			$\mathfrak{g}(\mathfrak{mf}, :)$
$\mathfrak{mf}_1$	not *A*	*R*, *R*	$\mathfrak{mf}_1$
	A	*L*, *L*, *L*, *L*	$\mathfrak{mf}_2$
$\mathfrak{mf}_2$	*C*	*R*, *Px*, *L*, *L*, *L*	$\mathfrak{mf}_2$
	:		$\mathfrak{mf}_4$
	D	*R*, *Px*, *L*, *L*, *L*	$\mathfrak{mf}_3$
$\mathfrak{mf}_3$	not :	*R*, *Pu*, *L*, *L*, *L*	$\mathfrak{mf}_3$
	:		$\mathfrak{mf}_4$
$\mathfrak{mf}_4$			$\mathfrak{con}(\mathfrak{l}(\mathfrak{l}(\mathfrak{mf}_5)),)$
$\mathfrak{mf}_5$	Any	*R*, *Pw*, *R*	$\mathfrak{mf}_5$
	None	*P* :	$\mathfrak{sh}$

$\mathfrak{mf}$. Последняя полная конфигурация помечается в четырех местах. Конфигурация остается непомеченной. Предшествующий ей символ помечается символом *x*. Остаток полной конфигурации делится на две части, первая из которых помечается *v*, вторая – *w*. После всего печатается двоеточие. → *oh*.

M-конфигурация $\mathfrak{mf}$ использует $\mathfrak{g}$ для поиска самого правого двоеточия. Это двоеточие предшествует последней полной конфигурации. Полная конфигурация находится в *F*-клетках и состоит главным образом из букв *D*, за которыми могут следовать *C*, представляющие символы на ленте. Где-то среди этих символов находится *m*-конфигурация – буква *D*, за которой следует одна или более *A*.

M-конфигурация $\mathfrak{mf}_1$ ищет внутри полной конфигурации *m*-конфигурацию. Найдя *A*, она движется влево до клетки с последним символом, предшествующим *m*-конфигурации. Эта клетка помечается *x*. Затем $\mathfrak{mf}_3$ помечает все предшествующие символы меткой *v*.

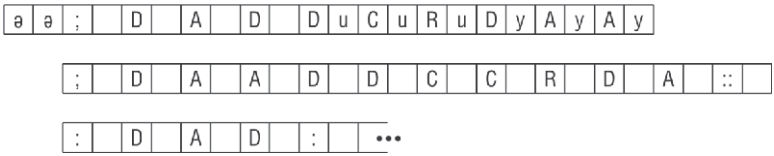
Когда $\mathfrak{mf}_3$ добирается до двоеточия, начинает работу $\mathfrak{mf}_4$. Она использует $\mathfrak{con}$, чтобы пропустить *m*-конфигурацию и текущий символ. Она останавливается, когда находит что-то, отличное от *C*. Все символы, кроме текущего, помечаются *w*. В заключение $\mathfrak{mf}_5$ печатает двоеточие.

Вот полная конфигурация, которая чуть сложнее простого примера, который мы рассматривали:

Эта полная конфигурация представляет ленту, начинающуюся с 0 (*DC*) и пустой клетки (*D*). Далее идет текущая клетка, обозначенная конфигурацией $\mathfrak{q}_1$ (*DA*). Текущая клетка – 0 (*DC*), за которой следует пустая клетка (*D*) и 0 (*DC*). После ее обработки $\mathfrak{mf}$ она станет такой:

Не помечено только одно – конфигурация (которая состоит из *m*-конфигурации *DA* и текущего символа *DC*).

В нашем упрощенном примере нет символов ни слева от *m*-конфигурации, ни справа от текущей клетки, поэтому метки *v*, *x* и *w* не имеют значения:



Теперь все помечено. Операция и заключительная *m*-конфигурация команды помечены *u* и *y*, а части полной конфигурации помечены *v*, *x* и *w*.

Если команда печатает 0 или 1, то Универсальная машина тоже должна напечатать 0 или 1, за исключением случаев, когда они перепечатываются сразу после просмотра. Универсальная машина должна напечатать 0 или 1, только если текущая клетка пуста. Эту задачу решает $\mathfrak{sh}$ (что может означать ***sh****ow* – *показать*).

$\mathfrak{sh}$			$\mathfrak{f}(\mathfrak{sh}_1, \mathfrak{inst}, u)$	$\mathfrak{sh}$. Проверяются (помеченные u) команды. Если оказывается, что они включают «Print 0» или «Print 1», то в конце печатается 0: или 1:.
$\mathfrak{sh}_1$		L, L, L	$\mathfrak{sh}_2$	
$\mathfrak{sh}_2$	D	R, R, R, R	$\mathfrak{sh}_2$	
	not D		$\mathfrak{inst}$	
$\mathfrak{sh}_3$	C	R, R	$\mathfrak{sh}_4$	
	not C		$\mathfrak{inst}$	
$\mathfrak{sh}_4$	C	R, R	$\mathfrak{sh}_5$	
	not C		$\mathfrak{pe}_2(\mathfrak{inst}, 0, :)$	
$\mathfrak{sh}_5$	C	R, R	$\mathfrak{sh}_5$	
	not C		$\mathfrak{pe}_2(\mathfrak{inst}, 1, :)$	

Сначала $\mathfrak{sh}$ находит самую левую метку u, а $\mathfrak{sh}_1$ сдвигает головку влево на три позиции, чтобы стать на последний символ, представляющий текущую ячейку. Если текущая клетка пустая, то этим символом будет D. Если это не D, то остаток данной m-конфигурации пропускается переходом к $\mathfrak{inst}$.

Если текущий символ пустой, то $\mathfrak{sh}_2$ переходит к $\mathfrak{sh}_3$ (а не к $\mathfrak{sh}_2$, как в таблице), после чего $\mathfrak{sh}_3$, $\mathfrak{sh}_4$ и $\mathfrak{sh}_5$ проверяют, печатает ли команда DC (0) или DCC (1). Если так, то $\mathfrak{pe}_2$ печатает в конце ленты эту цифру и двоеточие. Теперь лента выглядит так:

ə	ə	;		D		A		D		D	u	C	u	R	u	D	y	A	y	A	y

;		D		A		A		D		D		C		C		R		D		A		::	

:		D		A		D		:		0		:		...

Раздел таблицы $\mathfrak{sh}$ явно упрощается за счет использования двоичных чисел вместо десятичных. Десятичные числа потребовали бы еще восьми m-конфигураций (от $\mathfrak{sh}_6$ до $\mathfrak{sh}_{13}$) для печати цифр от 2 до 9.

Напечатав 0, 1 или ничего, Универсальная машина переходит к $\mathfrak{inst}$ (что может означать ***inst**ruction – команда*, а может быть, ***in-st**igate* (*побуждать*), что более вероятно). Все, что осталось сделать, – воспроизвести очередную полную конфигурацию $\mathcal{M}$. Новая полная конфигурация включает все символы текущей конфигурации, помеченные x, v и w, поскольку эти символы не должны меняться. Однако m-конфигурация и текущая клетка изменятся. Они будут заменены m-конфигурацией, помеченной y, и символом, помеченным u.

В таблице $\mathfrak{inst}$ есть еще одна ссылка на функцию $\mathfrak{g}$, которая сначала была определена как $\mathfrak{q}$. Кроме того, в пятой строке вместо $\mathfrak{ec}_5$ должно быть $\mathfrak{ce}_5$, как в третьей и четвертой строках.

[246]

$\mathfrak{inst}$		$\mathfrak{g}(\mathfrak{l}(\mathfrak{inst}_1), u)$	$\mathfrak{inst}$. Выписывается очередная
$\mathfrak{inst}_1$	α	R, E $\mathfrak{inst}_1(\alpha)$	полная конфигурация, выполняются помеченные команды. Символы u, v, w, x, y стираются. $\rightarrow \mathfrak{anf}$.
$\mathfrak{inst}_1(L)$		$\mathfrak{ec}_5(\mathfrak{ov}, v, y, x, u, w)$	
$\mathfrak{inst}_1(R)$		$\mathfrak{ec}_5(\mathfrak{ov}, v, y, x, u, w)$	
$\mathfrak{inst}_1(N)$		$\mathfrak{ec}_5(\mathfrak{ov}, v, y, x, u, w)$	
$\mathfrak{ov}$		$\mathfrak{e}(\mathfrak{anf})$	

На самом деле ни $\mathfrak{ce}_5$, ни $\mathfrak{ce}_4$ не определялись. Но их можно легко определить на основе $\mathfrak{ce}_3$:

$\mathfrak{ce}_4(\mathfrak{B}, \alpha, \beta, \gamma, \delta)$ $\quad$ $\mathfrak{ce}(\mathfrak{ce}_3(\mathfrak{B}, \beta, \gamma, \delta), \alpha))$
$\mathfrak{ce}_5(\mathfrak{B}, \alpha, \beta, \gamma, \delta, \varepsilon)$ $\quad$ $\mathfrak{ce}(\mathfrak{ce}_4(\mathfrak{B}, \beta, \gamma, \delta, \varepsilon), \alpha))$

Функция $\mathfrak{ce}_5$ последовательно копирует в конец ленты символы, помеченные α, затем символы, помеченные β, и т. д., по ходу дела стирая метки.

m-конфигурация $\mathfrak{inst}$ обращается к $\mathfrak{g}$, которая переходит на самый правый символ, помеченный u; этот символ – L, R или N. m-конфигурация $\mathfrak{inst}_1$ просматривает этот символ, стирает его, а затем в зависимости от символа переходит к $\mathfrak{inst}_1(L)$, $\mathfrak{inst}_1(R)$ или $\mathfrak{inst}_1(N)$. Что хочет здесь сделать Тьюринг, понятно, но я, в сущности, против введения нового синтаксиса в этом месте машины, особенно когда в этом нет необходимости. Давайте заменим всю конфигурацию $\mathfrak{inst}_1$ следующей:

$$\mathfrak{inst}_1 \begin{cases} L & R, E & \mathfrak{ce}_5(\mathfrak{ov}, v, y, x, u, w) \\ R & R, E & \mathfrak{ce}_5(\mathfrak{ov}, v, x, u, y, w) \\ N & R, E & \mathfrak{ce}_5(\mathfrak{ov}, v, x, y, u, w) \end{cases}$$

Во всех трех случаях в конец ленты сначала копируются клетки, помеченные v, а потом клетки, помеченные w. Все символы, помеченные v, находятся в полной конфигурации левее клетки слева от текущей. Эта клетка помечена x. Все символы, помеченные w, – правее текущей клетки.

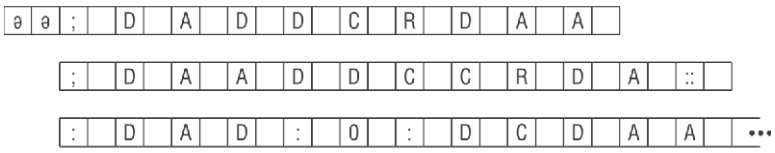
Три копии $\mathfrak{ce}_5$ в середине таблицы зависят от сдвига головки – влево, вправо или на месте. Порядок такой:

- Влево: Следующая m-конфигурация / Символ слева от головки / Печатаемый символ.

- Вправо: Символ слева от головки / Печатаемый символ / Следующая *m*-конфигурация.
- На месте: Символ слева от головки / Следующая *m*-конфигурация / Печатаемый символ.

Например, если головка сдвигается влево, то следующая *m*-конфигурация вставляется перед клеткой слева от предыдущего положения головки. Если головка двигается вправо, следующая *m*-конфигурация – справа от печатаемого символа.

Каждая из функций $\mathfrak{ce}_5$ переходит в $\mathfrak{ov}$ (что, наверное, означает ***ov****er* – *за*). Функция $\mathfrak{e}$ стирает все *E*-клетки и переходит в $\mathfrak{anf}$ для дальнейшего перемещения. Теперь наша лента выглядит так:



Вторая полная конфигурация содержит символы *DC* (обозначающие 0) и следом – *DAA*, обозначающие новую *m*-конфигурацию $\mathfrak{q}_2$.

Универсальная машина, как ее определил Тьюринг, имеет несколько ограничений. Она не может воспроизвести любую машину Тьюринга вообще. Она не будет работать правильно с машиной, головка которой уходит куда-то влево от своего исходного положения, потому что у нее нет возможности добавлять пустые клетки левее полных конфигураций. (Более того, в функции $\mathfrak{con}$ Тьюринг опустил добавление пустых клеток и *справа* тоже.). Кроме того, Универсальная машина работает правильно лишь с машинами, которые заносят цифры 0 и 1 только в пустые клетки, причем исключительно слева направо. Универсальная машина может обработать машины, которые действуют не так, но не напечатает правильной последовательности нулей и единиц.

Несмотря на эти ограничения, а также небольшие опечатки и «ляпы», Тьюринг сделал нечто совершенно необыкновенное. Он обобщил вычисления, показав, что одна универсальная машина, запрограммированная должным образом, может выполнить работу любой вычислительной машины. В одной широко известной книге по вычислимости говорится: «теорема Тьюринга о существовании универ-

сальной машины Тьюринга – одна из интеллектуальных вех прошлого столетия»[1].

Все это вызывает вопрос:

Компьютер изобретен Аланом Тьюрингом?

[1] John P. Burgess, предисловие к книге George S. Boolos, John P. Burgess, and Richard C. Jeffrey, *Computability and Logic*, fourth edition (Cambridge University Press, 2002), xi.

Глава 10

Вычислительные машины и вычислимость

Представив вычислительную машину, которая не делает почти ничего, Тьюринг фактически придумал универсальный компьютер «общего назначения». Это было революционной идеей. Тогда считалось общепринятым, что компьютеры следует проектировать специально для решения конкретных задач. Первый аналоговый компьютер, известный как Дифференциальный анализатор (разработанный и созданный в 1920-е годы профессором Массачусетского технологического института Ванневаром Бушем и его студентами), – пример такого подхода. Дифференциальный анализатор делал нечто очень важное – решал обыкновенные дифференциальные уравнения, – но это было все, что он делал.

Даже люди, серьезно занимавшиеся разработкой цифровых машин, часто не осознавали универсальности цифровой логики. Говард Эйкен, например, был одним из настоящих первопроходцев вычислительной техники и занимался ею с 1937 года. Тем не менее в 1956 году он заявил:

> Если бы оказалось, что внутренняя логика машины, разработанной для численного решения дифференциальных уравнений, совпадает с логикой машины, предназначенной для обработки складских накладных, я бы расценил это как самое удивительное совпадение, с которым я когда-либо сталкивался[1].

---

[1] Paul Ceruzzi, *Reckoners: The Prehistory of the Digital Computer, from Relays to the Stored Program Concept, 1935–1945* (Greenwood Press, 1983), 43. Первоисточник см. Howard Aiken, «The Future of AutomaticComputing Machinery», *Elektronische Rechenanlage und Informationsverarbeitung* (Darmstadt, 1956), 33.

Тьюринг, представивший компьютер как логическую машину, смотрел глубже. В то время как первые создатели компьютеров мыслили аппаратными категориями, Тьюринг с 1936 года писал программы. По Тьюрингу, даже такие элементарные арифметические операции, как сложение, можно было выполнить программно. Сравните заявление Эйкена 1956 года с тем, что писал Тьюринг в 1950 году:

> Это особое качество цифровых компьютеров имитировать любую машину с дискретными состояниями, позволяет сказать, что они являются *универсальными* машинами. Важным следствием существования такого рода машин, без учета быстродействия, является то, что не нужно разрабатывать разнообразные новые машины для выполнения разнообразных вычислительных процессов. Все они могут быть выполнены одним цифровым компьютером, программируемым для каждого случая должным образом. Можно видеть, что вследствие этого все цифровые компьютеры в некотором смысле эквивалентны[1].

Тьюринг добавляет важное условие «без учета быстродействия». Можно возразить, что при выборе компьютера быстродействие – это еще не все; это *только* один фактор. Всякий раз, когда людям нужны специализированные компьютеры, – скажем, для создания на компьютере изображения (computer-generated imagery, CGI) для голливудского блокбастера стоимостью в несколько миллионов долларов – обычно в первую очередь принимается во внимание быстродействие, хотя и большая память тоже не повредит. Но по своим возможностям перемалывать[2] числа все компьютеры действительно универсальны.

Место Алана Тьюринга в общей истории вычислительной техники никогда не было вполне ясным. В одной устоявшейся истории[3] он заслуживает лишь упоминания, но когда историю пишет видный математик, который рассматривает компьютер как физическое воплощение математических идей[4], Тьюринг становится основным

---

[1] Alan Turing, «Computing Machinery and Intelligence», *Mind*, Vol. LIX, No. 236 (October 1950), 441–2.

[2] Нередко употребляемый автором оборот «перемалывать числа», скорее всего, пришел из проекта Аналитической машины Ч. Бэббиджа, в которой был «склад» (прообраз современной оперативной памяти) и «мельница» (прообраз современного процессора). Числа на перфокартах передавались со «склада» на «мельницу» для «перемалывания» (обработки). Этот процесс проходил под управлением программы, тоже набитой на перфокартах. – *прим. перев.*

[3] Martin Campbell-Kelly and William Aspray, *Computer: A History of the Information Machine* (Basic Books, 1996).

[4] Martin Davis, *The Universal Computer: The Road from Leibniz to Turing* (Norton, 2000).

действующим лицом. На самом деле оценка Тьюринга в книгах по истории вычислительной техники зависит от подхода к компьютеру – технического и коммерческого или математического и научного.

Одна интригующая роль, которую сыграл Тьюринг, касается его взаимоотношений с Джоном фон Нейманом. Эти два человека встретились впервые в апреле 1935 года, когда Нейман приехал из Принстона в Кембридж, чтобы прочитать курс лекций о почти периодических функциях. Вскоре после этого Тьюринг решил, что ему самому нужно поехать в Принстонский университет[1]. Когда в конце 1936 года Тьюринг добрался до Принстона, у них снова произошла встреча[2]. Однажды фон Нейман заявил, что после Гёделевской теоремы о неполноте он перестал читать статьи по математической логике[3], поэтому не ясно, когда он в действительности читал «О вычислимых числах». У этих двух математиков были и другие общие математические интересы (почти периодические функции и теория групп), и они упоминались в письме фон Неймана от 1 июня 1937 года с рекомендацией Тьюринга на Проктеровскую стипендию на втором году его пребывания в Принстоне.

До того как Тьюринг покинул Принстон в июле 1938 года, фон Нейман предложил ему работу в Институте перспективных исследований (Institute for Advanced Study, IAS) в качестве своего ассистента за 1500 долл. в год, но Тьюринг отклонил это предложение. К тому времени фон Нейман почти наверняка прочитал статью Тьюринга. Эндрю Ходжес, писавший биографию Тьюринга, спрашивал физика Станислава Улама (который тоже был в IAS) об оценке Тьюринга фон Нейманом. (Сам фон Нейман умер в 1957 году в возрасте 53 лет.) Улам вспоминал путешествие с фон Нейманом летом 1938 года, когда фон Нейман в качестве игры предложил

> записать на листке бумаги как можно большее число, определив его методом, который действительно имеет некоторое отношение к какой-либо схеме Тьюринга… Фон Нейман очень восхищался им и упоминал его имя и «блестящие идеи», кажется, уже в начале 1939 года… Во всяком случае, фон Нейман не раз упоминал при мне имя Тьюринга в 1939 году в разговорах, касающихся автоматических способов создания формальных математических систем[4].

---

[1] Andrew Hodges, *Alan Turing: The Enigma* (Simon & Schuster, 1983), p. 95. Алан Тьюринг: Загадка (Simon & Schuster, 1983), p. 95.
[2] Hodges, *Alan Turing*, 118.
[3] Hodges, *Alan Turing*, 12.
[4] Hodges, *Alan Turing* , 145.

Эти первые короткие встречи Тьюринга и фон Неймана становятся неожиданно важными в сентябре 1944 года, когда фон Нейман приехал в Электротехническую школу Мура Университета Пенсильвании. Уже создавался компьютер, названный ENIAC (Electronic Numerical Integrator and Computer – электронный числовой интегратор и вычислитель), 30-тонное чудище, созданное под руководством Джона Преспера Эккерта (1919–1995) и Джона Уильяма Моучли (1907–1980). Недостатки ENIAC стали очевидны уже на этапе его создания, и планировался его преемник, который должен был называться EDVAC (Electronic Discrete Variable Automatic Computer – электронный автоматический вычислитель с дискретными переменными).

> С самого начала задачей фон Неймана было не просто быть потенциальным пользователем, но и вносить научный и инженерный вклад. В оставшиеся месяцы 1944 года и весь 1945 год, когда его не было в Лос-Аламосе, он принимал участие в технических конференциях по EDVAC, оказывал техническое содействие и выдвигал предложения по логическому проектированию[1].

С тех пор, как 30 июня 1945 года появился документ под названием «Предварительное сообщение о EDVAC»[2] с Джоном фон Нейманом в качестве единственного автора, разгорелся спор, и дым от него еще не рассеялся. В сообщении выделяются важные понятия – что компьютер должен быть электронным, что он должен работать с двоичными числами и что программы должны храниться в памяти, – но никогда так и не было окончательно установлено, исходили ли они от фон Неймана или он лишь четко сформулировал те идеи, которые витали в Школе Мура со времен ENIAC. Десятки лет при описании компьютеров ссылаются на «архитектуру фон Неймана», но употребление этого термина сомнительно, хотя бы из уважения к тем многим, кто внес вклад в архитектуру компьютеров.

В «Предварительном сообщении о EDVAC» приводится ссылка только на одну публикацию – статью под названием «Логическое исчисление идей, относящихся к нервной деятельности», опубликованной в *Bulletin of Mathematical Biophysics* (*Бюллетене математической биофизики*)[3]. Эта ссылка показывает заинтересованность фон Нейма-

---

[1] Nancy Stern, «John von Neumann's Influence on Electronic Digital Computing, 1944–1946», *Annals of the History of Computing*, Vol. 2 No. 4 (October 1980), 353.

[2] Перепечатано в Brian Randell, ed., *The Origins of Digital Computers* (Springer, 1973).

[3] W.S. MacCulloch and W. Pitts, «A Logical Calculus of the Ideas Immanent in Nervous Activity», *Bulletin of Mathematical Biophysics*, Vol. 5 (1943), 115–133.

на связью между компьютером и мозгом человека, но интересно еще и то, что авторы данной статьи в своих идеях о физиологии мозга опирались на функции машин Тьюринга. Норберт Винер (1894–1964) тоже цитирует статью Маккаллока и Питтса в своей классической книге «Кибернетика, или Управление и связь в животном и машине» (1948). Я еще расскажу о Маккаллоке, Питтсе, Винере и фон Неймане в главе 17.

Физик Стэнли Френкель, работавший с фон Нейманом в Лос-Аламосе, вспоминает восторг фон Неймана по поводу статьи Тьюринга в 1943 или 1944 году:

> Фон Нейман познакомил меня с этой статьей, и по его настоянию я внимательно изучил ее. Многие провозгласили фон Неймана «отцом вычислительной машины» (в современном смысле слова), но я уверен, что он никогда бы не совершил такой ошибки по отношению к себе самому. Наверное, его лучше было бы назвать повивальной бабкой, так как он неизменно твердил мне и, я уверен, другим, что фундаментальными концепциями мы обязаны Тьюрингу – в той мере, в какой они не были предвосхищены Бэббиджем, Лавлейс и др. На мой взгляд, основная роль фон Неймана – в том, что мир узнал об этих фундаментальных понятиях, введенных Тьюрингом, и об исследовательских работах, проведенных в школе Мура и других местах[1].

Во второй половине 1940-х годов фон Нейман, видимо, указывал на значимость Тьюринга нескольким людям. Например, в 1946 году он писал Норберту Винеру о «большом позитивном вкладе Тьюринга... том, что определенный механизм может быть “универсальным”»[2].

Хотя Алан Тьюринг известен главным образом своими рукописями, с его именем связаны также три больших компьютерных проекта.

Первым был Colossus – компьютер для расшифровки сообщений, разработанный и созданный в 1943 году в Блэтчли-парк. Он был спроектирован Томасом Х. Флауэрсом (1905–1998), который был знаком с коммутаторами и электроникой по своей работе в 1930-х годах в исследовательских лабораториях Отдела телефонии Министерства связи (Telephone Branch of the Post Office). Хотя некоторые авторы

---

[1] Письмо, приведенное в B. Jack Copeland, ed., *The Essential Turing: The Ideas that Gave Birth to the Computer Age* (Oxford University Press, 2004), 22. Это письмо – часть 6-страничного раздела в «Turing, von Neumann, and the Computer» in Copeland's guide to the «Computable Numbers» paper.

[2] B. Jack Copeland and Diane Proudfoot, «Turing and the Computer» в B. Jack Copeland, ed., *Alan Turing's Automatic Computing Engine: The Master Codebreaker's Struggle to Build the Modern Computer* (Oxford University Press, 2005), 116.

считали, что Тьюринг был привлечен к проекту Colossus[1], оказалось, что это не так. Он, конечно, знал о нем, но он «отклонил приглашение принять в нем непосредственное участие»[2]. Тем не менее влияние статьи Тьюринга на логическую конструкцию Colossus признается однозначно[3].

Гораздо более активное участие Тьюринг принимал в компьютерном проекте Национальной физической лаборатории (NPL) в Теддингтоне на юго-западе Лондона. В 1944 году директором NPL был сэр Чарльз Дарвин (1887–1962), чей дед опубликовал несколько значительных книг по биологии. Дарвин создал Отделение математики, которому была поручена работа по разработке автоматических вычислительных машин.

Дж. Р. Уомерсли (J. R. Womersley), руководитель Отделения математики, вызвал Тьюринга в NPL для беседы в июне 1945 года[4]. Уомерсли читал статью «О вычислимых числах» и хотел, чтобы Тьюринг спроектировал компьютер под названием «Автоматическая вычислительная машина» (Automatic Computing Engine, ACE), и если слово «машина» вызывало воспоминания о Чарльзе Бэббидже, то это было преднамеренно.

Изучив сообщение об EDVAC фон Неймана и предложив несколько собственных идей для компьютера, Тьюринг до конца 1945 года закончил «Предложение Отделению математики по разработке Автоматической вычислительной машины (или ABM)». В сообщении Тьюринга говорится, что в нем «приводится довольно подробный отчет о предлагаемом вычислительном устройстве», но рекомендуется, чтобы он «изучался вместе с “Сообщением о EDVAC” Дж. фон Неймана»[5].

Предложенная Тьюрингом машина была электронной, использовала двоичные числа и имела тактовую частоту 1 мегагерц, хотя биты передавались последовательно. В ней использовалась память на ртутных линиях задержки, в которых биты сохранялись в ртутных трубках в виде акустических импульсов. Пятифутовая ртутная трубка могла хранить 1024 бита. На прохождение от одного конца трубки до

---

[1] В том числе и я в Чарльз Петцольд, *Code: The Hidden Language of Computer Hardware and Software* (Microsoft Press, 1999), 244.

[2] Hodges, *Alan Turing*, 268.

[3] Hodges, *Alan Turing*, 554 (note 5.7).

[4] Введение к B. E. Carpenter and R. W. Doran, *A. M. Turing's ACE Report of 1946 and Other Papers* (MIT Press, 1986), 5–6.

[5] *ACE Report*, 21.

другого каждому биту требовалось около миллисекунды, после чего он мог быть выбран и возвращен в начало трубки. Тьюринг ожидал, что на сложение двух 32-битовых чисел потребуется 32 микросекунды (то есть по одному биту за такт), а 32-битовое умножение потребует «чуть более двух миллисекунд»[1].

Проектом Тьюринга предусмотрено довольно небольшое количество простейших команд, главным образом передач между памятью и регистрами. В этом смысле он напоминает современные компьютеры с сокращенным набором команд (Reduced Instruction Set Computers, RISC), которые привлекают быстрые аппаратные средства и выполняют наиболее сложные задачи программного обеспечения. И наконец, Тьюринг, по-видимому, изобрел *стек* – разновидность компьютерной памяти, похожую на стопку тарелок в кафетерии, поднимающихся вверх по колодцу с помощью пружины. Последняя «помещенная» (pushed) в стопку тарелка становится очередной, «выталкиваемой» (popped) из стопки. Процедуры Тьюринга для этих двух операций назывались BURY и UNBURY (от *bury* – прятать, *прим. пер.*)[2].

20 февраля 1947 года на лекции в Лондонском Математическом обществе Тьюринг изложил свой личный взгляд на вычислительную технику. «Вычислительные машины, подобные ACE... – это, по сути дела, практическое воплощение универсальной машины». Вся сложность работы, которую должна выполнить машина, «сосредоточена на ленте» – то есть в программе – «и никоим образом не влияет на универсальную машину»[3]. Конечно, Тьюринг осознавал значение быстродействия компьютера, но стремился подчеркнуть преимущества большой памяти:

> Я полагаю, что наличие достаточной памяти – ключевая проблема цифровых компьютеров, и, конечно, если они должны убедительно показать хоть какой-то истинный интеллект, им нужно предоставить гораздо больше, чем есть, возможностей. По-моему, эта проблема создания огромной памяти с довольно коротким временем доступа гораздо важнее, чем выполнение с высокой скоростью таких операций, как умножение[1].

Как и ожидалось, Тьюринг ясно понимал преимущества двоичных чисел над десятичными:

> Двоичная обработка – наиболее естественное, что делает любой полномасштабный компьютер. Гораздо проще работать в системе счисления по

---

[1] *ACE Report*, 116.
[2] *ACE Report*, 76.
[3] *ACE Report*, 112–113.
[4] *ACE Report*, 112.

основанию два, нежели в любой другой, поскольку так же просто создать механизмы, имеющие два устойчивых положения[1].

В конце доклада Тьюринг сказал о машинах, которые могут изменять свои собственные таблицы команд:

Они должны быть подобны ученику, который многому научился у своего учителя, но добавил еще больше своей работой. Как только это случится, уверен, любой обязательно разглядит проявление интеллекта машиной[2].

К сентябрю 1947 года Тьюринг был в расстроенных чувствах из-за отсутствия прогресса в работе над ACE. Он попросил годовой творческий отпуск за половину жалованья и отбыл в Кембридж. В NPL рассчитывали, что он должен вернуться не позже, чем через два года, но этого так и не случилось. (Опытный образец ACE не был готов до 1950 года, а это несколько расходилось с первоначальными планами Тьюринга.)

Вместо этого Тьюринг отправился в университет Манчестера, чтобы присоединиться к Максу Ньюмэну, работавшему там с 1945 года. Ньюмэн получил грант на новую Лабораторию вычислительных машин и строил вычислительную машину под названием Mark I. В июне 1948 года Mark I стала «первой законченной электронной вычислительной машиной типа EDVAC с хранимой программой»[3].

В сентябре Тьюринг влился в Манчестерский факультет математики и в проект Ньюмэна. Два месяца спустя было достигнуто соглашение с Ferranti Ltd., фирмой-производителем электроники, о разработке машины, которую Ferranti намеревалась продавать.

Тьюринг отвечал главным образом за программные аспекты Mark I. Примерно в 1951 году Тьюрингу поручили работу по написанию первого «Руководства программиста» для промышленной машины, где Тьюринг определил программирование как «деятельность, благодаря которой цифровую вычислительную машину заставляют выполнять намерения человека, выражая эти намерения должным образом на перфолентах»[4].

---

[1] *ACE Report*, 113–114.
[2] *ACE Report*, 123.
[3] Martin Campbell-Kelly, «Programming the Mark I: Early Programming Activity at the University of Manchester», *Annals of the History of Computing*, Vol. 2, No. 2 (April 1980), 134.
[4] Campbell-Kelly, «Programming the Mark I», 147. Книга доступна на www.alanturing.net/turing archive/archive/index/manchesterindex.html.

Вместо ртутных линий задержки в Mark I использовалась память на электронно-лучевых трубках (ЭЛТ). Этот тип памяти – нередко называемый трубкой Вильямса – впервые предложил Ф. К. Вильямс (F. C. Williams), приехавший в Манчестер в декабре 1946 года. Данные для сохранения направлялись в виде электрических импульсов в ЭЛТ, где они отображались на экране в виде массива точек, каждая из которых представляет один бит. Разная яркость или размер точки позволяли отличить 0 от 1. Металлическая пластина перед лицевой частью ЭЛТ улавливала заряды этих точек и позволяла ЭЛТ обновлять или считывать данные. Вторая ЭЛТ позволяла рассматривать точки и проверять данные. К 1947 году каждая ЭЛТ была способна хранить 2048 битов.

Данные в Mark I хранились в виде 40-битовых слов, в которых также можно было сохранить две 20-битовые команды. Поскольку слова отображались на ЭЛТ группами по 5 битов, была разработана запись по основанию 32, где каждые 5 битов представлялись телетайпным символом, соответствующим этому коду. Чтобы прочитать число, нужно было знать код, соответствующий каждому символу. Эти символы шли не по алфавиту, поэтому все, кто программировал на Mark I, следуя примеру Тьюринга, были вынуждены помнить 32-символьную последовательность:

/E@A:SIU½DRJNFCKTZLWHYPQOBG»MXV£

За участием Тьюринга в этих реальных компьютерных проектах мы можем потерять из виду тот очевидный факт, что целью его статьи «О вычислимых числах» было *не* построение универсального компьютера. Целью статьи было применение этого гипотетического компьютера для решения Entscheidungsproblem. Сделано еще несколько шагов. Решающий шаг – показать, что на самом деле машины Тьюринга ограничены в том, что они могут делать.

Во введении к своей статье Тьюринг заявил: «Хотя класс вычислимых чисел довольно большой и во многом похож на класс вещественных чисел, он тем не менее перечислим» (у него – страница 230; у меня – 88). В разделе 5 он показал, что «Каждой вычислимой последовательности соответствует хотя бы один описатель, тогда как ни одному описателю не соответствует более одной вычислимой последовательности. Следовательно, вычислимые последовательности и числа перечислимы» (у него – страница 241; у меня – 160).

Тем не менее некоторые сомнения могут еще оставаться. Понятно, что класс вычислимых чисел включает, по меньшей мере, какие-то

трансцендентные числа. Машины Тьюринга, вычисляющие π или *e*, или константу Лиувилля, безусловно, возможны. Трансцендентные числа, цифры которых выстроены по некоторому правилу, конечно, вычислимы машиной Тьюринга. Это – та самая игра, в которую играли Улам и фон Нейман во время совместного путешествия.

Однако подавляющее большинство – нет-нет-нет, давайте будем реалистами и скажем, *практически все* – трансцендентные числа – это, как правило, потоки случайных цифр. В царстве вещественных чисел правильные или вычислимые последовательности цифр – исключение, а полный и всеобщий беспорядок – правило.

Каким образом создать машину, которая вычисляет число без закономерности? Разве что генерировать цифры случайно?

Случайность – это то, что компьютерам не очень удается, и все же им нередко предоставляют право вести себя случайно. Случайные числа нужны некоторым статистическим приложениям, а компьютерным играм случайные числа нужны постоянно для изменения поведения. Без случайных чисел каждый расклад пасьянса был бы в точности тем же самым.

Языки программирования нередко предоставляют некоторое стандартное средство генерации случайных чисел. Например, компьютерная программа на языке программирования C может использовать функцию *rand* для получения случайного числа от 0 до 32767. Функция *rand* начинается с числа, называемого *начальным значением* (*seed*), которое по умолчанию устанавливается сначала в 1. Сам алгоритм в различных функциях *rand* может быть реализован по-разному; вот, например, как реализована функция *rand* в Microsoft C[1]:

```
int seed = 1;
int rand()
{
  return ((seed = seed * 214013 + 2531011) >> 16) & 32767;
}
```

Функция *rand* умножает *seed* на 214 013 и добавляет 2 531 011, а затем заносит этот результат обратно в *seed* для последующего вызова *rand*; но так как *seed* определено как 32-разрядное целое число со знаком, результатом может быть переполнение или потеря значимости. Результат вычисления усекается до 32 разрядов, и если самый старший разряд – 1, то значение фактически становится от-

[1] rand.c ©1985–1997, Microsoft Corporation. Некоторые детали в функции *rand* были изменены в целях ясности.

рицательным[1]. После этого результат сдвигается вправо на 16 разрядов или делится на 65 536 с отбрасыванием дробной части. Наконец, к этому результату и числу 32 767 применяется поразрядная логическая операция «И». Она выделяет младшие 15 разрядов результата и гарантирует, что он будет заключен между 0 и 32 767.

Даже если не вдаваться в эти замысловатые вычисления, должно быть ясно, что функция *rand* вовсе не генерирует случайные числа! Функция полностью детерминирована. Для заданного начального значения 1 повторные вызовы функции всегда приводят к вычислению одной и той же последовательности чисел:

41
18 467
6 334
26 500
19 169
. . .

При первом вызове программой функции *rand* она возвращает 41 и при 30 546-ом вызове программой функции *rand* она тоже возвращает 41, а затем цикл повторяется.

Поскольку эта последовательность полностью определяется начальным значением и алгоритмом, она действительно не случайна. Наоборот, ее называют *псевдослучайной последовательностью*. Если провести определенные статистические испытания над числами, сгенерированными с помощью *rand*, то окажется, что они проявляют случайный характер. В некоторых приложениях псевдослучайная последовательность предпочтительнее действительно случайных чисел, потому что можно воспроизвести результаты и убедиться, что программа работает правильно.

Однако в играх генерация одной и той же последовательности случайных чисел *нежелательна*. По этой причине всякий раз, сдавая карты в пасьянсе, программа начнет, наверное, с получения текущего времени дня вплоть до секунд и миллисекунд, а затем использует его для установки нового начального значения. В предположении, что карты не сдаются точно – до миллисекунд – в одно и то же время дня, получается якобы случайный расклад.

Джон фон Нейман однажды сказал, что «Всякий, кто питает страсть к арифметическим методам получения случайных чисел, несомненно

---

[1] Обсуждение переполнения и потери значимости может быть найдено в моей книге *Code*, 153–154.

впадает в грех»[1]. (Это было незадолго до того, как он описал арифметические методы для получения случайных чисел.) Поскольку компьютеры генерируют случайные числа неумело, приложения, которым *действительно* нужны случайные числа, выходят за пределы компьютера и используют устройства, специально предназначенные для таких задач. Аппаратный генератор случайных чисел (random number generator, RNG) может использовать для генерации случайных чисел «белый шум» или квантовые процессы.

Довольно любопытно, что идея аппаратной генерации случайных чисел исходит от Алана Тьюринга. Тьюринг потребовал, чтобы промышленная модель Mark I Манчестерского университета имела специальную команду, которая генерирует случайное число из источника шума. Он оказался не таким случайным, как должен быть, но достаточно случайным, чтобы помешать его надлежащей отладке[2].

Давайте предположим, что у нас есть работающее RNG-устройство. Применим его, как машину Тьюринга, для генерации последовательности нулей и единиц. Поставив перед ней двоичную запятую (и если нужно, нуль – *прим. перев.*), можно наблюдать, как прямо на наших глазах рождается вещественное число – несомненно, трансцендентное. (Если это попытаться сделать с программной псевдослучайной последовательностью, то начальное значение в конце концов вернется в исходное значение, а последовательность начнет повторяться. В итоге в числе появится повторяющаяся группа цифр, а это значит, что оно – рациональное.)

Теперь попробуем определить машину Тьюринга, которая генерирует то же самое вещественное число, что и RNG-устройство. Нам не удастся сделать этого. Единственный способ повторить результат RNG – создать машину Тьюринга, которая явно печатает именно те цифры, которые нам нужны, но она не будет машиной Тьюринга с конечным числом конфигураций.

Может быть, случайность большинства вещественных чисел – просто иллюзия. Хотя цифры числа π кажутся случайными, но π определенно вычислимо. Может быть, кажущиеся случайными вещественные числа на самом деле имеют в основе своей некую структуру, о которой мы еще не знаем. Может быть, если подойти к этому вопросу с другой стороны, можно доказать обратное – что вычислимые

---

[1] John von Neumann, *Collected Works, Volume V, Design of Computer, Theory of Automata, and Numerical Analysis* (Macmillan, 1963), 768. Первоначально сообщение было сделано на симпозиуме по методу Монте-Карло 1949 году.

[2] Martin Campbell-Kelly, «Programming the Mark I», 136.

числа *не* перечислимы. Тогда можно спать спокойно, будучи уверенными, что каждое вещественное число вычислимо.

Тьюрингу нужно противостоять курсу на такую перспективу.

8. *Применение диагонального процесса.*

Можно предположить, что аргументы, которые убеждают в том, что вещественные числа неперечислимы, должны убеждать и в том, что вычислимые числа и последовательности не могут быть перечислимыми*. Можно, например, предположить, что предел последовательности вычислимых чисел должен быть вычислимым.

* См. Hobson, *Theory of functions of a real variable* (2-е изд., 1921), 87, 88.

Тьюринг ссылается на первое (1874 года) доказательство Георга Кантора неперечислимости вещественных чисел, которое я приводил, начиная со страницы 41. Видимо, Тьюринг не имел доступа к оригинальной публикации Кантора, и вместо нее он ссылается на учебник Э. В. Хобсона, выпущенный издательством Кембриджского университета[1]. Хобсон очень близко придерживается Кантора, даже используя во многом те же обозначения.

Тьюринг предлагает повторить рассуждения Кантора для перечисления вычислимых, но не вещественных чисел. В обоих случаях числа стремятся к пределу. В доказательстве Кантора этот предел должен быть вещественным числом (чем же еще ему быть?), но Кантор смог показать, что предела нет в перечне вещественных чисел, доказав таким образом, что вещественные числа неперечислимы.

Когда тот же самый процесс применяется к вычислимым числам, они тоже стремятся к пределу. Будет ли этот предел вычислимым числом тоже? Ответ Тьюринга:

[1] Полный заголовок этой важной книги кембриджского профессора математики Эрнеста Уильяма Хобсона (1856–1933) *The Theory of Functions of a Real Variable and the Theory of Fourier's Series* (Теория функций действительной переменной и теория рядов Фурье) и первый выпуск был выпущен издательством Кембриджского университета в 1907 году. Второй выпуск, к которому обращается Тьюринг, датируется 1921 годом, но было добавлено так много нового материала, что второй том должен был быть издан в 1926 году. Этот том II также упоминался как второе издание. Том I был исправлен как третье издание в 1927 году. Третий выпуск тома I и второй выпуск тома II были переизданы «Харрен пресс» в 1950 году и «Дувр пресс» в 1957 году, и они могли бы быть изданиями, которые проще всего можно было разыскать. Обсуждение, к которому обращается Тьюринг, находится на страницах 84 и 85 этого третьего издания тома I. Оно сопровождается диагональным доказательством Кантора на страницах 85 и 86.

Ясно, что это истинно только тогда, когда последовательность вычислимых чисел определяется по некоторому правилу.

Под «последовательностью» Тьюринг понимает последовательности α и β, которые стремятся к пределу. Этот предел – вычислимое число, если только его можно вычислить, то есть разработать некий алгоритм, сообщающий нам числовой предел, к которому стремятся эти α и β. Это кажется невероятным. Если у нас нет способа вычислить данный предел, то он – невычислимое число. Он – еще одно невычислимое вещественное число, и, следовательно, мы не опровергли того, что вычислимые числа неперечислимы.

Или мы можем применить диагональный процесс.

Тьюринг заключает остальную часть параграфа в кавычки, как будто в его статью ворвался оппонент, стремящийся убедить нас, что вычислимые числа неперечислимы. Оппонент Тьюринга придерживается более тяжелых арифметических обозначений для диагонального процесса, чем предложенные мною на странице 45.

«Если вычислимые последовательности – перечислимые, то пусть α_n – n-я вычислимая последовательность и пусть $\phi_n(m)$ – m-я цифра α_n.

Это – лишь обозначение. Каждая вычислимая последовательность – это ряд из нулей и единиц, и каждая из этих двоичных цифр представляется греческой буквой ϕ. Вычислимые последовательности могут быть перечислены с обилием нижних индексов и обычных индексов примерно так:

$\alpha_1 = \phi_1(1)\ \phi_1(2)\ \phi_1(3)\ \phi_1(4)\ ...$
$\alpha_2 = \phi_2(1)\ \phi_2(2)\ \phi_2(3)\ \phi_2(4)\ ...$
$\alpha_3 = \phi_3(1)\ \phi_3(2)\ \phi_3(3)\ \phi_3(4)\ ...$
...

Пусть β – последовательность с $1 - \phi_n(n)$ в качестве ее n-й цифры.

Иными словами, β – диагональ из «перевернутых» нулей и единиц:

$\beta = (1 - \phi_1(1))\ (1 - \phi_2(2))\ (1 - \phi_3(3))\ (1 - \phi_4(4))\ ...$

Так как β вычислима, то существует номер K – такой, что $1 - \phi_n(n) = \phi_K(n)$ для всех n.

То есть для некоторого K существует α_K в перечне вычислимых чисел:

$\beta = \alpha_K = \phi_K(1)\ \phi_K(2)\ \phi_K(3)\ \phi_K(4)\ ...$

В общем случае для цифры n

$1 - \phi_n(n) = \phi_K(n)$

или

$1 = \phi_K(n) + \phi_n(n)$

Оппонент Тьюринга теперь использует этот арифметический аргумент, чтобы показать, что β не может существовать:

Полагая $n = K$,

то есть

$1 = \phi_K(K) + \phi_K(K)$,

мы имеем $1 = 2\phi_K(K)$, *т. е.* 1 – четное. Это невозможно. Следовательно, вычислимые последовательности неперечислимы».

Да, это интересно. Этот таинственный оппонент только что описал, как вычислить число, названное β, на основе перечисления вычислимых последовательностей, но этого вычисленного числа нет в списке. Следовательно, считает оппонент, вычислимые последовательности неперечислимы.

Но Тьюринг остается невозмутимым и говорит:

Ошибка этого вывода кроется в предположении, что β вычислимо.

Ошибка? В чем? Разве β может быть невычислимым? β вычислено по перечислению вычислимых последовательностей, поэтому оно должно быть вычислимым, правильно?

Да, но не совсем.

Давайте ненадолго вернемся назад. Тьюринг первоначально определил вычислимые числа как числа, которые вычисляются конечным образом. Он создал воображаемые машины для вычисления этих чисел и показал, что каждая машина может однозначно определяться положительным целым числом, называемым описателем. Поскольку целые числа перечислимы, машины Тьюринга тоже перечислимы, и потому перечислимы вычислимые последовательности.

В каком-то смысле перечисление машин Тьюринга так же просто, как перечисление положительных целых чисел:

1
2
3
4
5
. . .

Все машины Тьюринга появятся в этом списке в виде описателей, а по описателю можно получить описание машины и затем передать его Универсальной машине, чтобы получить вычислимую последовательность.

Конечно, что-то мы не учли: мы не учли способ точного определения, какие из положительных целых чисел этого списка – описатели ациклических машин.

Если вспомнить определения из раздела 2, то ациклическая машина – это машина, которая бесконечно печатает нули и единицы. Хотя машина, которая никогда не останавливается, может казаться «неконтролируемой» или «сумасшедшей», ациклические машины нужны для вычисления иррациональных чисел и рациональных чисел с бесконечно повторяющимися цифрами. Даже при печати таких рациональных чисел, как 0,1 (двоичный эквивалент $\frac{1}{2}$), лучше, чтобы машина была без циклов и печатала 1 и бесконечный ряд из 0:

0,10000000...

С другой стороны, *циклическая* машина – это машина, которая попадает в нежелательный цикл. Циклическая машина может, например, постоянно печатать цифру 0, не передвигая головки, или бесконечно печатать символы, отличные от 0 и 1.

Термины *ациклические* и *циклические* не очень удачны: ациклическая машина может провести остаток вечности в небольшом цикле, печатая цифры 0 или 1, и это может быть прекрасно. Циклическая

машина может попасть в тупик из-за того, что отправляется в несуществующую *m*-конфигурацию, и это только одна из многих бед, которые могут с ней случиться.

Нам нужно выделить описатели ациклических машин, так как только они годятся для интерпретации Универсальной машиной. Возможно, мы успешно перечислили все машины Тьюринга (где-то внутри этого списка положительных целых чисел), но не выделили из них ациклические, поэтому мы не можем использовать их для генерации вычислимых последовательностей.

Вполне понятно, что многие целые числа не являются описателями ни одной машины вообще. Мы можем легко установить (обычным просмотром или при помощи машины Тьюринга), является ли конкретное целое число *правильным* описателем, что означает, что оно разбивается на правильные команды, каждая из которых начинается с *m*-конфигурации, и т. д. Мы можем даже установить, обращается ли машина к *m*-конфигурациям, которых не существует. Мы могли бы проверить, что определенные *m*-конфигурации не используются. Мы также легко могли бы проверить, включают ли какие-либо *m*-конфигурации команды, которые действительно печатают цифры 0 или 1. Такой процесс установил бы, что наименьший правильный описатель – 31 334 317, и это – циклическая машина. (Она всего лишь печатает пустые клетки.) До номера 313 324 317 не появится ни одной ациклической машины, а до номера 313 325 317 мы не встретим ни одной ациклической машины, печатающей слева направо.

Вот самое начало перечисления положительных целых чисел, где появляются первые две ациклические машины Тьюринга с печатью вправо:

1
2
3
4
5
. . .
313325317 ← Печатает нули вправо
. . .
3133225317 ← Печатает единицы вправо
. . .

Конечно, они – простейшие из простых машин, и метод их идентификации так же прост. Гораздо труднее – на самом деле, как покажет

Тьюринг, невозможно – построить машину, которая реализует *общий процесс* определения, является ли конкретное целое число описателем ациклической машины.

Этот общий процесс – как раз в том, что нам нужно произвести диагонализацию. Каждая цифра β берется из другого вычислимого числа, поэтому для вычисления β нужно, чтобы были известны все ациклические машины Тьюринга. Тьюринг покажет, что эти ациклические машины не могут быть опознаны конечным образом, а это значит, что мы не можем явно перечислить вычислимые последовательности. Поэтому то, что β – вычислимая последовательность, просто неверно.

> Оно было бы верным, если бы мы могли перечислить вычислимые последовательности конечным образом, но задача перечисления вычислимых последовательностей равносильна задаче выяснения, является ли данное число D.N (описателем – *прим. пер.*) ациклической машины, а у нас нет общего процесса, чтобы сделать это за конечное число шагов.

Теперь Тьюринг едва заметно смещает фокус. Он начал с попытки применить к вычислимым последовательностям метод диагонализации Кантора, но теперь ему нужно просто проверить, что происходит при попытке определить все описатели ациклических машин.

> Фактически, правильно применяя замысел диагонального процесса, мы можем показать, что такого общего процесса не может быть.

Если, как утверждает Тьюринг, не существует общего процесса определения, является ли конкретное целое число описателем ациклической машины, то β невычислимо. Это сделало бы «доказательство» оппонента о неперечислимости вычислимых последовательностей недействительным. Тогда уже ничто не поколебало бы нашей уверенности в том, что вычислимые последовательности действительно перечислимы и, следовательно, не могут включать все вещественные числа.

К сожалению, Тьюринг начинает следующий абзац весьма расплывчато:

> Простейшее и самое прямое доказательство этого – показать, что если существует общий процесс, то существует машина, которая вычисляет β.

Думаю, он имеет в виду, что не может быть общего процесса определения, является ли конкретная машина ациклической, потому что если бы он был, то мы могли бы вычислить β, а мы знаем, что не можем вычислить β, потому что тогда диагональный метод был бы правильным и вычислимые последовательности не были бы перечислимыми.

Это доказательство, хоть и кажется безупречным, имеет недостаток, который может оставить читателя с чувством, что «здесь что-то не так».

Сомнение все еще гложет нас. По этой причине Тьюринг докажет теперь напрямую, что нет машины, которая определит, является ли конкретное целое число описателем ациклической машины.

Доказательство, которое я приведу, не имеет этого недостатка и дает некоторое понимание значения идеи «ацикличности».

Он мог бы еще добавить, что выводы выходят далеко за рамки этого небольшого примера из теории чисел.

Все, что теперь нужно Тьюрингу, – машина, которая извлекает по одной цифре из каждой вычислимой последовательности. Он не хочет возиться с вычитанием цифр из единицы. На самом деле он собирается предпринять попытку вычислить нечто более простое, чем β:

Оно связано не с построением β, а с построением β', чья n-я цифра есть $\phi_n(n)$.

На самой первой странице своей статьи (страница 88 этой книги) Тьюринг сказал: «Однако вычислимые числа не включают все определяемые числа, и приводится пример определяемого числа, которое невычислимо». И β, и β' – такие определяемые числа. β' – определяемое, потому что можно дать инструкцию по его вычислению: составьте перечень всех чисел, начиная с 1. Для каждого числа определите, является ли оно правильным описателем машины Тьюринга. Если да, то проверьте, что эта машина ациклическая. Если это так, вычислите это число до n-ой цифры (где n на 1 больше количества уже найденных ациклических машин). Эта цифра – n-ая цифра β'.

Можно видеть, что число β' полностью определено, но можно ли его вычислить?

Хотя у Тьюринга нет команды, которая останавливала бы машину, проблема, которую он сейчас атакует, изучается скорее как разновидность *Проблемы останова*. (Термин впервые появился в 1958 году в книге Мартина Дэвиса «Computability and Unsolvability»[1].) Можно ли определить такую машину Тьюринга, которая выяснит, остановится ли некоторая машина Тьюринга или она будет работать вечно? Если заменить останов цикличностью, получится подобная проблема. Может ли одна машина Тьюринга проанализировать другую машину Тьюринга и определить ее окончательную судьбу?

Тьюринг начинает с предположения о существовании машины, которая определяет, что какая-то произвольная машина – ациклическая машина. В следующем обсуждении он ссылается не на описатель (D.N), а на описание (S.D) машины, но это на самом деле не имеет значения, так как преобразование между ними тривиально.

[247]

Предположим, что такой процесс существует; то есть мы утверждаем, что можем создать машину $\mathcal{D}$, которая, будучи применена к S.D любой вычислительной машины $\mathcal{M}$, проверит это S.D и, если $\mathcal{M}$ – циклическая, пометит S.D символом «*u*», а если она без циклов – символом «*s*».

Машина $\mathcal{D}$ – Решающая машина (Decision machine). «*u*» обозначает *неприемлемую* (то есть циклическую машину), а «*s*» – *приемлемую* (ациклическую машину). Тьюринг определил эти термины в конце раздела 5 (у него – страница 241, у меня – 164).

Объединив машины $\mathcal{D}$ и $\mathcal{U}$, мы могли бы создать машину $\mathcal{H}$ для вычисления последовательности β'.

Фактически машина $\mathcal{H}$ тоже должна генерировать положительные целые числа, а затем преобразовывать их в описания, но это довольно просто. Для каждого генерируемого машиной $\mathcal{H}$ положительного целого числа она применяет $\mathcal{D}$ для решения вопроса, определяет ли это число приемлемую машину. Если это так, то $\mathcal{H}$ передает это описание

---

[1] Martin Davis, *Computability and Unsolvability* (McGraw-Hill, 1958), 70. Дэвис полагает, что он первым использовал термин в лекциях в 1952 г. (See Copeland, *The Essential Turing*, 40, footnote 61.) Понятие также обнаруживается в главе 13 книги Stephen Cole Kleene, *Introduction to Metamathematics* (Van Nostrand, 1952).

универсальной машине $\mathcal{U}$ для вычисления последовательности. Для n-ой вычислимой последовательности машине $\mathcal{U}$ необходимо лишь прогнать приемлемую машину до n-ой цифры. Тогда эта цифра становится n-ой цифрой β'. Поскольку $\mathcal{U}$ работает под управлением $\mathcal{H}$, последняя может остановить первую, как только получит нужную цифру.

Сначала машине $\mathcal{H}$ с помощью $\mathcal{D}$ нужно проверить описание, так как нежелательно, чтобы $\mathcal{U}$ застряла, отрабатывая неприемлемую машину. Если $\mathcal{H}$ передает $\mathcal{U}$ описание неприемлемой машины и эта неприемлемая машина никогда не печатает цифру, то процесс застрянет и не сможет двинуться вперед.

Тьюринг в действительности не показывает нам, как выглядит эта чудесная Решающая машина $\mathcal{D}$, потому что это было бы большой подсказкой, что такой машины не может быть. Даже без подготовки видно, что она должна быть довольно сложной. Может ли $\mathcal{D}$ решить, что данная машина ациклическая, не повторив ее и не проследив каждый ее шаг?

Во всяком случае, машина $\mathcal{D}$ подобна $\mathcal{U}$ в том, что работает с описанием (или, что то же, с описателем), записанным на ленте.

Машине $\mathcal{D}$ может потребоваться лента. Мы можем предположить, что она использует помимо всех символов в F-клетках E-клетки, и когда она вынесет свое решение, вся выполненная $\mathcal{D}$ черновая работа стирается.

После нее остается только s или u как окончательное решение.

Движение машины $\mathcal{H}$ поделено на участки. В первых $N - 1$ участках, помимо прочего, машиной $\mathcal{D}$ выписываются и проверяются целые числа 1, 2, ..., $N - 1$.

Выражение «поделено на участки» не означает наличие отдельных частей $\mathcal{H}$, обрабатывающих отдельные числа. Отдельные множества конфигураций для каждого числа означали бы, что $\mathcal{H}$ должна быть бесконечной. На самом деле Тьюринг имеет в виду последовательные во времени действия. Настоящий процесс должен быть общим и применяться ко всем целым числам: машина $\mathcal{H}$ генерирует положительные целые числа одно за другим, передает очередное число в $\mathcal{D}$ для определения, приемлемо ли оно, и, если так, применяет $\mathcal{U}$ для вычисления определенного количества цифр вычислимой последовательности.

Некоторое их количество, скажем $R(N - 1)$, уже обнаружено, и они должны быть D.N ациклических машин.

R лишь подсчитывает количество уже обнаруженных ациклических машин. R нужно машине для определения того, сколько цифр нужно вычислить для каждой найденной ациклической машины.

На N-ом участке машина $\mathcal{D}$ проверяет число N. Если N – приемлемое, *т. е.* если оно – D.N ациклической машины, то $R(N) = 1 + R(N - 1)$ и вычисляются первые $R(N)$ цифр последовательности, D.N которой есть N.

Если N равно, например, 3 133 225 317, то $R(N - 1)$ равно 1. (См. выше список положительных целых чисел с двумя первыми обнаруженными приемлемыми машинами.) До этого была обнаружена лишь одна приемлемая машина. Машина $\mathcal{D}$ определит, что N – это действительно описатель ациклической машины. Таким образом, $R(N)$ устанавливается в 2, и $\mathcal{U}$ вычисляет первые две цифры машины, определяемой числом 3 133 225 317. Обе эти цифры – 1. $\mathcal{H}$ использует вторую из них как вторую цифру β'. Все идет своим чередом!

$R(N)$-я цифра этой последовательности записывается как одна из цифр последовательности β', вычисляемой $\mathcal{H}$.

Обычный случай, конечно, когда описатель – или вообще не номер машины, или номер циклической машины.

Если N не приемлемо, то $R(N) = R(N - 1)$ и машина переходит к $(N + 1)$-му участку своего движения.

Дело в том, что $\mathcal{H}$ должна просматривать возможные описатели один за другим и для каждого приемлемого описателя $\mathcal{H}$ должна исполнить машину до $R(N)$-ой цифры.

Тьюринг теперь прилагает большие усилия, чтобы показать, что $\mathcal{H}$ – ациклическая. $\mathcal{H}$ просто выполняет $\mathcal{D}$ для каждого возможного описателя, а $\mathcal{D}$, согласно исходному предположению, – ациклическая машина.

Из построения $\mathcal{H}$ мы можем видеть, что $\mathcal{H}$ – ациклическая. Каждый участок движения $\mathcal{H}$ заканчивается через конечное число шагов. Поскольку, согласно нашему предположению о $\mathcal{D}$, решение о том, что N приемлемо, достигается за конечное число шагов. Если N не при-

емлемо, то N-й участок завершается. Если N приемлемо, это означает, что машина $\mathcal{M}(N)$, чей D.N есть N, – ациклическая, и поэтому ее $R(N)$-я цифра может быть вычислена за конечное число шагов. Когда эта цифра вычислена и записана как $R(N)$-я цифра β', N-й участок завершается. Следовательно, $\mathcal{H}$ – ациклическая.

Машина $\mathcal{H}$ – машина Тьюринга, поэтому у $\mathcal{H}$ есть описатель (который Тьюринг называет K). В некоторый момент $\mathcal{H}$ должна будет иметь дело со своим собственным описателем. $\mathcal{H}$ должна будет определить, ациклична ли она сама.

Теперь пусть K будет D.N машины $\mathcal{H}$. Что делает $\mathcal{H}$ на K-м участке своего движения? Она должна проверить, приемлемо ли K, приняв решение «*s*» или «*u*». Так как K – это D.N машины $\mathcal{H}$ и так как $\mathcal{H}$ – ациклическая, решением не может быть «*u*».

Потом Тьюринг также добавляет:

С другой стороны, решением не может быть «*s*».

Основная проблема – в том, что $\mathcal{H}$ попадает в бесконечную рекурсию. Прежде чем прийти к числу K (своему описателю), $\mathcal{H}$ должна проверить все положительные целые числа от 1 до $K - 1$. К этому моменту количество машин без циклов – $R(K - 1)$ и определены первые $R(K - 1)$ цифр β'.

Какова $R(K)$-я цифра β'? Чтобы получить эту цифру, $\mathcal{H}$ должна проследить свои собственные действия, что означает, что она должна повторить все до того места, где встретила K, после чего процесс начинается заново. Именно поэтому $\mathcal{H}$ не может быть ациклической.

Ибо, если бы это было, то на K-м участке своего движения $\mathcal{H}$ была бы обязана вычислить первые $R(K - 1) + 1 = R(K)$ цифр последовательности, вычисляемой машиной с K в качестве ее D.N, и записать $R(K)$-ю цифру как цифру последовательности, вычисляемой $\mathcal{H}$. Вычисление первых $R(K) - 1$ цифр было бы выполнено хорошо, но команды для вычисления $R(K)$-й были бы равносильны тому, чтобы «вычислить первые $R(K)$ цифр, вычисленные H, и записать $R(K)$-ю». Эта $R(K)$-я цифра никогда не была бы найдена.

(В предпоследней строке вместо H должно быть $\mathcal{H}$.)

Машина $\mathcal{H}$ генерирует последовательность цифр на основании последовательностей, сгенерированных другими машинами. Довольно просто, когда мы представляем себе машины, генерирующие такие последовательности, как двоичные эквиваленты 1/3, π и $\sqrt{2}$, но откуда $\mathcal{H}$ получит цифру K той последовательности, которую она генерирует? Она должна получить эту цифру от самой себя, но это бессмысленно, поскольку $\mathcal{H}$ получает цифры только от других машин.

Ладно, стало быть, у $\mathcal{H}$ есть небольшая проблема при столкновении с ее собственным описателем. А нельзя ли его просто пропустить? Ну да, можно, но, как мы видели, любую вычислимую последовательность можно вычислить разными машинами. Одну и ту же последовательность машины могут вычислять по-разному или могут иметь лишние команды. Подобные машины $\mathcal{H}$ должна тоже пропускать. А как быть с машинами, которые вычисляют не β', а что-то близкое к ней, например β' с переставленными местами 27-й и 54-й цифрами? Таких машин – бесконечное множество, и избавление от всех них ложится настоящим бременем на $\mathcal{H}$ – *непосильным* бременем.

Т. е. $\mathcal{H}$ – циклическая, в противовес тому, что мы выяснили в последнем абзаце, и решению «*s*». Таким образом, оба решения невозможны, и мы заключаем, что машины $\mathcal{D}$ не может быть.

Не может быть общего процесса определения того, что машина – ациклическая. Отсюда следует, что не может быть компьютерной программы, которая определит окончательную судьбу других компьютерных программ.

Тьюринг также разрешил парадокс диагонального процесса: сначала он установил, что вычислимые числа перечислимы, хотя диагональный процесс должен указывать, что можно создать вычислимое число не из списка. Тьюринг показал, что диагональ *не* может быть вычислена конечным образом и, следовательно, невычислима. Вычислимые числа могут быть перечислимыми, но в действительности их нельзя перечислить за конечное число шагов.

Тьюринг еще не покончил с этим разделом. Теперь он выдвигает гипотезу о машине $\mathcal{E}$, что могло бы означать «ever print» («напечатает когда-нибудь»).

[248]
Далее можно показать, что *не может быть машины* $\mathcal{E}$, *которая, получив S.D произвольной машины* $\mathcal{M}$, *определит, напечатает ли* $\mathcal{M}$ *когда-нибудь заданный символ* (*скажем, 0*).

Эта машина $\mathcal{E}$ понадобится Тьюрингу в заключительном разделе статьи, где он использует ее для доказательства неразрешимости Entscheidungsproblem. Здесь же он докажет, что $\mathcal{E}$ не может существовать, показав сначала, что из существования $\mathcal{E}$ следует существование процесса определения, печатает ли машина 0 бесконечное число раз, но это влечет существование похожего процесса определения, печатает ли машина 1 бесконечное число раз. Если бы была возможность установить, печатает ли машина 0 бесконечное число раз или 1 бесконечное число раз (или того и другого), то была бы возможность установить, ациклична ли машина. Уже было доказано, что такой процесс невозможен, поэтому машина $\mathcal{E}$ тоже должна быть невозможна.

> Сначала покажем, что если существует машина $\mathcal{E}$, то существует и общий процесс для определения, печатает ли данная машина $\mathcal{M}$ 0 бесконечное число раз.

Тьюринг показывает это довольно необычным методом определения модификаций произвольной машины $\mathcal{M}$.

> Пусть $\mathcal{M}_1$ – машина, которая печатает ту же самую последовательность, что $\mathcal{M}$, за исключением того, что в позиции, где $\mathcal{M}$ печатает первый 0, $\mathcal{M}_1$ печатает $\overline{0}$. $\mathcal{M}_2$ должна заменять два первых символа 0 на $\overline{0}$, и т. д. Так, если $\mathcal{M}$ должна напечатать
>
> $A\ B\ A\ 0\ 1\ A\ A\ B\ 0\ 0\ 1\ 0\ A\ B$..,
>
> то $\mathcal{M}_1$ напечатала бы
>
> $A\ B\ A\ \overline{0}\ 1\ A\ A\ B\ 0\ 0\ 1\ 0\ A\ B$..,
>
> а $\mathcal{M}_2$ напечатала бы
>
> $A\ B\ A\ \overline{0}\ 1\ A\ A\ B\ \overline{0}\ 0\ 1\ 0\ A\ B$...

Если у нас есть машина $\mathcal{M}$, можно ли определить машину, которая читает описание $\mathcal{M}$ и производит описания $\mathcal{M}_1$, $\mathcal{M}_2$ и т. д.? Тьюринг говорит: да – и называет эту машину $\mathcal{F}$:

> Теперь пусть $\mathcal{F}$ – это машина, которая, получив S.D $\mathcal{M}$, будет записывать последовательно S.D $\mathcal{M}$, $\mathcal{M}_1$, $\mathcal{M}_2$, ... (такая машина существует).

Чтобы убедиться, что $\mathcal{F}$ возможна, давайте рассмотрим очень простую машину, которая печатает по очереди без пропусков 0 и 1, то есть двоичное представление 1/3:

q_1	None	$P0, R$	q_2
q_2	None	$P1, R$	q_1

Это – машина $\mathcal{M}$. Вот машина $\mathcal{M}_1$:

q_1	None	$P\bar{0}, R$	q_4
q_2	None	$P1, R$	q_1
q_3	None	$P0, R$	q_4
q_4	None	$P1, R$	q_3

Обе исходные конфигурации были просто размножены и получили другие m-конфигурации. В первой группе конфигураций каждая строка, печатавшая 0, теперь печатает $\bar{0}$, а затем переходит к соответствующей конфигурации второй группы. У $\mathcal{M}_2$ – три группы:

q_1	None	$P\bar{0}, R$	q_4
q_2	None	$P1, R$	q_1
q_3	None	$P\bar{0}, R$	q_6
q_4	None	$P1, R$	q_3
q_5	None	$P0, R$	q_6
q_6	None	$P1, R$	q_5

Можно заметить, что эти модифицированные машины никогда не попадают в конфигурацию q_2, но это – только счастливая случайность для этой конкретной машины.

Поэтому вполне возможно, что $\mathcal{F}$ существует. Отметим зависимость между этими $\mathcal{M}$-машинами. Если $\mathcal{M}$ никогда не печатает 0, то этого не делают ни $\mathcal{M}_1$, ни $\mathcal{M}_2$, и т. д. Если $\mathcal{M}$ печатает 0 только однажды, то никогда не печатает 0 ни $\mathcal{M}_1$, ни $\mathcal{M}_2$ и т. д. Если $\mathcal{M}$ печатает 0 дважды, то $\mathcal{M}_1$ печатает 0 один раз, $\mathcal{M}_2$ никогда не печатает 0 и т. д. Если $\mathcal{M}$ печатает 0 бесконечное число раз, то это же делают $\mathcal{M}_1$, $\mathcal{M}_2$ и т. д.

Вы помните, что $\mathcal{E}$ по предположению должна определить, напечатает ли машина когда-нибудь 0.

Объединяем $\mathcal{F}$ и $\mathcal{E}$ и получаем новую машину $\mathcal{G}$. В $\mathcal{G}$ сначала применяется $\mathcal{F}$, чтобы записать S.D $\mathcal{M}$, а затем $\mathcal{E}$ проверяет его, записывается : 0 :, если оказывается, что $\mathcal{M}$ никогда не печатает 0; затем $\mathcal{F}$ записывает S.D машины $\mathcal{M}_1$, и проверяется она, печатая : 0 : тогда и только тогда, когда $\mathcal{M}_1$ никогда не печатает 0, и т. д.

Машина $\mathcal{G}$ использует $\mathcal{F}$ для генерации описателей $\mathcal{M}$, $\mathcal{M}_1$, $\mathcal{M}_2$ и т. д., а $\mathcal{E}$ – для определения того, печатает ли очередная машина когда-нибудь 0. Если очередная машина никогда не печатает 0, $\mathcal{G}$ печатает 0.

Итог таков: если $\mathcal{M}$ никогда не печатает 0 или печатает 0 только конечное число раз, то $\mathcal{G}$ печатает 0 бесконечное число раз. Если $\mathcal{M}$ печатает 0 бесконечное число раз, то $\mathcal{G}$ никогда не печатает 0.

Теперь давайте проверим $\mathcal{G}$ с $\mathcal{E}$. Если оказывается, что $\mathcal{G}$ никогда не печатает 0, то $\mathcal{M}$ печатает 0 бесконечное число раз; если $\mathcal{G}$ печатает 0 иногда, то $\mathcal{M}$ не печатает 0 бесконечное число раз.

Это означает, что $\mathcal{G}$ может сказать нам, что $\mathcal{M}$ печатает 0 бесконечное число раз. Она говорит нам это, никогда не печатая 0.

Аналогично существует общий процесс для определения того, печатает ли $\mathcal{M}$ бесконечное число раз 1. Объединив эти процессы, мы получим процесс для определения, будет ли $\mathcal{M}$ печатать цифры бесконечно, *т. е.* получим процесс для определения, без циклов ли машина $\mathcal{M}$. Поэтому машины $\mathcal{E}$ не может быть.

Еще одним доказательством от противного Тьюринг показал, что $\mathcal{E}$ не может существовать, потому что это в конечном счете предполагало бы существование $\mathcal{D}$ – машины, которая определяет, является ли некая машина ациклической машиной, – а такой машины не может быть.

Тьюринг завершает данный раздел напоминанием о том, что нужно по-настоящему исследовать эту предполагаемую равносильность человека-вычислителя и машины Тьюринга, потому что мы весьма полагались на нее.

Всюду в этом разделе выражение «существует общий процесс определения...» употреблялось как эквивалент «существует машина, которая определит...». Такое употребление может быть оправдано тогда и только тогда, когда мы можем подтвердить наше определение «вычислимости».

Это исследование войдет в следующий раздел.

Затем Тьюринг намекает на другой аспект данного доказательства, который не будет рассматриваться вплоть до части III этой книги. Тьюринг начал с предположения, что выход машин Тьюринга – «вычислимое число», но машины могут быть более гибкими, чем такие машины. Например, рассмотрим машину, которая печатает такую последовательность:

0011010100010100010100010000010...

Возможно, она и напоминает число, но на самом деле она – выход машины «Простое число», которую можно было бы обозначить *IsPrime*(*n*). *n*-я цифра этой последовательности (начиная с $n = 0$) есть 1, если *n* – простое число, и 0, если *n* не простое число. Напечатанная машиной последовательность говорит, что 2, 3, 5, 7, 11, 13, 17, 19, 23 и 29 – простые числа. Такая машина вполне возможна, но она фактически не вычисляет числа. Вместо этого она сообщает нам что-то о натуральных числах.

> Для каждого из этих «общих процессов» проблему можно свести к проблеме об общем процессе определения, обладает ли данное целое число *n* свойством *G*(*n*) [например, *G*(*n*) могло бы означать «*n* приемлемо» или «*n* есть Гёделево представление доказуемой формулы»], и это равнозначно вычислению числа, чья *n*-ая цифра есть 1, если *G*(*n*) истинно, и 0, если ложно.

Теперь Тьюринг – в самой малой степени, которая проявится более в части III этой книги, – установил связь между его вычислительными машинами и математической логикой. Символы 1 и 0 служат не только в качестве двоичных цифр, но они – как много лет тому назад понял Джордж Буль – могут также обозначать *истину* и *ложь*.

Рассмотрим группу функций, аргументы которых – натуральные числа, а возвращаемые значения – *истина* и *ложь* (или 1 и 0):

IsPrime(*n*)
IsEven(*n*)
IsOdd(*n*)
IsLessThanTen(*n*)
IsMultipleOfTwentyTwo(*n*)

и т. д. Их иногда называют булевыми функциями, и они могут быть реализованы машинами Тьюринга, которые печатают последовательности из символов 0 и 1 для *n*, равного 0, 1, 2, 3 и т. д. Функция *IsOdd* печатает ту же чередующуюся последовательность, что и машина из первого примера Тьюринга.

Тьюринг установил, что эти вычислимые последовательности перечислимы. Как, впрочем, и сами имена функций! Их можно упорядочить, например, по алфавиту. В обозначениях Кантора для трансфинитных чисел мощность (или кардинальное число) множества всех вычислимых и упорядоченных по алфавиту булевых функций от натуральных чисел – $\aleph_0$, мощность перечислимых множеств.

Каждая булева функция возвращает *истину* или 1 для подмножества натуральных чисел. Например, *IsPrime* возвращает 1 для следующего подмножества натуральных чисел:

{2, 3, 5, 7, 11, 13, ...}

Каждой из этих булевых функций сопоставлены разные подмножества натуральных чисел. Как можно помнить из главы 2, множество всех подмножеств называется *степенным множеством*, и если исходное множество имеет мощность $\aleph_0$, то *степенное множество* имеет мощность $2^{\aleph_0}$.

Множество всех мыслимых булевых функций имеет мощность $2^{\aleph_0}$, хотя множество всех *вычислимых* булевых функций (а именно множество всех булевых функций, которые можно записать именем на английском языке) имеет мощность $\aleph_0$. Это еще один большой разрыв между мыслимым и вычислимым.

Глава 11

О машинах и людях

В начале первого раздела своей статьи (страница 89 этой книги) в определении вычислимых чисел Алан Тьюринг сказал, что «вплоть до §9 не будет ни одной попытки обосновать данное определение». И вот мы добрались до раздела 9, и следующие ниже страницы были названы биографом Тьюринга Эндрю Ходжесом «самыми замечательными из когда-либо встречавшихся в математических статьях»[1].

Тьюринг сделает попытку обосновать равенство возможностей машины Тьюринга и человека-вычислителя (human computer), выполняющего строго определенные математические операции. Следовательно, если алгоритмический процесс неразрешим машиной Тьюринга, то он неразрешим и человеком. Эта идея – выражаемая, как правило, более формально – стала известной как тезис Тьюринга или (близкой по форме) тезис Чёрча–Тьюринга. Она названа «тезисом», потому что представляется слишком аморфной, чтобы подвергнуться строгому математическому доказательству. Тем не менее этот тезис распространяется и на другие цифровые компьютеры: их вычислительные возможности не больше, чем у машин Тьюринга.

В этой главе представлена только первая часть раздела 9; оставшаяся часть требует некоторой подготовки в области математической логики и будет отложена до части III этой книги. В большинстве случаев я не буду прерывать анализ Тьюринга. Вот краткое заключение Мартина Дэвиса:

> «Анализ» Тьюринга – это выдающийся пример прикладной философии, когда, начав с человека, выполняющего вычисления, он приходит путем устранения несущественных деталей через цепь упрощений к конечному результату – хорошо знакомой модели, состоящей из устройства с конечным числом состояний, работающего с полубесконечной линейной лентой[2].

---

[1] Andrew Hodges, *Alan Turing: The Enigma* (Simon & Schuster, 1983), 104.

[2] Martin Davis, «Why Gödel Didn't Have Church's Thesis», *Information and Control*, Vol. 54, Nos. 1/2, (July/Aug. 1982), 14.

[249]

9. Пространство вычислимых чисел.

Еще не было ни одной попытки показать, что «вычислимые» числа включают все числа, которые было бы естественно считать вычислимыми. Все аргументы, которые можно привести, вынуждены обращаться в основном к интуиции и по этой причине довольно неудовлетворительны математически. Действительно нерешенный вопрос – «Каковы возможные процессы, которые могут происходить при вычислении числа?»

Аргументы, которые я буду использовать, – трех видов.

(*a*) Непосредственное обращение к интуиции.

(*b*) Доказательство эквивалентности двух определений (в случае, если новое определение имеет большую интуитивную привлекательность).

(*c*) Приведение примеров больших классов чисел, которые являются вычислимыми.

Аргумент (*b*) – в части III этой книги; аргумент (*c*) отложен до раздела 10 статьи Тьюринга.

Как только допускается, что все вычислимые числа «вычислимы», из этого следует еще несколько утверждений того же характера. В частности, из этого следует, что если существует общий процесс проверки доказуемости формулы гильбертова функционального исчисления, то такую проверку можно осуществить механически.

«Гильбертово функциональное исчисление» – это логико-математическая система, всеми называемая сегодня «логикой предикатов первого порядка». Именно в этой логике Гильберт определил Entscheidungsproblem. Вряд ли Тьюринг знал о том, что процесс, который «можно осуществить механически», – это именно то, к чему призывал Генрих Бехман (Heinrich Behmann) при самых первых упоминаниях Entscheidungsproblem (страница 68). Обращение Бехмана до недавнего времени оставалось неопубликованным.

I. [Тип (*a*)]. Этот аргумент – лишь развитие идей §1.

Вычисление обычно выполняется путем записи определенных символов на бумаге. Мы можем предположить, что эта бумага делится на клетки подобно детской тетради по арифметике. Двумерность

бумаги порой полезна в элементарной арифметике. Но такого ее использования всегда можно избежать, и, я думаю, можно согласиться с тем, что двумерность бумаги не существенна для вычисления. Тогда я предполагаю, что вычисление выполняется на одномерной бумаге, *т. е.* на ленте, разделенной на клетки. Я также предположу, что число символов, которые можно печатать, конечно. Если допустить бесконечное число символов, то существовали бы символы, отличающиеся в довольно малой степени†. Влияние такого ограничения на число символов не очень важно. Всегда можно использовать вместо отдельных символов их последовательности. Так, такие арабские числа, как 17 или 999999999999999, обычно считаются отдельными [250]
символами. Также в любом европейском языке слова считаются отдельными символами (китайский язык, однако, намерен иметь счетную бесконечность символов). С нашей точки зрения, разница между простыми и составными символами – в том, что составные символы, если они слишком длинны, нельзя охватить одним взглядом. Это согласуется с опытом. Мы не можем сразу сказать, одинаковы ли 9 999 999 999 999 999 и 999 999 999 999 999.

† Если считать, что символ буквально печатается в клетке, то можно предположить, что эта клетка $0 \le x \le 1$, $0 \le y \le 1$. Символ задается множеством точек в этой клетке, то есть множеством, заполненным типографской краской. Если эти множества ограничены, чтобы быть измеримыми, мы можем определить «расстояние» между двумя символами как стоимость преобразования одного символа в другой, если стоимость перемещения единицы площади типографской краски на единицу расстояния принять за 1 и есть бесконечный запас краски при x = 2, y = 0. При такой топологии символы образуют условно компактное пространство.

В следующем абзаце Тьюринг употребляет слово «вычислитель». Конечно же, он ведет речь о *человеке*-вычислителе.

Поведение вычислителя в любой момент определяется символами, которые он обозревает, и его «состоянием памяти» в этот момент. Мы можем предположить, что существует предел B числа символов или клеток, которые вычислитель может обозревать одномоментно. Если он желает видеть больше, он должен использовать последовательные просмотры. Мы также предположим, что число состояний памяти, которые должны приниматься во внимание, конечно. Причины этого – того же характера, что и те, что ограничивают число символов. Если допустить бесконечное число состояний памяти, то

некоторые из них будут «произвольно близкими» и приведут к путанице. И вновь это ограничение не такое, чтобы серьезно влиять на вычисление, так как использования более сложных состояний памяти можно избежать, записывая больше символов на ленте.

О проводимом Тьюрингом в этом разделе анализе Курт Гёдель в 1972 году написал короткую заметку, которую назвал «Философская ошибка в работе Тьюринга»[1]. Гёдель утверждал, что «*память при ее использовании не статична, а постоянно развивается*» и что число умственных состояний памяти может даже стремиться к бесконечности. Эти расхождения во мнениях представляют собой основной конфликт между теми, кто считает, что мышление должно быть в конечном счете механическим процессом мозга, и теми, кто так не считает.

Давайте представим, что операции, выполняемые вычислителем, разбиты на «простые операции», которые настолько элементарны, что дальнейшее их деление трудно представить. Каждая такая операция заключается в некотором изменении физической системы, состоящей из вычислителя и его ленты. Мы знаем состояние системы, если знаем последовательность символов на ленте, которые обозреваются вычислителем (возможно, в определенном порядке), и состояние памяти вычислителя. Мы можем предположить, что в простой операции изменяется не более одного символа. Любые другие изменения могут быть разбиты на простые изменения такого типа. Ситуация относительно клеток, символы которых могут быть изменены таким образом, – та же, что относительно обозреваемых клеток. Следовательно, без потери общности мы можем считать, что клетки, символы которых изменяются, – это только «обозреваемые» клетки.

Помимо изменений символов, простые операции должны включать изменения расположения обозреваемых клеток. Новые обозреваемые клетки должны немедленно распознаваться вычислителем. Думаю, разумно предположить, что ими могут быть только те клетки,

[1] Kurt Gödel, *Collected Works, Volume II: Publications 1938–1974* (Oxford University Press, 1990), 306. Введение Джадсона К. Вебба (Judson C. Webb), начинающееся на странице 292, – и, в частности, выявление на странице 297 веры Гёделя, что у человеческого разума есть существование, отдельное от физической материи мозга, – полезно в понимании гёделевских замечаний. Другой анализ – Орон Шагир (Oron Shagrir), «Gödel on Turing on Computability», http://edelstein.huji.ac.il/staff/shagrir/papers/Goedel on Turing on Computability.pdf.

чье расстояние от ближайшей из обозреваемых непосредственно до этого клеток не превышает определенной фиксированной величины. Давайте считать, что каждая из новых обозреваемых клеток находится в пределах L клеток, обозревавшихся непосредственно до этого.

В связи с «немедленным распознаванием» можно считать, что существуют и другие виды клеток, которые немедленно распознаются. В частности, клетки, помеченные специальными символами, [251]
могли бы считаться немедленно распознаваемыми. Теперь, если эти клетки помечаются только одиночными символами, которых может быть лишь конечное число, то мы не должны нарушить нашу теорию, присоединив эти помеченные клетки к обозреваемым. С другой стороны, если они помечаются последовательностью символов, то мы не можем считать процесс распознавания простым. Этот момент важен и должен быть проиллюстрирован. В большинстве математических статей уравнения и теоремы нумеруются. Обычно номера не выходят за пределы (скажем) 1000, поэтому теорему можно сразу распознать по ее номеру. Но если бы статья была очень длинной, мы могли бы дойти до теоремы 157767733443477; тогда далее в статье мы могли бы найти «...отсюда (по теореме 157767733443477) мы имеем...». Чтобы убедиться, что же это была за теорема, мы должны сравнить эти два номера цифра за цифрой, возможно помечая цифры карандашом, чтобы быть уверенными в том, что они не были учтены дважды. Если, несмотря на это, все же считать, что есть другие «немедленно распознаваемые» клетки, то и это не нарушает мое утверждение до тех пор, пока эти клетки могут быть обнаружены некоторым процессом, на который способны мои машины. Эта идея развивается ниже в пункте III.

Говоря «помечая цифры карандашом», Тьюринг, наверное, имеет в виду похожую машинную операцию «пометки» клеток нечисловыми значками.

Поэтому простые операции должны включать:

(a) Изменения символа в одной из обозреваемых клеток.

(b) Изменения одной из обозреваемых клеток на другую клетку в пределах L клеток от одной из обозревавшихся до этого клеток.

Может случиться так, что некоторые из этих изменений обязательно повлекут за собой изменение состояния памяти. Поэтому самая общая отдельная операция должна быть одной из следующих:

(*A*) Возможное изменение (*a*) символа вместе с возможным изменением состояния памяти.

(*B*) Возможное изменение (*b*) обозреваемых клеток вместе с возможным изменением состояния памяти.

Выполняемая операция фактически определяется, как предполагалось на с. 250, состоянием памяти вычислителя и обозреваемыми символами. В частности, она определяет состояние памяти вычислителя после выполнения операции.

Страница, на которую он ссылается, – это как раз предыдущая страница его статьи.

Теперь мы можем создать машину, выполняющую работу этого вычислителя. Каждому состоянию памяти вычислителя соответствует «*m*-конфигурация» машины. Машина просматривает *B* клеток, соответствующих *B* клеткам, обозреваемым вычислителем. При любом перемещении машина может изменить символ в просматриваемой клетке или может изменить любую из просматриваемых клеток на другую клетку, отстоящую не более чем на *L* клеток от любой из про-
[252]
сматриваемых клеток. Выполняемое перемещение и последующая конфигурация определяются просматриваемым символом и *m*-конфигурацией. Только что описанные машины, в сущности, почти не отличаются от вычислительных машин, определенных в §2, и для любой машины этого типа можно создать вычислительную машину, вычисляющую ту же самую последовательность, то есть последовательность, вычисляемую вычислителем.

То есть *человеком*-вычислителем.

На этом мы пока остановимся. Второй аргумент Тьюринга в этом разделе начинается с упоминания «ограниченного гильбертова функционального исчисления», за которым следует утверждение в этом исчислении, а для этого нужна некоторая подготовка, с которой и начинается часть III этой книги.

Увлечение Тьюринга связью человеческого мозга и машин продолжалось и после его статьи 1936 года о вычислимых числах. Другая

известная работа Тьюринга «Вычислительные машины и интеллект» была опубликована в октябрьском 1950 года выпуске философского журнала *Mind*.

«Могут ли машины мыслить?» – спрашивает Тьюринг. Теперь он придумывает тест с человеком, сидящим за телетайпом. (Современным эквивалентом ему может быть средство передачи сообщений или что-то еще, что не позволяет людям видеть или слышать того, с кем они общаются.) Человеку позволено задавать вопросы и получать ответы. Если на другом конце – действительно компьютер и человек не может сказать, что это – компьютер, мы вынуждены признать, что компьютер обладает разумом.

Этот тест получил известность как тест Тьюринга, и он до сих пор остается спорным. Все, кто решительно против теста Тьюринга, должны прочитать его статью, в которой уже есть ответы на многие разумные возражения.

В этом вопросе Тьюринг предпочитает иметь дело скорее с «интеллектом», чем с «мышлением», поскольку «мышление» подразумевает определенную деятельность, происходящую *внутри* компьютера.

> Я полагаю, что первоначальный вопрос «Могут ли машины мыслить?» слишком бессмысленный, чтобы заслуживать обсуждения. Тем не менее я считаю, что в конце столетия употребление слов и всеобщее просвещенное мнение изменятся настолько, что каждый будет в состоянии говорить о думающих машинах, не встречая возражений[1].

Конец столетия минул, но наоборот, еще больше, чем когда-либо, людей знают: что бы ни делал компьютер, он не «мыслит». Ожидания того, что наши компьютеры станут интеллектуальными, не оправдались, и в большинстве случаев с нашими компьютерными приложениями нам работается лучше, когда мы уверены, что они действуют совершенно детерминированным образом. Компьютерная программа, которая пытается сделать что-то «умное», часто напоминает двухлетнего малыша, сияющего восторгом от только что разрисованной стены и вопрошающего «А я думал, что тебе это *понравится*».

А вот в области научной фантастики прогноз Тьюринга попал прямо в точку, о чем свидетельствует самый известный вымышленный компьютер всех времен:

> Вопрос о том, может ли действительно Эал думать, уже был обоснован британским математиком Аланом Тьюрингом в далеких 1940-х годах.

---

[1] Alan Turing, «Computing Machinery and Intelligence», *Mind*, Vol. LIX, No. 236 (October 1950), 442.

> Тьюринг показал, что если вести долгий разговор с машиной – не важно, с помощью пишущей машинки или микрофона, – так и не сумев отличить ее ответы от тех, что мог бы дать человек, то машина была бы думающей в некотором разумном смысле слова. Эал мог бы с легкостью пройти тест Тьюринга[1].

В 1968 году, когда вышли книга и фильм «2001», Алану Тьюрингу было бы 56 лет. Наверное, он порадовался бы идее настолько интеллектуального компьютера, который способен испытать даже нервный срыв.

Летом 1950 года Тьюринг переехал в дом в Уилмслоу (Wilmslow) примерно в десяти милях к югу от Манчестера. Он заинтересовался морфогенезом, который изучает, как развиваются и различаются клетки в организме, демонстрируя разнообразные структуры и формы. Исследование включало в себя моделирование на манчестерском компьютере.

15 марта 1951 года Алан Тьюринг в знак признания его работы по вычислимым числам был избран членом Королевского общества. Его поручителями были Макс Ньюмэн и Бертран Рассел. В тот вечер Би-Би-Си передала по радио запись беседы с Тьюрингом под названием «Могут ли вычислительные машины мыслить?» (Ни записи той радиопередачи, ни какой-либо другой записи голоса Тьюринга, как известно, не существует.)

В декабре 1951 года была запущена цепь событий, которые будут иметь серьезные последствия. На улицах Манчестера Тьюринг встретил молодого человека. Арнольд Мюррэй происходил из рабочих, был условно осужден за воровство и был безработным. Тьюринг и Мюррэй пообедали, снова встретились и вместе вернулись в дом Тьюринга. На протяжении следующего месяца они встречались несколько раз.

В конце января 1952 года Тьюринг обнаружил, что его дом ограблен. Он сообщил об этом в полицию, которая приехала и сняла отпечатки пальцев. При встрече с Тьюрингом Арнольд Мюррэй заявил о своей невиновности, но сказал, что знает, кто это сделал, – знакомый по имени Гарри. Полиция тоже опознала Гарри по отпечаткам пальцев, снятым в доме Тьюринга. Гарри уже был под стражей за что-то еще. На допросе об ограблении Тьюринга Гарри передал полиции сплетню о том, что происходило между Тьюрингом и его другом.

---

[1] Arthur C. Clark, *2001: A Space Odyssey* (New American Library, 1968), ch. 16.

7 февраля 1952 года, в день после смерти Георга VI и восшествия на престол самой старшей его дочери Елизаветы, Алан Тьюринг был вызван в полицию. После непродолжительного допроса Тьюринг признался в характере своих отношений с Мюррэем. За это признание Тьюринг подлежал аресту по статье 11 Поправок к Уголовному закону (Criminal Law Amendment Act) от 1885 года:

> Любое лицо мужского пола, которое прилюдно или наедине совершает или соучаствует в совершении, организует или пытается организовать с любым лицом мужского пола любой акт грубой непристойности с другим лицом мужского пола, виновно в проступке и, будучи признанным виновным в нем, по усмотрению суда подлежит заключению в тюрьму на срок, не превышающий два года, с привлечением к тяжелым работам или без оных[1].

Термин «грубая непристойность» не был определен в законе, но означал обычно такие действия, как взаимная мастурбация и оральный секс. Другие положения касались более серьезных преступлений анального секса (или «мужеложства», как это называлось в британском праве).

Печально известная статья 11 была спорной с самого ее появления. Поправка к Уголовному закону 1885 года звучит как «Закон о дополнительных мерах по обеспечению защиты женщин и девочек, пресечению борделей и других намерений». Закон повышал возраст совершеннолетия с 13 до 16 лет и содержал несколько мер, направленных на пресечение таких форм эксплуатации женщин, как применение наркотиков в борделях или принуждение к проституции.

Сначала Закон пару лет болтался в палате общин, пока не стал срочным после ряда статей либерального журналиста Уильяма Томаса Стида (1849–1912) о детской проституции. Смелые разоблачения Стида достигли кульминации в связи с делом о фактической покупке им 13-летней девочки у ее родителей.

После общественной шумихи вокруг статей Стида Закон был принят. Статья 11 была представлена членом парламента Анри Лабушером 6 августа 1885 года и дополнила Закон на следующий день, всего за неделю до окончательного его принятия. В то время существовало некоторое сомнение, следовало ли дополнять этой статьей проект Закона, в центре которого была защита девочек и женщин[2].

---

[1] Формулировка получена из http://www.swarb.co.uk/acts/1885Criminal LawAmendmentAct.shtml (доступ в апреле 2008 г.).

[2] H. Montgomery Hyde, *The Love That Dared Not Speak Its Name: A Candid History of Homosexuality in Britain* (Little, Brown and Company, 1970), 134. Эта книга была первоначально издана в Англии под заголовком «*The Other Love*».

Статья 11 касалась исключительно мужчин, и действия, скрывающиеся за словами «грубая непристойность», никогда раньше в Великобритании не были незаконными, во всяком случае если они совершались взрослыми наедине по обоюдному согласию. В то же время оговорка «наедине», похоже, позволяла использовать закон для шантажа[1].

Самой известной жертвой статьи 11 был Оскар Уайльд (1854–1900), который преследовался по суду согласно Закону от 1895 года. Уайльд отбывал свой срок, выполняя тяжелую работу, которая, вероятно, ускорила его смерть.

Но в 1950-х годах были возможны и другие методы наказания. Признав свою вину, Тьюринг согласился на годовой испытательный срок, в течение которого его обязали получать гормональное лечение.

Опыты по лечению гомосексуализма с применением половых гормонов начались в 1940-е годы. Сначала гомосексуализм считался следствием недостатка мужских гормонов, но назначение тестостерона приводило на деле к обратному результату. Тогда на гомосексуальных мужчинах были опробованы женские половые гормоны, и те, похоже, дали более желательный эффект[2]. Ко времени вынесения приговора Тьюрингу это лечение было известно как органотерапия, но называлось еще и «химической кастрацией» и, казалось, предполагало скорее унижение, чем что-то еще. Эстроген сделал Тьюринга импотентом и вызвал рост его груди.

Начало 1950-х было неблагоприятным временем для уличенных в гомосексуализме. В Соединенных Штатах «красная угроза» начала 1950-х годов вскоре обернулась охотой на ведьм иного рода. Коммунистов, работавших в Госдепартаменте, было на самом деле не так много, но было много скрытых геев, занимавших государственные должности в Вашингтоне округа Колумбия. «В 1950-х и 1960-х годах из Госдепартамента по причине гомосексуальности было уволено около 1000 человек»[3].

Теоретически слова «угроза безопасности» (неблагонадежность) могли быть применены к любому, кто имеет склонность к разглашению государственных тайн. На деле ими, в сущности, подменяли

---

[1] H. Montgomery Hyde, *The Love That Dared Not Speak Its Name: A Candid History of Homosexuality in Britain* (Little, Brown and Company, 1970), 136.

[2] Hodges, *Alan Turing*, 467–471.

[3] David K. Johnson, *The Lavender Scare: The Cold War Persecution of Gays and Lesbians in the Federal Government* (University of Chicago Press, 2004), 76.

слово «гомосексуальность»[1]. Сложилось предвзятое мнение, что гомосексуалов можно путем шантажа склонить к разглашению государственных тайн. Лучшая тому иллюстрация из всего, что можно придумать, – это то, что действительно случилось с главой австрийской разведки накануне Первой мировой войны, однако было не вполне ясно, что же произошло на самом деле[2].

То, что происходило с геями в правительстве Соединенных Штатов, имело значение и для Великобритании. В 1951 году Государственный департамент США начал консультировать британское Министерство иностранных дел по «гомосексуальной проблеме» в правительстве, а позже настоятельно требовал от британского правительства быть более усердным в отношении гомосексуалов в связи с возможными проблемами безопасности[3].

Возможности найти работу у Алана Тьюринга, безусловно, становились более ограниченными. Сверхсекретная правительственная работа, которая была у Тьюринга во время войны, теперь была немыслима, и при этом у Тьюринга не было возможности снова поехать в Америку. Закон 1952 года запрещал въезд «иностранцам, страдающим психическими расстройствами личности», под которой понималась и гомосексуальность[4].

Было ли этого достаточно, чтобы довести Тьюринга до самоубийства? Мы не знаем.

На улицах Англии, как и в правительстве, жизнь геев становилась все труднее. Когда сэр Джон Нотт-Бауэр в 1953 году был назначен комиссаром Лондонской столичной полиции, он поклялся, что «сорвёт покров со всех грязных пятен Лондона». Тогда же министерство внутренних дел выпустило указания по новому наступлению на «мужскую безнравственность». Так, один лондонский судья, устав нянчиться с людьми, требовал считать их виновными и «отправлять за решетку, коль они бывали там раньше». В конце 1953 и в начале 1954 года заголовки газет пестрели судебными делами на разных людей[5].

Были ли у Тьюринга новые отношения с человеком, который теперь угрожал ему шантажом?

---

1 Johnson, *Lavender Scare*, 7–8.
2 Johnson, *Lavender Scare*, 108–109.
3 Johnson, *Lavender Scare*, 133.
4 Hodges, *Alan Turing*, 474.
5 Hyde, *The Love That Dared Not Speak Its Name*, 214-6.

Или самоубийство Тьюринга было просто нелепой случайностью, как полагала его мать?

Показателем нашей неосведомленности является то, что душевное состояние Тьюринга в тот период убедительнее всего раскрывает не история или биография, а роман. Романист Жанна Левин (она же профессор астрономии и физики в Колледже Барнарда) изображает человека униженного и не способного выразить это:

> Он не знает, как передать свое унижение или даже вынести его. Оно скрежещет в нем, как сломанная деталь, выбитая и болтающаяся в своем металлическом кожухе. Унижение не обживется в одном месте, где, осев, будет, вне всякого сомнения, гноиться, но хотя бы может быть изолировано и, возможно, даже вылечено. И если оно не разрушится постоянной незаметной шлифовкой волн времени, то, может быть, при более энергичном воздействии будет вырезано его юнговским психоаналитиком. И только позор невозможно ни закопать, ни замуровать[1].

Нам ничего не известно о том, что еще было вечером 7 июня 1954 года. Нам неизвестно, что заставило Тьюринга опустить свое неизменное вечернее яблоко в цианистый калий, перед тем как лечь спать.

Следующим утром он был найден мертвым. Алану Тьюрингу был 41 год.

---

[1] Janna Levin, *A Madman Dreams of Turing Machines* (Alfred A. Knopf, 2006), 214.

Часть III

Entscheidungsproblem

Глава 12

Логика и вычислимость

Летом 1958 года у логика Хао Вана, выходца из Китая, выдался перерыв в преподавании в Оксфорде, чтобы провести некоторое время с современным компьютером IBM 704 в Исследовательской лаборатории IBM в Покипси (Poughkeepsie), штат Нью-Йорк. Ван набивал на перфокартах теоремы непосредственно со страниц книги А. Н. Уайтхеда и Б. Рассела *Principia Mathematica*, вышедшей почти 50 лет назад. Например, теорема *11.26 в системе обозначений *Principia Mathematica* выглядела так:

***11 · 26**. |–: .(∃*x*): (*y*). ϕ (*x*, *y*): ⊃: (*y*): (∃*x*). ϕ (*x*, *y*).

А на перфокарте в системе обозначений Вана она стала такой:

```
11*26/EXAYGXY-AYEXGXY.
```

Ван написал три программы для чтения этих перфокарт и доказал закодированные теоремы, применяя различные тождества и правила вывода для обращения высказываний в аксиомы. Бóльшую часть затраченного этими программами времени занимали механические процессы чтения карт и печати шагов доказательства. По оценкам Ванна, фактическое время доказательства 220 теорем из глав с *1 по *5 из *Principia Mathematica* составило меньше 3 минут, а улучшенная версия его третьей программы позже смогла доказать 158 теорем из глав с *9 по *13 примерно за 4 минуты[1].

Программа Вана была не первой попыткой доказательства теорем с помощью компьютера[2].

---

[1] Hao Wang, «Toward Mechanical Mathematics», *IBM Journal of Research and Development*, Vol. 4, No. 1 (Jan. 1960), 2–22. Доступна на http://www.research.ibm.com/journal/rd/041/ibmrd0401B.pdf.

[2] Donald MacKenzie, «The Automation of Proof: A Historical and Sociological Exploration», *Annals of theHistory of Computing*, Vol. 17, No. 3 (Fall 1995), 7–29.

В 1954 году Мартин Дэвис воспользовался компьютером, созданным в Институте перспективных исследований (Institute for Advanced Study) в Принстоне, чтобы запрограммировать процедуру Пресбургера для простой арифметики с одним лишь сложением. В первом известном компьютерном математическом доказательстве программа Дэвиса доказала, что сумма двух четных чисел – также четное число[1].

В 1957 году Аллен Ньюэлл, Дж. К. «Клифф» Шоу и Герберт Саймон опубликовали результаты программы «Логик-теоретик» или «Логическая машина» («Logic Theory Machine»), созданной для вычислительной машины JOHNNIAC корпорации RAND, названной так в честь Джона фон Неймана[2]. Ньюэлл, Шоу и Саймон использовали в качестве источника теорем ту же *Principia Mathematica*. Интересуясь больше искусственным интеллектом, чем математической логикой, они написали свою программу так, чтобы она при доказательстве теорем подражала человеку (они назвали это «эвристическим» подходом). Позже Хао Ван нашел лучший алгоритм с большей производительностью и скоростью достижения успеха.

Узнав из письма о результатах «Логической машины», Бертран Рассел дал примерно такой ответ: «Я рад узнать, что сделанное в *Principia Mathematica* теперь может быть сделано машиной. Мне жаль, что Уайтхед и я не знали об этой возможности раньше и потратили впустую 10 лет, занимаясь этим вручную»[3].

Три тома и почти 2000 страниц *Principia Mathematica* были монументальным достижением в математике и логике. В опубликованный Библиотекой современности список 100 лучших научных работ XX века *Principia Mathematica* вошла под номером 23[4]. Однако очень немногие были готовы к такой самоотверженности. Стивен Клини, студент Алонзо Чёрча, который позже написал знаковые книги *Введение в метаматематику* (1952) и *Математическая логика*

---

[1] Martin Davis, «A Computer Program for Presburger's Algorithm», в Jörg Seikmann and Graham Wrightson, eds., *Automation of Reasoning 1: Classical Papers on Computational Logic, 1957–1966* (Springer-Verlag, 1983), 41–48.

[2] Allen Newell, J. C. Shaw and H. A. Simon, «Empirical Explorations with the Logic Theory Machine: A Case Study in Heuristics», *Proceedings of the Western Joint Computer Conference*, Vol. 15 (1957), 218–239. Reprinted in Edward A. Feigenbaum and Julian Feldman, eds., *Computers and Thought* (MIT Press, 1995), 109–133.

[3] Quoted in Michael J. Beeson, «The Mechanization of Mathematics», in Christof Teuscher, ed., *Alan Turing:* Life and Legacy of a Great Thinker (Springer, 2004), 93.

[4] http://www.randomhouse.com/modernlibrary/100bestnonfiction.html.

(1967), без стеснения признавался, что он никогда не читал *Principia Mathematica*[1]. И среди тех, кто после 1913 года оказывал большое влияние на математическую логику, он находится, наверное, в подавляющем большинстве.

Введение к *Principia Mathematica* ставит цель, никак не меньшую, чем «полное перечисление всех идей и этапов рассуждений, применяемых в математике». Это – программа (и философия математики), известная как *логицизм* – использование логики в качестве фундамента всей остальной математики. Чтобы совершить этот подвиг, Уайтхед и Рассел применили весь арсенал теоретико-множественных и логических инструментальных средств. Идея намеренного ограничения математических методов казалась им абсурдной.

Но не для Давида Гильберта, человека, тесно связанного с философией математики, называемой *формализмом*. Формализм фокусируется на аксиоматике, и особенно в программе Гильберта, делавшей упор на такие понятия, как непротиворечивость, разумность, полнота и разрешимость.

Отчасти в педагогических и аналитических целях Давид Гильберт разбил логику *Principia Mathematica* на расширяющиеся подмножества, каждое из которых можно изучать независимо. Такой подход был основным в курсе, который он преподавал в Гёттингене зимой 1917–1918 годах. В 1928 году он стал 120-страничной книгой *Grundzüge der Theoretischen Logik* (*Основы математической логики*) Давида Гильберта и Вильгельма Аккермана, известной всем как Hilbert & Ackermann. Эта книга – первоисточник Entscheidungsproblem, которая является центральной в статье Тьюринга.

Статья Тьюринга явно ссылается на *Grundzüge der Theoretischen Logik*, как и на более позднюю книгу Давида Гильберта и Пауля Бернайса *Grundlagen der Mathematik* (*Основания математики*), первый том которой вышел Берлине в 1934 году и которая известна как Hilbert & Bernays. (Второй том появился в 1939 году после публикации статьи Тьюринга.)

Следующая часть статьи Тьюринга требует некоторого знакомства с математической логикой в изложении Hilbert & Ackermann. В последующем кратком обзоре этой логики я буду придерживать-

---

[1] William Aspray, The Princeton Mathematics Community in the 1930s: An Oral-History Project. An interview with J. Barkley Rosser and Stephen C. Kleene in Madison, Wisconsin, on 26 April 1984, http://www.princeton.edu/~mudd/finding aids/mathoral/pmc23.htm.

ся системы обозначений Тьюринга, которая *очень* похожа на систему обозначений, применяемую в Hilbert & Ackermann. Я буду также придерживаться подхода к описанию этой логики как расширения подмножества возможностей; это уже стало стандартом и присутствует в учебниках по математической логике Алонзо Чёрча, Стивена Клини, Эллиота Мендельсона, Герберта Б. Эндертона и многих других.

Начну я с того, что Hilbert & Ackermann называет *Aussagenkalkül*, переведенное позже как *исчисление высказываний*, но более известное сегодня как *пропозициональное исчисление* или *пропозициональная логика*.

Затем я расширю логику до того, что первоначально называлось в Hilbert & Ackermann *engere Funktionenkalkül*, или *ограниченное функциональное исчисление*. Во втором издании книги 1938 года оно было переименовано в *engere Prädikatenkalkül*, или узкое исчисление предикатов, но более известное сегодня как логика *первого порядка*, или логика предикатов первого порядка, или исчисление предикатов первого порядка. По мере появления новых понятий я буду различать логику первого порядка и логику второго порядка, которая в Hilbert & Ackermann называется *erweiterte Funktionenkalkül*, а позже – *erweiterte Prädikatenkalkül*, или *расширенное исчисление предикатов*.

Пропозициональная логика (логика высказываний) имеет дело со всеми декларативными утверждениями (или высказываниями), которые имеют *истинностное значение* – то есть их можно считать либо истинными, либо ложными. Например:

Сегодня среда.
Семь – простое число.
Идет дождь.
Мою мать зовут Барбара.
Десять – полный квадрат.

Некоторые из этих высказываний истинны, некоторые ложны, а некоторые могут быть истинными для меня, но ложными для вас. (Без драки!) В пропозициональной логике высказывания имеют единственное непротиворечивое, недвусмысленное истинное значение, и чем меньше мы делаем вид, что успешно разбираем что-то, кроме математических высказываний, тем меньше мы будем путаться.

В пропозициональной логике высказывания часто представляются наклонными прописными буквами. Первыми буквами алфавита (*A*, *B* и *C*) часто обозначаются высказывания с фиксированным зна-

чением истинности, тогда как последние буквы алфавита (*X, Y* и *Z*) используются для высказываний-переменных.

Мы можем соединять отдельные высказывания с помощью определенных связок (соединителей), чтобы получать еще большие составные высказывания.

Первой из этих связок является строчная буква v, от латинского слова *vel* со значением *или*, а точнее *включающее или* как противоположность латинскому *aut*, *исключающему или*. Высказывание

X v *Y*

истинно, если истинно либо *X*, либо *Y*, либо они оба. Для представления возможных комбинаций *X* и *Y* полезна небольшая таблица истинности:

X	**Y**	**X v Y**
false	false	false
false	true	true
true	false	true
true	true	true

Разрешается опускать символ v, если это не приводит к путанице. Формула

XY

равносильна формуле:

X v *Y*.

Заметьте, что я не записываю равносильность этих двух формул одной строкой, связав их знаком равенства. Знак равенства не включен в язык исчисления высказываний – или, по крайней мере, в исчисление высказываний из книги Hilbert & Ackermann.

Когда мы говорим, что одно высказывание *равносильно* другому, мы подразумеваем, что они имеют одни и те же значения истинности для соответствующих значений истинности составляющих их высказываний. Мы выражаем эту равносильность на естественном языке, также известном как *метаязык*. Строгое разделение языка логики и метаязыка необходимо, чтобы избежать путаницы.

Для обозначения равносильности Hilbert & Ackermann допускают метаязыковое сокращение aq. (от нем. äquivalent) или eq. (от англ. equivalent):

X v Y eq. XY.

Запомните, что это сокращение *не* является частью языка пропозициональной логики и необходимо исключительно для удобства.

Понятие «и» представляется знаком амперсанда &. Формула

X & Y

истинна тогда и только тогда, когда истинны и X, и Y, как показано в таблице истинности:

X	Y	X & Y
false	false	false
false	true	false
true	false	false
true	true	true

Операция «и» часто называется *конъюнкцией* от известного из грамматики понятия «союз» (conjunction – союз; *прим. перев.*); в свою очередь, операция «или» часто называется менее знакомым словом *дизъюнкция*.

Из таблиц истинности очевидно, что:

X v X eq. X
X & X eq. X

Если в составном высказывании используются обе связки, то v выполняется раньше &. Если же нужен иной порядок выполнения операций, его можно изменить правильной расстановкой круглых скобок или использовать их всегда для большей строгости.

X & Y v Z eq. X & (Y v Z).

Эти высказывания не равносильны:

(X & Y) v Z.

Например, если X ложно, Y истинно и Z истинно, то первые два высказывания ложны, а последнее истинно.

Не буду забивать вам голову различными правилами, которые обеспечивают строгую парность круглых скобок и появление логических связок только в нужных местах. Эти правила связаны с понятием правильной формулы (well-formed formula) или wff (произносится как «woof»). Слова «истина» и «ложь» не входят в словарь пропозициональной логики так же, как и символы «T» и «F», но для удобства их

можно использовать вместо пропозициональных символов. Можно считать, что T – это всегда истинное высказывание, а F – всегда ложное высказывание. Впредь в таблицах истинности я буду использовать T и F.

Следующие тождества очевидны из таблиц истинности:

X v T eq. T
X v F eq. X
X & T eq. X
X & F eq. F

Из таблиц истинности видно, что обе операции коммутативны:

X v Y eq. Y v X
X & Y eq. Y & X

Обе операции также ассоциативны:

X v (Y v Z) eq. (X v Y) v Z
X & (Y & Z) eq. (X & Y) & Z

Обе операции дистрибутивны по отношению друг к другу:

X v (Y & Z) eq. (X v Y) & (X v Z)
X & (Y v Z) eq. (X & Y) v (X & Z)

Если в таблицах истинности заменить F на 0, а T на 1, то можно видеть, что конъюнкция в точности равносильна умножению двух однозначных двоичных чисел, а дизъюнкция чем-то напоминает сложение. Поэтому конъюнкцию иногда называют «логическим умножением», а дизъюнкцию – «логическим сложением». Однако в использовании этих терминов есть некоторая натянутость, поэтому они не поощряются.

Конъюнкция и дизъюнкция – это бинарные операции. Единственная унарная (одноместная) операция называется «не», или «отрицание», и обозначается черточкой, очень похожей на знак минус:

X	–X
F	T
T	F

Обозначение отрицания у Тьюринга отличается от Hilbert & Ackermann тем, что он использует черту над буквой (надчеркивание) или более сложное выражение. Отрицание всегда вычисляется первым:

знак отрицания применяется только к идущему вслед за ним символу. Двойное отрицание взаимоуничтожается:

$$--X \text{ eq. } X.$$

Эти два отношения – самые простые:

$$X \text{ v } -X \text{ eq. T}$$
$$X \text{ \& } -X \text{ eq. F.}$$

Два самых основных – но и самых интересных – логических отношения объединяют дизъюнкцию, конъюнкцию и отрицание. Их называют законами Де Моргана в честь математика XIX века Огастеса Де Моргана (1806–1871), хотя это основополагающее понятие было известно еще Аристотелю:

$$-(X \text{ v } Y) \text{ eq. } -X \text{ \& } -Y$$
$$-(X \text{ \& } Y) \text{ eq. } -X \text{ v } -Y.$$

Эти тождества понятны в обычной речи. Например, «нет дождя или снега» или $-(X \text{ v } Y)$ – это то же самое, что «нет дождя *и* нет снега» или $-X \text{ \& } -Y$. Когда мне говорят «Увы, ты явно *не* богат и красив» или $-(X \text{ \& } Y)$, я могу лишь сделать вывод «надо полагать, я или беден, или уродлив... или все это вместе» или $-X \text{ v } -Y$.

Заметьте, что законы Де Моргана можно записать так, чтобы все знаки отрицания были сгруппированы в одной части тождества:

$$X \text{ v } Y \text{ eq. } -(-X \text{ \& } -Y)$$
$$X \text{ \& } Y \text{ eq. } -(-X \text{ v } -Y).$$

В таблицах истинности для операций v и & можно заменить все F на T и все T на F, и получится таблица истинности для противоположной операции. Это называется «принципом двойственности», который применим также и к составным высказываниям. Вот одно из них:

$$X \text{ \& } -Y \text{ v } Z.$$

Применим отрицание к каждой переменной и поменяем местами v и & (не забыв вставить скобки, чтобы не исказить смысл высказывания) – и получим отрицание первоначального высказывания:

$$-X \text{ v } (Y \text{ \& } -Z).$$

Если нужно проверить, что эти два высказывания действительно отрицают друг друга, можно создать небольшую таблицу истинности для проверки всех значений:

X	Y	Z	X & –Y v Z	–X v (Y & –Z)
F	F	F	F	T
F	F	T	F	T
F	T	F	F	T
F	T	T	F	T
T	F	F	T	F
T	F	T	T	F
T	T	F	F	T
T	T	T	T	F

Последние два столбца содержат противоположные значения истинности, значит:

$X \& -Y \text{ v } Z$ eq. $-(-X \text{ v } (Y \& -Z))$.

У операции *исключающего* «или» нет собственного обозначения, но она вычисляется по следующей формуле:

$(X \text{ v } Y) \& -(X \& Y)$.

Вот таблица истинности для этой операции:

X	Y	X v Y	X & Y	(X v Y) & – (X & Y)
F	F	F	F	F
F	T	T	F	T
T	F	T	F	T
T	T	T	T	F

Исключающее «или» очень похоже на обычную дизъюнкцию, за исключением случая, когда и X, и *Y* истинны.

Если мы применим законы Де Моргана ко второй половине высказывания для исключающего или, то получим:

$(X \text{ v } Y) \& (-X \text{ v } -Y)$.

Симметрия такого вида гораздо более симпатична.

В компьютерах исключающее «или» используется для вычисления суммы двух двоичных цифр, а конъюнкция – для бита переноса[1].

Третья бинарная операция – сложная. Она называется «импликация». $X \rightarrow Y$ читается как «*X* влечет *Y*» или «если *X*, то *Y*». Следует

[1] Charles Petzold, *Code: The Hidden Language of Computer Hardware and Software* (Microsoft Press, 1999), гл. 12.

предостеречь, что многие, впервые столкнувшись с ее таблицей истинности, недоумевают: «Здесь что-то не так».

X	**Y**	**X → Y**
F	F	T
F	T	T
T	F	F
T	T	T

Первые две строки таблицы могут показаться странными. Если X ложно, то почему $X \rightarrow Y$ должно быть истинно независимо от значения Y? Давайте посмотрим на это с другой стороны, начав с предположения, что $X \rightarrow Y$ истинно. Если $X \rightarrow Y$ истинно и X истинно, то Y должно быть истинно. Однако если X не истинно, то что можно сказать о Y? Ничего. Y может быть чем угодно. Именно поэтому $X \rightarrow Y$ может быть истинным, если X ложно независимо от Y.

Рассмотрим высказывание «Если идет дождь, то есть осадки». Это высказывание истинно, если идет дождь. Но оно также истинно, если дождь не идет (скажем, идет снег или град – *прим. перев.*). Единственный случай, когда высказывание ложно, – это когда идет дождь, но осадков нет.

Импликация очень часто используется в математической логике. Довольно нередко слева от знака импликации – формула, о которой мы знаем, что она истинна. Тогда если мы можем показать, что само высказывание истинно, то мы можем заключить, что формула справа истинна.

Импликация не коммутативна и не ассоциативна. Однако

$$X \rightarrow Y \quad \text{eq.} \quad -Y \rightarrow -X.$$

Второе высказывание называется *контрапозицией* (*contrapositive*). Если идет дождь, я беру зонт. У меня нет зонта, значит, дождя не должно быть. У импликации есть очень простая связь с дизъюнкцией:

$$X \rightarrow Y \quad \text{eq.} \quad -X \text{ v } Y.$$

Иными словами, $X \rightarrow Y$ истинно, если либо X ложно, либо Y истинно. Импликацию можно также выразить через конъюнкцию:

$$X \rightarrow Y \quad \text{eq.} \quad -(X \,\&\, -Y).$$

Если слева от знака импликации – конъюнкция произвольного числа членов, то любой из этих членов может быть справа от знака импликации:

$X \& Y \rightarrow X$
$X \& Y \rightarrow Y$

Hilbert & Ackermann описывает двунаправленное условие (или «тогда и только тогда»), обозначаемое тильдой:

X	Y	X ~ Y
F	F	T
F	T	F
T	F	F
T	T	T

$X \sim Y$ истинно только тогда, когда X и Y имеют одинаковые значения истинности. Тьюринг в своей статье вообще не использует двунаправленное условие. Тем не менее стоит заметить, что оно равносильно конъюнкции импликаций противоположных направлений:

$X \sim Y$ eq. $(X \rightarrow Y) \& (Y \rightarrow X)$.

Если допустить, что это тождество имеет смысл, то оно верно только тогда, когда T → T истинно (что несомненно) и F → F тоже истинно (что сомнительно). Кроме того, T → F и F → T должны иметь противоположные значения истинности. Несомненно, что T → F должно быть ложно, что означает, что F → T должно быть истинно. Это подтверждает правильность таблицы истинности для импликации.

Допустим, дано следующее высказывание:

$X \vee Y \vee (-X \& -Y)$.

Что вы можете о нем сказать? Истинно оно или ложно? Вы можете возразить и заявить, что этого сказать нельзя, покуда неизвестны значения X и Y. Тогда вам предлагается составить таблицу истинности, дабы проверить все комбинации X и Y. Вот она:

X	Y	X v Y v (–X & –Y)
F	F	T
F	T	T
T	F	T
T	T	T

Независимо от конкретных значений X и Y это высказывание всегда истинно. Мы говорим, что такое высказывание – *тавтология*, или что оно *всюду справедливо*. Всюду справедливые высказывания очень

любимы в математической логике, потому что они истинны независимо от значений истинности их составляющих.

Давайте рассмотрим другой случай:

X	**Y**	**X & Y & –X**
F	F	F
F	T	F
T	F	F
T	T	F

Это высказывание всюду ложно. Это – *опровержение* (или *противоречие*). Отрицание тавтологии – это противоречие, а отрицание противоречия – тавтология.

И третий случай:

X	**Y**	**X v (Y&–Y)**
F	F	F
F	T	F
T	F	T
T	T	T

Это высказывание иногда истинно, а иногда ложно в зависимости от значений *X* и *Y*. Это высказывание, как говорят, *выполнимо*, то есть обладает способностью быть истинным при определенной комбинации логических значений.

Всюду справедливое высказывание (тавтология) считается также выполнимым. Высказывание всюду справедливо тогда и только тогда, когда отрицание этого высказывания невыполнимо.

Для любого высказывания мы можем использовать таблицу истинности, чтобы определить, является ли это высказывание всюду справедливым, опровержимым (противоречивым) или просто выполнимым. Процесс построения таблицы истинности – чисто механический. Он не требует никакого особого вдохновения, понимания или интуиции. Программист может легко представить себе программу, которая читает высказывание пропозициональной логики, проверяет его и печатает слова «справедливо», «противоречиво» или «выполнимо».

Поэтому мы говорим, что высказывания в исчислении высказываний *разрешимы*. В пропозициональной логике существует *разрешающая процедура* определения справедливости или выполнимости любого высказывания.

Иными словами, Entscheidungsproblem для исчисления высказываний решена. И здесь нам делать нечего.

Нельзя сказать, что таблица истинности всегда удобна. Предположим, что высказывание имеет 100 логических переменных. Число строк в такой таблице истинности – 2^{100}, или 1 267 650 600 228 229 401 496 703 205 376, или примерно 10^{30}, что является очень большим числом. Даже на воображаемом компьютере, который обрабатывает каждую строку за 1 наносекунду (одна миллиардная доля секунды), или время, за которое свет пролетает примерно один фут, – продолжительность обработки составила бы 38 триллионов лет, или примерно в 3000 раз больше возраста нашей вселенной.

А вот если ограничиться 55 логическими переменными и все еще иметь возможность обрабатывать каждую строку за 1 наносекунду, то придется ждать примерно год. Каждая новая переменная удваивает продолжительность обработки. В теории вычислимости и сложности время вычисления, уходящее на обработку таблицы истинности, называется *экспоненциальным временем*, потому что растет экспоненциально с ростом размера задачи.

По этим причинам при обработке высказываний в пропозициональной логике решения, отличные от таблиц истинности, имеют большее значение. Такие методы часто приводят высказывание к *нормальной форме* – либо к конъюнкции множества членов, каждый из которых есть дизъюнкция переменных, либо к дизъюнкции множества членов, каждый из которых – конъюнкция переменных.

На этом мой довольно скорый обзор пропозициональной логики завершается. Нужно идти дальше, потому что логики высказываний недостаточно для многих других целей. Главная проблема логики высказываний – в том, что мы имеем дело всего лишь с декларативными высказываниями и не можем связать их содержание друг с другом. Логика высказываний терпит неудачу при попытке разобрать силлогизмы (умозаключения) Аристотеля («Все люди смертны; Сократ – человек; следовательно...») или откровенные «выкрутасы» (sorites), придуманные Льюисом Кэрроллом («Ни один котенок, любящий рыбу, не приручаем; Ни один котенок без хвоста не может играть с гориллой; Котята с усами всегда любят рыбу; Нет приручаемых котят с зелеными глазами; Нет котят с хвостами, если у них нет усов; следовательно...»[1]).

---

[1] Lewis Carroll, *Symbolic Logic: Part I. Elementary* (Macmillan, 1896), 118. Решение: «Котята с зелеными глазами не будут играть с гориллой». Но вам же это было известно!

Можно сделать логику более мощной, введя в нее *логические функции* или *предикаты*. (Тьюринг предпочел первый термин; второй – более современный. Я использую оба в равной степени.) Термин *предикат* пришел из грамматики, где предложения раскладываются на подлежащие и сказуемые. Например, в предложении «политик говорит великие истины» подлежащее – «политик», а сказуемое «говорит великие истины».

Введение предикатов – первый шаг в превращении логики высказываний в логику предикатов первого порядка. Всякий раз, когда мы используем предикаты, мы ограничиваемся определенной *областью определения*, или совокупностью. В действительности чаще всего эта область определения – натуральные числа. Представители этой совокупности – *аргументы* предикатов.

У Hilbert & Ackermann (и у Тьюринга) предикаты похожи на функции, но их значениями могут быть только истина и ложь. Мне нравится именовать предикаты словами или словосочетаниями вроде *IsPrime* (*ЕстьПростое – прим. перев.*). Область определения предиката *IsPrime* – натуральные числа, поэтому *IsPrime*(7) истинен, а *IsPrime*(9) ложен. Элементы области определения могут обозначаться строчными буквами, которые выступают в качестве переменных, например *IsPrime*(*x*).

Стоит в предикате указать явный аргумент, и он вместе с аргументом становится высказыванием. Например, *IsPrime*(10) – то же самое, что высказывание «Десять – простое число». Именно так логика предикатов соотносится с логикой высказываний.

Предикаты могут иметь много аргументов. Допустим, вы имеете дело с областью определения, состоящей из ваших друзей, каждый из которых имеет уникальное имя. Предикат *Любит*(*x*, *y*) истинен, если человек *x* любит человека *y*. Например, *Любит*(*Пэт*, *Терри*) – это то же самое, что высказывание «Пэт любит Терри». Аргументы некоторых предикатов могут быть коммутативными, но не в нашем случае: *Любит*(*Пэт*, *Терри*) – это *не* то же самое, что *Любит*(*Терри*, *Пэт*).

Мы можем соединять предикаты теми же самыми связками, которые используются в логике высказываний. Высказывание

Любит(*Пэт*, *Терри*) & *Любит*(*Терри*, *Пэт*)

истинно, если *Пэт* и *Терри* оба любят друг друга, и

Любит(*Пэт*, *Терри*) ~ *Любит*(*Терри*, *Пэт*)

истинно, если их чувства (какими бы они ни были) обоюдны.

А вот что означает

Любит(*x*, *Пэт*),

не вполне ясно. Если это вообще имеет хоть какой-то смысл, то можно предположить, что это значит, что *все* любят Пэт. Или, быть может, хоть *кто-то* любит Пэт.

Чтобы избежать подобной двусмысленности, нужно ввести еще два *квантора* (*quantifiers* или *quantors* у Тьюринга). Первый – *квантор общности*, который представляет собой переменную в круглых скобках перед предикатом:

(*x*)*Любит*(*x*, *Пэт*)

(*x*) означает «для всех *x*» (или «для всякого *x*», «для каждого *x*», «для любого x» – *прим. перев.*). Эта формула истинна, если «для всех *x* – *x* любит Пэт». Она истинна, если все любят Пэт. А формула

(*x*)*Любит*(*Пэт*, *x*)

истинна, если Пэт любит всех. Часто квантор общности обозначается символом ∀, но не у Hilbert & Ackermann или у Тьюринга.

Второй тип квантора – *квантор существования*, который переводится как «существует». Hilbert & Ackermann обозначают его обычной буквой E, а Тьюринг предпочитает общепринятый символ ∃. Например,

(∃*x*)*Любит*(*x*, *Терри*)

значит, что «существует *x*, такой, что *x* любит Терри», или кто-то (хотя бы один) любит Терри, даже если это – сам Терри.

В логике первого порядка (Hilbert & Ackermann называют ее *ограниченным* исчислением) кванторы применяются только к переменным, обозначающим элементы из области определения. В логике второго порядка (или *расширенном* исчислении) кванторы могут применяться к переменным, представляющим собой логические функции. В статье Тьюринга используется только логика первого порядка.

Если мы имеем дело с *конечной* совокупностью (областью определения), то квантор общности можно выразить конъюнкцией, а квантор существования – дизъюнкцией. Например, допустим, что вся наша совокупность включает только Пэт, Терри и Ким. Тогда формула

(*x*)*Любит*(*Ким*, *x*)

равносильна конъюнкции:

Любит(*Ким*, *Пэт*) & *Любит*(*Ким*, *Терри*) & *Любит*(*Ким*, *Ким*).

Чтобы всё высказывание было истинным, каждый из входящих в него предикатов должен быть истинным. Формула

(∃*x*)*Любит*(*Ким*, *x*)

равносильна дизъюнкции:

Любит(*Ким*, *Пэт*) v *Любит*(*Ким*, *Терри*) v *Любит*(*Ким*, *Ким*).

Чтобы всё высказывание было истинным, только один из его предикатов должен быть истинным.

Если мы вспомним двоякость законов Де Моргана и применим их к этим двум формулам, то, наверное, не слишком удивимся, обнаружив связь кванторов общности и существования при посредстве отрицания. Следующие две равносильные формулы истинны, если не все любят Терри:

– (*x*)*Любит*(*x*, *Терри*) eq. (∃*x*)–*Любит*(*x*, *Терри*).

Следующие две формулы истинны, если никто не любит Терри:

(*x*)–*Любит*(*x*, *Терри*) eq. –(∃*x*)*Любит*(*x*, *Терри*).

Точно так же

(*x*)*Любит*(*x*, *Терри*) eq. –(∃*x*)–*Любит*(*x*, *Терри*).

Формулу справа можно перевести как «нет никого, кто не любит Терри». Точно так же

(∃*x*)*Любит*(*x*, *Терри*) eq. –(*x*)–*Любит*(*x*, *Терри*)

означает «Неверное, никто не любит Терри».

Когда американский математик Чарлз Сандерс Пирс (1839–1914) придумал логические кванторы, он использовал для квантора существования символ Σ, обычно относящийся к суммированию, а для квантора общности – символ Π, обозначающий произведение, подчеркнув тем самым связь между логикой и двоичной арифметикой.

Переменная *x*, которая используется в этих формулах, называется *связанной переменной*, потому что она связана с квантором. Она исполняет ту же роль, что переменная в функции. Любая переменная, не входящая в квантор общности или квантор существования, называется *свободной переменной*. В следующей формуле *x* связана, а *y* свободна:

(∃*x*)*Любит*(*x*, *y*).

Свободные или связанные переменные можно изменить, но только если это не приводит к конфликту с другими переменными. Например, в предыдущей формуле мы можем изменить x на z:

$(\exists z)$*Любит*(z, y).

Формула имеет тот же самый смысл. Но мы не можем заменить связанную переменную x на y, потому что это привело бы к конфликту со свободной переменной и стало бы чем-то совершенно другим.

Одна и та же формула не может содержать одноименные связанные и свободные переменные. В логике первого порядка формула, не содержащая свободных переменных, может считаться *высказыванием* или *предложением*. Не следует употреблять эти термины для формул, содержащих свободные переменные.

Область действия связанных переменных нередко заключают в скобки. В следующем высказывании переменная x связана выражением в скобках:

(x) [*Любит*$(x,$ *Ким*$)$ & *Любит*$(x,$ *Пэт*$)$].

Отметим, что квадратные скобки используются вместо круглых только для удобства чтения утверждения. Высказывание означает, что все любят Ким и любят Пэт; оно имеет тот же смысл, что:

(x)*Любит*$(x,$ *Ким*$)$ & (x)*Любит*$(x,$ *Пэт*$)$.

Теперь две связанные переменные независимы друг от друга, и одна или другая может быть изменена:

(y)*Любит*$(y,$ *Ким*$)$ & (x)*Любит*$(x,$ *Пэт*$)$.

Следующее утверждение истинно, если кто-то любит и Ким, и Пэт:

$(\exists x)$[*Любит*$(x,$ *Ким*$)$ & *Любит*$(x,$ *Пэт*$)$].

Однако его смысл меняется, стоит только разделить эти два предиката:

$(\exists x)$*Любит*$(x,$ *Ким*$)$ & $(\exists x)$*Любит*$(x,$ *Пэт*$)$.

Теперь есть кто-то, кто любит Ким, и кто-то, кто любит Пэт, но это не обязательно один и тот же человек.

Заменим в последних нескольких формулах конъюнкцию на дизъюнкцию:

(x)[*Любит*$(x,$ *Ким*$)$ v *Любит*$(x,$ *Пэт*$)$].

Это высказывание истинно, если любой человек любит Ким или любит Пэт (или любит обеих). Оно истинно, если Терри любит Ким, но не Пэт, и если Терри любит Пэт, но не Ким. Если разделить эти два предиката, значение высказывания меняется:

(x)*Любит*$(x,$ *Ким*$)$ v (x)*Любит*$(x,$ *Пэт*$)$.

Оно истинно, только когда все любят Ким, или все любят Пэт, или все любят обеих.

Применим квантор существования к дизъюнкции:

$(\exists x)[$*Любит*$(x,$ *Ким*$)$ v *Любит*$(x,$ *Пэт*$)]$.

Существует некто, кто любит или Ким, или Пэт, или обеих. Разделение двух предикатов сохраняет смысл высказывания:

$(\exists x)$*Любит*$(x,$ *Ким*$)$ v $(\exists x)$*Любит*$(x,$ *Пэт*$)$.

Ко всем логическим функциям применимы два основных отношения. В обоих примерах A – это предикат и a – элемент области определения. Первое отношение:

$(x)A(x) \rightarrow A(a)$.

Если предикат истинен всюду, то он истинен для каждого элемента. Второе отношение:

$A(a) \rightarrow (\exists x)A(x)$.

Кванторы могут применяться друг к другу. Например,

$(\exists x)(y)$*Любит*(x, y)

нужно понимать следующим образом:

$(\exists x)[(y)$*Любит*$(x, y)]$.

Оно истинно, если есть кто-то, кто любит всех. Если поменять порядок следования кванторов, то смысл становится несколько иным:

$(y)(\exists x)$*Любит*(x, y).

Оно истинно, если все любимы кем-то, но это не обязательно один и тот же человек. Например, если Ким любит Ким и Терри любит Терри, но они не любят друг друга, то

$(y)(\exists x)$*Любит*(x, y)

истинно, но

$(\exists x)(y)$*Любит*(x, y)

ложно. Однако если высказывание, начинающееся с квантора существования, истинно, то таковым является и другое. Это соотношение заключено в теореме *11.26 из *Principia Mathematica*, которую мы видели в начале этой главы:

$*11 \cdot 26. \vdash :.(\exists x) : (y).\phi(x, y) :\supset: (y) : (\exists x).\phi(x, y),$

где $\phi(x, y)$ – предикат. В системе обозначений Тьюринга это выглядит так:

$(\exists x)(y)\phi(x, y) \rightarrow (y)(\exists x)\phi(x, y).$

Если в формуле есть цепочка кванторов общности, то их можно произвольно менять местами, не меняя смысла формулы. То же справедливо для цепочки кванторов существования. (Превратите высказывание в составную конъюнкцию или дизъюнкцию и убедитесь, что это так.) Однако в самом общем случае в цепочке чередующихся кванторов общности и существования такие перестановки невозможны без изменения смысла формулы.

Как и в логике высказываний, формулы могут быть вычислены независимо от значений области определения и предикатов. Считается, что формула

$(x)[F(x) \vee -F(x)]$

всюду справедлива, потому что она истинна независимо от области определения и логической функции F. Однако следующая формула всегда ложна:

$(\exists x)(F(x) \ \& \ -F(x)).$

Говорят, что такая формула *опровержима*. В подобном случае есть формулы, которые попадают между этими двумя. Они очень просты:

$(x)F(x).$

Нетрудно представить себе область определения x и функцию F, где формула истинна. Скажем, область определения – натуральные числа, а F означает «больше или равно нулю». Так же легко задать область определения и функцию, где она ложна. Скажем, функция F возвращает истину, если аргумент – простое число. Говорят, что такая формула «выполнима», потому что она истинна при определенных истолкованиях.

Справедливость и выполнимость – разные стороны одной и той же проблемы, потому что эти понятия связаны: высказывание или выполнимо, или опровержимо. Если высказывание $\mathfrak{A}$ справедливо, то оно и выполнимо (но не обязательно наоборот). Если $\mathfrak{A}$ выполнимо, но не справедливо, то $-\mathfrak{A}$ тоже выполнимо, но не опровержимо. $\mathfrak{A}$ справедливо тогда и только тогда, когда $-\mathfrak{A}$ не выполнимо.

Слова *справедливость* и *выполнимость* иногда связывают с *семантическим* подходом к математической логике, называемым так потому, что он имеет дело с истинными значениями включаемых в него высказываний.

Другой подход к математической логике – *синтаксический*. Вы начинаете с аксиом и выводите теоремы. Говорят, что такие теоремы *доказуемы*, то есть они – следствия аксиом. При синтаксическом подходе к логике нет нужды погружаться в беспорядок – возможно, метафизический – в понимании истины.

Для логики высказываний Hilbert & Ackermann зафиксировали четыре довольно очевидные аксиомы, выведенные из *Principia Mathematica*:

(a) $X \vee X \rightarrow X$
(b) $X \rightarrow X \vee Y$
(c) $X \vee Y \rightarrow Y \vee X$
(d) $(X \rightarrow Y) \rightarrow (Z \vee X \rightarrow Z \vee Y)$

Хотя в аксиомах присутствуют только дизъюнкция и импликация, их можно применять и к конъюнкции тоже, если считать X & Y сокращенной записью для $-(-X \vee -Y)$.

Для логики первого порядка Hilbert & Ackermann добавили еще две аксиомы. Для любого предиката F следующие утверждения суть аксиомы:

(e) $(x)F(x) \rightarrow F(y)$
(f) $F(y) \rightarrow (\exists x)F(x)$

Аксиомы дополнены правилами построения сложных утверждений из простых:

1. Подстановка: чтобы избежать конфликтов между связанными и свободными переменными, логическая переменная может быть непротиворечиво заменена формулой; свободные и связанные переменные могут быть заменены, если это не приводит к конфликту; предикаты могут быть заменены формулами.
2. Импликация: если формула $\mathfrak{A}$ истинна и формула $\mathfrak{A} \rightarrow \mathfrak{B}$ истинна, то $\mathfrak{B}$ истинна.

Второе правило известно как *modus ponens* (метод подтверждения). Оно выглядит очевидным, но должно быть именно аксиомой. Его нельзя вывести, а если и кажется, что можно, то стоит заглянуть в эссе Льюиса Кэрролла «Что сказала черепаха Ахиллесу»[1].

Все, что можно вывести из этих шести аксиом и двух правил, называют *теоремой*. Сам вывод называется *доказательством*. Говорят, что любая формула, ставшая результатом доказательства, *доказуема*. Теорема – это доказуемая формула.

Например, если $\mathfrak{A}$ и $\mathfrak{B}$ – теоремы, то аксиома (c) и правило (1) позволяют утверждать, что

$$\mathfrak{A} \vee \mathfrak{B} \rightarrow \mathfrak{B} \vee \mathfrak{A}$$

доказуема и, следовательно, это тоже теорема.

Применять правила можно двумя способами: начать с аксиом и использовать правила для вывода теорем или начать с формулы и применять правила для сведения ее к аксиомам, чтобы формула стала теоремой. Программы автоматического доказательства, которые обсуждались в начале этой главы, начинали с теорем из *Principia Mathematica* и применяли аксиомы и правила подстановки, чтобы свести их к аксиомам.

Как можно видеть, формализация математики Гильбертом сводит ее к механическому процессу манипуляции символами. Это заметил еще Анри Пуанкаре (1854–1912), который писал: «мы могли бы вообразить машину, с одной стороны которой вводятся аксиомы, а с другой выводятся теоремы, как в той легендарной машине из Чикаго, в которую свиньи входят живьем, а выходят, став ветчиной и сосисками»[2].

Кроме того, все теоремы можно механически последовательно *перечислить* (или *пронумеровать – прим. перев.*). Начав с аксиом, расширенных несколькими логическими переменными и несколькими предикатами, применяем правила подстановки и импликации для каждой возможной комбинации.

Теорема, по определению, – это формула, которая выводима из аксиом, поэтому такое перечисление теорем приведет к любой возможной теореме. Тогда сам собой возникает вопрос: действительно ли

---

[1] Lewis Carroll, «What the Tortoise Said to Achilles», *Mind*, New Series, Vol. 4, No. 14 (Apr. 1895), 278–280, и часто переиздавалось.

[2] Henri Poincaré, *Science and Method*, translated by Francis Maitland (Thomas Nelson & Sons, 1914; Dover, 2003), 147.

эти теоремы – то же, что и всюду справедливые формулы? Или возможны некие всюду справедливые формулы, которые нельзя создать на базе аксиом?

Используя в качестве трамплина книгу Hilbert & Ackermann, Курт Гёдель установил равносильность семантического и синтаксического подходов к логике первого порядка – сначала в своей докторской диссертации 1929 года «Über die Vollständigkeit des Logikkalküls» («О полноте логических исчислений»), а затем в статье 1930 года «Die Vollständigkeit der Axiome des logischen Funktionenkalküls» («Полнота аксиом исчисления функций логики»).

Еще до Гёделя было известно, что каждая доказуемая формула также всюду справедлива. Это называлось *разумностью* (*целостностью*) и было существенно для логической системы. Гёдель доказал, что каждая всюду справедливая формула также доказуема. Это одно из возможных определений «полноты» логической системы, и действительно, в названиях статей Гёделя есть слово *Vollständigkeit* – полнота. Теорема Гёделя о полноте показала, что аксиомы полны – что аксиоматическая система, предложенная в Hilbert & Ackermann для чистой логики предикатов, достаточна для перечисления (нумерации) каждого всюду справедливого утверждения в этой логике.

Можно предположить, что перечисление (нумерация) теорем в теореме Гёделя о полноте обеспечивает базис для разрешающей процедуры в логике первого порядка. Допустим, например, что нам нужно установить доказуемость формулы $\mathfrak{A}$. Мы начинаем перечислять все теоремы и сопоставлять их с $\mathfrak{A}$. Но если $\mathfrak{A}$ не доказуема, мы не получим совпадения и не будем знать, когда остановиться.

Да, но тогда можно поступить хитрее: при перечислении (нумерации) теорем сравнивать с $\mathfrak{A}$ и саму теорему, *и ее отрицание* (или сравнивать каждую теорему с $\mathfrak{A}$ и ее отрицанием). Но и это еще не гарантирует совпадения, потому что $\mathfrak{A}$ может быть просто выполнимой, но не всюду справедливой или опровержимой (противоречивой). По этой причине говорят, что основанная на перечислении (нумерации) процедура только *полуразрешима*. Только если мы знаем заранее, что или $\mathfrak{A}$, или $-\mathfrak{A}$ всюду справедливы, то процедура успешно завершится. Даже после статьи Гёделя 1930 года Entscheidungsproblem для логики первого порядка была все еще открытой проблемой.

Более известная статья Гёделя была опубликована в 1931 году и касалась применения логики первого порядка к элементарной арифметике – сложению и умножению. Используя эту арифметику, Гёдель смог присвоить номер каждой формуле и каждому доказательству.

Гёдель создал предикат с именем *Bew* (от *beweisbar*, что значит *доказуемый)* и смог применить этот предикат к гёделеву числу его отрицания, создав формулу, которая утверждает свою собственную недоказуемость.

Таким образом, в рамках логической системы, поддерживающей элементарную арифметику, можно обнаружить высказывания, которые не могут быть ни доказаны, ни опровергнуты. Хотя этот факт известен как Теорема Гёделя о неполноте, статья на самом деле называлась «Über formal unentscheidbare Sätze der *Principia Mathematica* under verwandter Systeme I» («О формально неразрешимых высказываниях *Principia Mathematica* и связанных Системах I»)[1]. В названии нет ни полноты, ни неполноты, а есть *unentscheidbare Sätze* – неразрешимое высказывание.

Подписывает ли теорема Гёделя о неполноте смертный приговор общей разрешающей процедуре? Необязательно, хотя общая разрешающая процедура, конечно, стала казаться в 1931 году менее вероятной, чем в 1930-м. Теорема Гёделя о неполноте – о неразрешимых высказываниях, тогда как Entscheidungsproblem касается существования общего процесса определения доказуемости любой заданной формулы. Разрешающая процедура, если бы она существовала, должна была отнести неразрешимое высказывание к недоказуемым.

Ранние компьютерные программы, которые доказывали теоремы со страниц *Principia Mathematica,* определенно действовали *не* так, начиная с аксиом и методично выводя все доказываемые формулы. Статья Ньюэлла–Саймона–Шоу называет такой подход «Алгоритмом Британского музея», потому что он напоминает доскональный осмотр каждого экспоната в надежде найти именно то, что нужно. Этот подход с позиции грубой силы после трезвой его оценки был отклонен этими ранними исследователями. Как выразился Мартин Дэвис,

> Было слишком очевидно, что попытка генерировать доказательство чего-то нетривиального, начав с аксиом логических систем и методично применяя правила вывода во всех возможных направлениях, непременно должна была привести к гигантскому комбинаторному взрыву[2].

---

[1] Все три цитируемые статьи Гёделя наиболее доступны в Kurt Gödel, *Collected Works: Volume I, Publications 1929–1936* (Oxford University Press, 1986).

[2] Martin Davis, «The Early History of Automated Deduction» в кн. Alan Robinson and Andrei Voronkov, eds., *Handbook of Automated Reasoning* (MIT Press, 2001), Vol. I, 3–15.

Только один программист действовал, не страшась комбинаторных взрывов, и им был Алан Тьюринг. Воображаемые компьютеры Тьюринга имели неограниченную память и все на свете время для работы, поэтому Тьюринг смог отправиться туда, где не ступала нога машинно-ограниченного программиста.

В предыдущей главе я остановился посередине раздела 9 «Пространство вычислимых чисел» статьи Тьюринга. Тьюринг начинает раздел с необходимости убедить нас в том, что числа, вычислимые его машиной, включают «все числа, которые было бы естественно считать вычислимыми» (стр. 249 статьи Тьюринга; стр. 216 этой книги).

Затем у Тьюринга идет пункт, обозначенный римской цифрой I (означающей первый из нескольких аргументов) и «Тип(*a*)», означающий «прямой призыв к интуиции».

Идущий следом пункт под римской цифрой II (это второй из аргументов Тьюринга) и далее «Тип(*b*)» означают «Доказательство равносильности двух определений (в случае, когда новое определение требует еще большего привлечения интуиции)».

Единственное предложение вслед за этим заголовком имеет три сноски. Первая лишь поясняет, что он говорит об ограниченном функциональном исчислении, которое известно нам как логика предикатов первого порядка. Вторую сноску мы пока пропустим, но обсудим ее довольно скоро.

> II. [Тип (*b*)].
>
> Если система обозначений функционального исчисления Гильберта† изменяется так, чтобы быть систематической и чтобы включать только конечное число символов, становится возможным создать автоматическую‡ машину K, которая найдет все доказуемые формулы исчисления§.

---

† Выражение «функциональное исчисление» используется всюду в смысле *ограниченного* функционального исчисления Гильберта.

‡ Чтобы сделать это, самое естественное – создать сначала машину для выбора (§2). А потом уже легко создать необходимый автомат. Мы можем считать, что выбор – это всегда выбор между двумя возможностями 0 и 1. Тогда каждое доказательство будет определяться последовательностью выборов $i_1, i_2, ..., i_n$ ($i_1 = 0$ или 1, $i_2 = 0$ или 1, ..., $i_n = 0$ или 1), и, следовательно, число $2^n + i_1 2^{n-1} + i_2 2^{n-2} + ... + i_n$ полностью определяет доказательство. Автоматическая машина выполняет последовательно доказательство 1, доказательство 2, доказательство 3... .

§ Автор нашел описание такой машины.

Я полагаю, что Тьюринг называет машину $\mathcal{K}$ в связи с немецким словом *Kalkül*. Хоть это и не вполне понятно из приведенного отрывка, но в конце концов вы увидите, что Тьюринг описывает алгоритм Британского музея. Машина $\mathcal{K}$ начинает работу или с уже записанными на ленту аксиомами, или же с записи аксиом на ленту. Это основные аксиомы логики первого порядка плюс какие-то ещё аксиомы, которые нужны для дополнительных предикатов. Машина последовательно применяет правила вывода для генерации всех доказуемых утверждений исчисления.

Тьюринг требует, чтобы система обозначений логики первого порядка «была изменена так, чтобы быть систематической». Конечно, нас не волнует равносильность утверждений, отличающихся лишь именами переменных или необязательными скобками. Например, следующие три утверждения должны считаться одинаковыми:

$(x)(\exists y)\phi(x, y)$
$(y)(\exists x)\phi(y, x)$
$(x)(\exists y)(\phi(x, y))$

Систему обозначений можно сделать систематической, потребовав, чтобы переменные всегда имели вид x_i и чтобы они появлялись в каждой формуле в порядке следования индексов. Кроме того, круглые скобки должны использоваться только там, где они влияют на порядок применения операций. Другой и наиболее приемлемый на практике способ – записывать формулы в префиксной нотации, исключающей круглые скобки, например в так называемой польской нотации, которую Ян Лукасевич (Jan Łukasiewicz) (1878–1956) изобрел специально для логики высказываний. Вместо

$(A \vee B) \& (C \vee D)$

утверждение можно было бы записать так:

$\&\vee AB\vee CD$.

Здесь важно то, что эта машина генерирует все доказуемые формулы. Из теоремы Гёделя о полноте мы знаем, что это множество то же, что все всюду справедливые формулы.

Я полагаю, что Тьюринг пытается здесь обратиться к тем первым читателям его статьи, которые, возможно, сомневаются в способностях его машин вычислять вещественные числа произвольной степени сложности. В 1936 году гораздо больше доверяли эффективности

логики первого порядка, чем компьютерам. С точки зрения реализации эта машина $\mathcal{K}$ кажется вполне осуществимой. Конечно, она намного проще машин, вычисляющих такие вещественные числа, как корень седьмой степени из 10. Машина $\mathcal{K}$ работает исключительно со строками символов и склеивает их различными способами согласно правилам подстановки. Большая часть логики сравнения и подстановки строк уже была представлена в функциях, которые Тьюринг использовал в Универсальной машине.

Вторая сноска этого отрывка, начинающаяся со слов «Чтобы сделать это, самое естественное – создать...», на самом деле описывает несколько иной подход, который, кажется, больше подходит к логике высказываний, нежели к логике первого порядка. Задав фиксированное число n логических переменных, можно разработать систему для генерации всех правильных формул, вставляя между этими переменными логические связки.

Затем для каждой из этих формул, пользуясь таблицами истинности, проверить, является ли она тавтологией.

Если такая правильная формула имеет n логических переменных, то для установления ее справедливости потребуется 2^n проверок. Если считать истиной и ложью двоичные цифры 1 и 0, то каждой проверке соответствует двоичное число из n цифр, где каждая цифра представляет значение истинности одной переменной. В сноске Тьюринга это двоичное число из n цифр слегка неточно: слагаемого 2^n в начале не должно быть. Проверки могут быть пронумерованы, начиная с 0 и заканчивая $(n - 1)$, чтобы соответствовать значению двоичного числа из n цифр.

Хотя Тьюринг намерен применить машину K для генерации утверждений логики первого порядка, а не логики высказываний, вы увидите, что каждый раз, когда ему нужны целые числа, он берет лишь *конечный* отрезок неотрицательных целых чисел. И он никогда не требует бесконечной области определения, поэтому очевидно, что его формулы первого порядка могут быть преобразованы в логические высказывания, и он мог бы тогда использовать таблицы истинности.

Введение натуральных чисел в систему логики первого порядка всегда несколько натянуто, но весьма и весьма существенно, когда мы собираемся применить логику к числовым понятиям. Поглощение чисел логикой обычно начинается с вариаций на тему аксиом Пеано. Это пять аксиом, выдернутых из первоначальных девяти аксиом, предложенных Джузеппе Пеано в его небольшой 1889 года книжке

Основы арифметики, представленные новым методом[1] и основанных на брошюре 1888 года Рихарда Дедекинда *Was sind und was sollen die Zahlen?* (*Что такое число и каким оно должно быть?*[2])

Аксиомы Пеано строятся вокруг понятия «преемник». Интуитивно это число, следующее непосредственно за данным числом. Например, преемник числа 12 – это число 13. Аксиомы Пеано гарантируют, что у каждого числа есть преемник, причем единственный. Лишь одно число не является преемником никакого другого числа. У Пеано это число 1, хотя в наше время натуральный ряд обычно начинают с нуля.

Вот один из вариантов аксиом Пеано на естественном языке:

1. Нуль есть число.
2. У каждого числа есть преемник, который тоже является числом.
3. Нуль не является преемником никакого другого числа.
4. Два числа, которые являются преемниками одного и того же числа, равны.
5. Если что-то истинно для нуля и если из того, что это же истинно для некоторого числа, следует, что это истинно и для его преемника, то это истинно для всех чисел.

Пятая аксиома обычно известна как метод математической индукции, и она лежит в основе многих математических доказательств о натуральных числах. (Тьюринг применит его дважды в доказательствах в следующих двух разделах.) Однако выразить индукцию на языке логики первого порядка проблематично. Индукция – это, по сути, понятие логики второго порядка, потому что она должна применяться ко всем предикатам, аргументы которых – натуральные числа. Понятие равенства – тоже понятие второго порядка, и именно поэтому вы поймете нежелание логиков – и Тьюринга в его статье – вводить предикат, который истинен, если два его аргумента равны.

Даже проблема включения первых четырех аксиом Пеано в язык логики первого порядка нетривиальна, и (как вы поймете) представление Тьюринга неадекватно (недостаточно). Эта проблема не оказывает влияния на его доказательство или выводы, но она определенно тревожит.

Другая проблема касается представления самих натуральных чисел. Странной традиции использовать 0, 1, 2, 3 и т. д. просто не должно

---

[1] Выдержки – в кн. Jean van Heijenoort, ed., *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931* (Harvard University Press, 1967), 83–97. Полностью – в кн. *Selected Works of Giuseppe Peano*, translated and edited by Hubert C. Kennedy (George Allen & Unwin, 1973), 101–134.

[2] Перепечатано в William Ewald, ed., *From Kant to Hilbert: A Source Book in the Foundations of Mathematics* (Oxford University Press, 1996), Vol. II, 787–833.

было быть. Ничто, кроме вековых традиций и грубого внушения средней школы, не говорит нам о том, что преемником числа 12 должно быть 13. В частности, если мы придумываем механические способы обработки символов, было бы гораздо лучше, если бы они сами несли в себе понятие преемственности.

В первом томе *Grundlagen der Mathematik* (1934) – книги, которую Тьюринг позже упоминает в своей статье, – Пауль Бернайс использует штрихи, чтобы обозначить, что одно число – преемник другого. Например, если a – число, то a' – преемник этого числа, а a'' – следующий преемник. Бернайс также использует символ 0 для представления нуля, тогда $0'$ – преемник нуля, а $0''$ – следующий преемник[1]. Что это за число *на самом деле*? Просто посчитайте штрихи у 0. Самый ранний пример такой системы обозначений в работах Гильберта и его последователей я обнаружил в книге Давида Гильберта «Основания математики» 1927 года издания[2].

Тьюринг *совершенно* не принимает такого подхода. Он явно не желает использовать даже символ 0. Вместо этого первое натуральное число он обозначает символом u. (У него даже не вполне ясно, что такое u – 0 или 1 и имеет ли он значение. В своих примерах я предполагаю, что u – это 0.) Преемники u – это u', u'', u''' и т. д. Такая система обозначений может стать громоздкой для больших чисел и не позволит представлять произвольные числа, такие как n или x. Приняв во внимание подсказку из Hilbert & Bernays, Тьюринг использует обозначение $u^{(r)}$ для u с r штрихами. Например, u'''''' можно представить как $u^{(5)}$, известное нам как число пальцев на одной руке.

Тьюринг определяет логическую функцию $N(x)$, которая истинна, если x – неотрицательное целое число. Если мы и без того ограничиваемся пространством неотрицательных целых чисел, то эта функция, в сущности, нам ни о чем не говорит, но Тьюринг находит ее полезной для выражения аксиом Пеано.

Тьюринг также определяет логическую функцию $F(x, y)$, которая истинна, если y – преемник x, или как в обычной арифметике $y = x + 1$. Учтите, что F не *предоставляет* преемника и не *вычисляет* его. Она – то, что программисты называют булевой функцией. Она становится истинной, только когда y – действительно преемник x.

Если у нас есть подходящий предикат преемника (скажем, с именем *Succ*, чтобы отличать его от предиката Тьюринга) и мы установили

---

[1] David Hilbert and Paul Bernays, *Grundlagen der Mathematik*, Volume I (Springer, 1934), 218.

[2] David Hilbert, «Foundations of Mathematics», *From Frege to Gödel*, 467.

аксиому для математической индукции, становится возможным определить предикат под названием *Sum*(*x*, *y*, *z*), который истинен, если *z* равняется *x* + *y*. Предикат *Sum* основан на следующих трех аксиомах:

$(x)Sum(x, u, x)$
$(x)Sum(u, x, x)$
$(x)(y)(z)(r)(s)(Sum(x, y, z) \ \& \ Succ(y, r) \ \& \ Succ(z, s) \rightarrow Sum(x, r, s))$

Первые две аксиомы определяют сложение нуля с любым числом. Третья утверждает, что если $x + y = z$ и $r = y + 1$ и $s = z + 1$, то $x + r = s$.

Подобным образом можно определить предикат умножения *Product*(*x*, *y*, *z*):

$(x)Product(x, u, u)$
$(x)Product(u, x, u)$
$(x)(y)(z)(r)(s)(Product(x, y, z) \ \& \ Succ(y, r) \ \& \ Sum(z, x, s) \rightarrow$
$Product(x, r, s))$

Первые две аксиомы определяют умножение на нуль, а третья утверждает, что если

$x \times y = z$ и $r = y + 1$ и $z + x = s$, то $x \times r = s$.

Давайте пойдем немного дальше. Предикат *IsEven*(*x*) может быть определен как

$(\exists y)Product(y, u'', x)$

IsEven(*x*) истинен, если существует такой *y*, что $x = y \times 2$. Предикат *IsOdd*(*x*) – это то же, что $-IsEven(x)$. И здесь я остановлюсь, поскольку этого достаточно для моих примеров.

Теперь пусть α – последовательность, и давайте обозначим $G_\alpha(x)$ высказывание «*x*-й символ α есть 1» так, чтобы[‖] $-G_\alpha(x)$ означало «*x*-й символ α есть 0».

[‖] Знак отрицания пишется перед выражением, а не над ним.

В сноске Тьюринг указывает на то, что он использует символ отрицания, отличный от гильбертова. α – это последовательность, которую мы обычно вычисляли при проектировании узкоспециализированной машины. Здесь же Тьюринг предлагает обозначать цифрами 1 и 0 соответственно значения «истина» и «ложь» вычисляемых предикатов. Например, последовательность, соответствующая квадратному корню из двух из главы 6, начинается так:

1011011010...

Если мы нумеруем цифры, начиная с 0, то $G_\alpha(0)$ истинен, $G_\alpha(1)$ ложен, $G_\alpha(2)$ истинен, $G_\alpha(3)$ истинен, $G_\alpha(4)$ ложен и т. д. Похоже, это довольно сложный предикат. Гораздо более простая последовательность – у машины Тьюринга из Примера I:

0101010101...

Если помните, эта последовательность – двоичное представление числа 1/3. Логическая функция, которая описывает эту последовательность, очень проста, если мы определим $G_\alpha(x)$ *как* $IsOdd(x)$.

Предположим далее, что мы можем найти ряд свойств, которые определяют последовательность α и которые могут быть выражены посредством $G_\alpha(x)$ и логических функций $N(x)$, означающих «x – неотрицательное целое число», и $F(x, y)$, означающей «$y = x + 1$».

Здесь Тьюринг вводит два предиката, которые он будет часто применять в оставшейся части статьи. F – функция преемника, крайне важная для определения натуральных чисел. Гораздо меньше меня убеждает необходимость функции N, если ее область определения явно ограничивается натуральными числами.

Я не уверен, что именно подразумевает Тьюринг под «рядом свойств». То, что нам здесь действительно нужно, – так это аксиомы, поддерживающие логические функции, из которых составляется $G_\alpha(x)$. В моем простом примере эти аксиомы должны включать аксиомы для предикатов *Sum* и *Product*.

Когда мы соединяем конъюнкцией все эти формулы вместе, мы получаем формулу, скажем A, которая определяет α.

Конъюнкция аксиом на самом деле не определяет α, а скорее обеспечивает основание для определения α.

Члены $\mathfrak{A}$ должны включать необходимые аксиомы Пеано, а именно:

$$(\exists u)\, N(u)\ \&\ (x)\,(N(x) \rightarrow (\exists y)\, F(x, y))\ \&\ (F(x, y) \rightarrow N(y)),$$

которые мы обозначим как P.

P – в честь Пеано, конечно. Это конъюнкция трех членов. Первый указывает, что *u* существует; второй говорит, что для каждого *x* есть преемник *y*, ну а третий указывает, что преемник – тоже натуральное число. Эта формула *не* устанавливает единственность нуля или единственность преемников, и это – проблема. В Hilbert & Bernays есть следующие три аксиомы для функции преемника (которую они называют *S*)[1].

$(x)(\exists y)S(x, y)$
$(\exists x)(y)\neg S(y, x)$
$(x)(y)(r)(s)(S(x, r)\ \&\ S(y, r)\ \&\ S(s, x) \rightarrow S(s, y)$

Первая утверждает, что у каждого числа есть преемник; вторая говорит, что существует число, у которого нет преемника; третья говорит, что если *r* – преемник *x* и *y*, а *x* – преемник *s*, то *y* – тоже преемник *s*, устанавливая, по существу, единственность преемников.

Когда мы говорим «$\mathfrak{A}$ определяет α», мы подразумеваем, что $\neg\mathfrak{A}$ – недоказуемая формула, а также что для каждого *n* одна из следующих формул (A_n) или (B_n) доказуема.

$$\mathfrak{A}\ \&\ F^{(n)} \rightarrow G_{\alpha}(u^{(n)}), \qquad (A_n)^{\P}$$
$$\mathfrak{A}\ \&\ F^{(n)} \rightarrow (\neg G_{\alpha}(u^{(n)})), \qquad (B_n)$$

где $F^{(n)}$ означает $F(u, u')\ \&\ F(u', u'')\ \&\ \ldots\ F(u^{(n-1)}, u^{(n)})$.

¶ Последовательность из *r* штрихов обозначается $^{(r)}$.

Сноска у пометки (A_n) относится к соглашению об использовании верхнего индекса в круглых скобках для обозначения ряда штрихов. Тьюринг применяет верхний индекс и к функции преемника *F* для обозначения конъюнкции функций преемника, говоря о том, что 1 – преемник 0, 2 – преемник 1 и т. д.

Конъюнкция функций преемника у Тьюринга неточна, потому что она не гарантирует уникальности преемников. Например, каково значение истинности для $F(u', u''')$? Ничто не говорит нам о том, что оно ложно. Одно довольно несложное исправление существенно (хотя и не в полной мере) усиливает $F^{(n)}$ – это включение в конъюнкцию отрицаний всех ложных предикатов преемников, например $\neg F(u, u'')$, $\neg F(u', u)$ и т. д. вплоть до $u^{(n)}$.

[1] Hilbert and Bernays, *Grundlagen der Mathematik*, Volume I, 209. Я чуть изменил нотацию согласно Тьюрингу.

Здесь n – номер цифры, начинающейся с цифры 0 и последовательно увеличивающейся. Последовательность генерируется с цифры 0, цифры 1, цифры 2 и т. д.

Вычисление каждой цифры требует лишь конечного числа неотрицательных целых чисел, поэтому формула $F^{(n)}$ – конечная конъюнкция своих членов. Однако в некоторых случаях формула может требовать несколько больше целых чисел. Например, для цифры 0 формула требует только u, а в моем примере для определения функции *IsOdd* требуется еще и u'', поэтому верхний индекс у F должен быть на самом деле наибольшим числом из n и 2.

С этими небольшими поправками следующие формулы будут доказуемы:

B_0: $\mathfrak{A} \,\&\, F^{(2)} \to -IsOdd\,(u)$
A_1: $\mathfrak{A} \,\&\, F^{(2)} \to IsOdd\,(u')$
B_2: $\mathfrak{A} \,\&\, F^{(2)} \to -IsOdd\,(u'')$
A_3: $\mathfrak{A} \,\&\, F^{(3)} \to IsOdd\,(u''')$
B_4: $\mathfrak{A} \,\&\, F^{(4)} \to -IsOdd\,(u'''')$
A_5: $\mathfrak{A} \,\&\, F^{(5)} \to IsOdd\,(u''''')$

и т. д. Если помните, $\mathfrak{A}$ включает все аксиомы, необходимые для поддержки функции *IsOdd*. Эти результаты соответствуют первым шести цифрам последовательности: 0, 1, 0, 1, 0, 1. Заметьте, что цифра 1 соответствует доказуемости (A_n), а 0 – доказуемости (B_n).

[253]

Тогда я говорю, что α – вычислимая последовательность: машину K_α для вычисления α можно получить путем довольно простой модификации K.

Если помните, машина $\mathcal{K}$ выводила из аксиом все доказуемые формулы.

Мы делим движение K_α на отрезки. n-й отрезок предназначен для поиска n-го символа α. По окончании $(n-1)$-го отрезка после всех символов печатается двойное двоеточие ::, а вся последующая работа целиком выполняется в клетках правее этого двойного двоеточия. Первый шаг – записать символ «A» и следом формулу (A_n), а затем – символ «B» и следом (B_n).

Для этого примера и n, равного 5, машина сначала пишет «A» и «B», сопровождаемые двумя возможными вариантами:

A $\mathfrak{A}$ & $F^{(5)} \rightarrow IsOdd(u'''''')$ **B** $\mathfrak{A}$ & $F^{(5)} \rightarrow -IsOdd(u'''''')$.

Однако это неточно: «A» и «B» не должны выделяться жирным шрифтом, член $\mathfrak{A}$ должен быть явной конъюнкцией всех аксиом, $F^{(5)}$ должна быть явной конъюнкцией остальных аксиом, *IsOdd* должна быть отрицанием приведенной ранее функции *Product*, а все функции, видимо, должны получить более короткие имена.

Но главное здесь в том, что одно или другое из этих двух утверждений должно быть доказуемым. Для выполнения этой работы в распоряжении машины – вся лента правее этих двух формул. Возможно, она сначала запишет на ленту аксиомы, а потом уже начнет работу по выводу доказуемых формул.

Затем машина K_α начинает выполнять работу машины K, но как только доказуемая формула найдена, она сравнивается с (A_n) и с (B_n). Если эта формула совпала с (A_n), печатается символ «1» и n-й отрезок заканчивается. Если она совпала с (B_n), печатается «0» и отрезок заканчивается. Если она отлична от обеих, работа K продолжается с точки, в которой она была отложена. Рано или поздно одна из формул (A_n) или (B_n) будет достигнута; это следует из наших гипотез о α и $\mathfrak{A}$ и известном поведении K. Следовательно, n-й отрезок в конце концов будет пройден. K_α не имеет циклов; α вычислима.

Думается, машина $\mathcal{K}_\alpha$ могла бы быть обобщена подобно Универсальной машине Тьюринга. Она могла бы начать работу с лентой, на которой уже записаны аксиомы. Стоит записать на ленту другие аксиомы и другие функции, и машина вычислит любую последовательность, определяемую логикой первого порядка.

Можно также показать, что числа α, определяемые таким методом применения аксиом, включают все вычислимые числа. Это делается путем описания вычислительных машин в терминах функционального исчисления.

В последнем разделе своей статьи (и следующей главе этой книги) Тьюринг действительно опишет вычислительную машину в терминах логики первого порядка. А пока он хочет напомнить читателю, что не каждое число может быть вычислено машиной, в частности последовательности, которые посредством нулей и единиц сообщают нам, какие из Описателей соответствуют приемлемым машинам.

Нужно помнить, что мы придали фразе «$\mathfrak{A}$ определяет α» довольно специфический смысл. Вычислимые числа не включают все (в обычном понимании) определимые числа. Пусть δ – последовательность, чей n-й символ равен 1 или 0 в зависимости от того, является или нет n подходящей. Это – прямое следствие теоремы из §8 о том, что δ невычислима. Можно (насколько теперь мы знаем) вычислить любое заданное число символов δ, но не универсальным процессом. Если уже вычислено довольно много символов δ, нужен совершенно новый метод для получения последующих символов.

Этим Тьюринг заканчивает свой второй аргумент, утверждающий, что его машины могут вычислять числа, считающиеся вычислимыми в общепринятом смысле. Дальше идет третий аргумент. Тут стоит напомнить, что Тьюринг имеет в виду «состояние памяти» («state of mind») человека-вычислителя. Некоторые читатели могут посчитать, что «состояние памяти» человека – слишком аморфное понятие, чтобы быть встроенным в машину.

Я позволю себе закончить эту главу, не прерывая краткого описания Тьюринга того, как на самом деле можно встроить состояние памяти в структуру машины.

III. Это можно считать модификацией I или следствием II.

Мы предполагаем, как в I, что вычисление производится на ленте; при этом мы оставляем в стороне представление физически более точного аналога «состояния памяти». Вычислитель всегда может прервать свою работу, уйти и вообще забыть о ней, а позже вернуться и продолжить ее. Когда он это делает, он должен оставить метку команды (записанную в некотором стандартном виде), объясняющую, как прерванная работа должна быть продолжена. Эта метка – подобие «состояния памяти». Мы будем считать, что вычислитель работает в такой беспорядочной манере так, что он никогда не делает более одного шага за один раз. Метка команды должна позволять ему выполнить один шаг и записать следующую метку. Таким образом, ход вычислений на любом этапе полностью определяется

[254]

меткой команды и символами на ленте. То есть состояние системы может быть описано единственным выражением (последовательностью символов), состоящим из символов на ленте, сопровождаемых символом Δ (который, как мы предполагаем, не должен появляться

нигде больше), и затем меткой команд. Это выражение можно назвать «формулой состояния». Мы знаем, что формула состояния на любом заданном этапе определяется формулой состояния до выполнения последнего шага, и мы предполагаем, что отношение этих двух формул выразимо в функциональном исчислении. Другими словами, мы предполагаем, что существует аксиома $\mathfrak{A}$, выражающая правила управления поведением вычислителя, отношением формулы состояния на любом этапе к формуле состояния на предыдущем этапе. Если это так, мы можем создать машину для записи последовательных формул состояния и, следовательно, для вычисления требуемого числа.

Глава 13

Вычислимые функции

Когда вы в последний раз загружали свой персональный компьютер работой по вычислению бесконечных цифр иррационального числа? Если вы не один из тех, кто ради развлечения запускает программы, вычисляющие миллионы цифр числа π, то маловероятно, что любая программа, которую вы используете, вычисляет больше цифр, чем ваша любимая программа-калькулятор.

Хотя очевидно, что Алан Тьюринг в своей статье заложил многие принципы и понятия программирования, вычисление бесконечного ряда цифр вещественных чисел, конечно, нетипично для компьютеров прошлого, настоящего и будущего.

Напротив, компьютеры выполняют сложные задачи, которые программисты разложили на небольшие части, называемые либо функциями, либо процедурами, либо подпрограммами, либо методами (в зависимости от конкретного языка программирования). Эти функции обычно выполняют некоторую определенную работу за конечный промежуток времени. Они начинают с некоторого ввода, обрабатывают его, чтобы получить результат, а затем завершаются, передавая управление какой-то другой функции.

Понятие функции пришло из математики. В общих чертах функция – это математический объект, который преобразует вход в выход. Вход называется *аргументом* функции, или *независимой* переменной; выход называется *значением* функции, или *зависимой* переменной. Нередко функции ограничиваются определенными типами чисел или других объектов. Допустимые значения на входе образуют *область определения* функции. Возможные итоговые значения на выходе образуют ее *область изменения*.

В первом параграфе своей статьи Тьюринг упоминает «вычислимую функцию» в качестве предмета предстоящего исследования:

> Хотя очевидно, что тема этой статьи – вычислимые *числа*, почти так же легко определить и исследовать вычислимые функции целочисленной переменной, или действительной, или вычислимой переменной, вычислимые предикаты и т. д... Я надеюсь вскоре представить отчет о взаимоотношениях между вычислимыми числами, функциями и т. д. Он будет включать в себя развитие теории функций действительной переменной в терминах вычислимых чисел.

Тьюринг так и не исследовал эту тему. Как вы увидите в главе 17, понятие вычислимой функции стало весьма важным позже, когда Стивен Клини (в книге 1952 года *Введение в метаматематику*) и Мартин Дэвис (в книге 1958 года *Вычислимость и неразрешимость*) переопределили машину Тьюринга скорее для вычисления целочисленных функций, чем для вычисления вещественных чисел.

В некотором смысле мы уже встречали машины, которые реализуют функции. Универсальная машина – именно такой «зверь», потому что она получает на входе Стандартное описание машины и создает на выходе законченные конфигурации этой машины вместе с последовательностью, которую она должна вычислить.

Ну а как же быть с обычными функциями? На что они должны быть бы похожи? Рассмотрим тригонометрическую функцию sin. На входе – число, задающее угол, обычно в градусах или радианах. Предполагается, что этот угол – часть прямоугольного треугольника. Функция sin вычисляет отношение противолежащей стороны этого треугольника к гипотенузе. В общем случае (чтобы определить функцию sin для углов, больших 90°) из начала декартовой системы координат в любую ее точку проводится отрезок прямой. Для угла между отрезком и осью X (измеренного в направлении против часовой стрелки) функция sin возвращает отношение расстояния от конца отрезка до оси X к длине самого отрезка.

В область определения функции sin входят все вещественные числа, хотя ее значения повторяются с периодом 360°, или 2π радиан. Область изменения функции состоит из вещественных чисел от -1 до 1 включительно.

На самом деле функция sin вычисляется с помощью бесконечного ряда, где x задается в радианах:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$

По этой формуле современные компьютеры вычисляют функцию sin, хотя обычно вычисление выполняет микросхема процессора, а не программа.

Компьютеры не могут хранить произвольные вещественные числа, потому что вещественные числа имеют бесконечное число десятичных разрядов. Вместо этого вещественные числа в компьютерах аппроксимируются рациональными. В 1985 году Институт инженеров по электротехнике и радиоэлектронике (Institute of Electrical and Electronics Engineers, IEEE) опубликовал свой стандарт для двоичной арифметики с плавающей запятой, который используется многими вычислительными системами для хранения чисел согласно представлению, принятому в научной системе обозначений. Например, популярный формат числа с двойной точностью хранится в 64 битах: 1 бит – для знака числа (положительное или отрицательное), 11 – для показателя степени (порядка) и 52 – для мантиссы, обеспечивая точность примерно с 16 десятичными цифрами[1]. Вещественное число 123,456 фактически хранится как два целых числа: 8 687 443 681 197 687 (мантисса) и 46 (показатель степени), потому что 8 687 443 681 197 687 ÷ 2^{46} приблизительно равно 123,456. Это дробь – рациональное число, но не вещественное.

При вычислении синуса на компьютере такое приближение является существенным, потому что позволяет закончить расчет в конечное время. Хотя функция sin вычисляется как бесконечный ряд, абсолютное значение его членов становится все меньше и меньше. Вычисление функции можно прекратить, когда члены ряда не влияют на результат при заданной точности вычислений.

А вот машина Тьюринга просто упивается бесконечной точностью. При вычислении вещественного числа она так и продолжает идти все дальше, дальше и дальше. При вычислении функции sin это могло бы стать настоящей проблемой, потому что каждый из бесконечного ряда членов имеет бесконечное число цифр. Допустим, например, что угол равен просто 1 радиану. Тогда второй член ряда 1/6 имеет бесконечное число цифр независимо от основания (10 или 2) системы счисления. Если машина должна вычислить бесконечное число цифр второго члена, как же она сможет перейти к третьему?

Одна из возможных стратегий состоит в том, чтобы вычислять сначала только первую цифру каждого члена ряда до тех пор, пока очередной его член не станет настолько малым, что его первая цифра окажется равной нулю. Затем вычислять вторую цифру каждого члена, пока он не станет настолько малым, что первые две его цифры

---

[1] Charles Petzold, *Code: The Hidden Language of Computer Hardware and Software* (Microsoft Press, 1999), гл. 23.

окажутся равными нулю, и т. д. Очевидно, что это сложный процесс, особенно если мы не хотим, чтобы машина стирала какие-либо цифры результата после того, как она их вычислила.

Реализация функции sin – это лишь одна из задач. А откуда брать ее аргумент?

Первое, что подсказывает нам чутье, – это «впечатать» куда-нибудь значение угла, когда машина в нем нуждается. Такую идею, очевидно, может навеять мир современных интерактивных компьютеров и экранных калькуляторов, но машина Тьюринга, чтобы позволить такого рода ввод данных, нуждается в некоторой переработке. А это несколько бóльшая работа, чем мы готовы сейчас сделать.

Вторая возможность – «зашить» входное значение функции внутри самой машины. Например, можно было бы создать машину, которая вычисляет синус только 37,85°. Хотя такая машина ограничивалась бы вычислением синуса лишь одного угла, можно было бы надеяться на то, что она спроектирована так, что ее можно довольно легко переделать для других углов.

Третий подход – записать значение угла на ленте. Машина читает ввод, вычисляет синус и записывает результат обратно на ленту. (Могу сказать, что вам нравится такой подход! Мне тоже.)

Четвертый подход – позволить машине генерировать ввод самой себе. Например, машина могла бы сначала вычислить синус 0°, потом – 1°, 2° и т. д., записывая каждый результат на ленту, чтобы составить полную «таблицу» значений. Такой подход должен потребовать от машины ограничиться конечным числом цифр результата.

Пятый подход предполагает две разные машины. Первая машина вычисляет вещественное число, а вторая машина вычисляет синус этого числа. Когда я говорю о двух машинах, на самом деле я говорю об одной машине, которая реализует логику этих двух машин. Мы уже видели машины, объединяемые таким образом.

В разделе 8 (стр. 181–182 этой книги) Тьюринг соединил машину принятия решений $\mathscr{D}$ с Универсальной машиной $\mathscr{U}$ так, чтобы создать машину $\mathscr{K}$, которая анализирует стандартные описания. Преимущество такого подхода – в том, что мы можем «подключить» другую первичную машину, когда нужно вычислить синус другого угла.

Ни один из этих подходов не лишен проблем. Большая проблема – в том, что вход, как и выход, функции sin – вещественное число (по крайней мере, теоретически), а у вещественных чисел бесконечное число цифр. Напечатать бесконечное число цифр или записать эти цифры на ленту невозможно.

Даже если мы ограничимся хорошими, простыми углами с конечным числом десятичных разрядов, для вычисления функции sin нужны радианы. 180° – это π радиан, отсюда ясно, что простой угол в 10° на самом деле равен $\pi/18$ радиан – трансцендентному числу с бесконечным числом десятичных разрядов.

Итак, мы пришли к тому, что одна машина должна вычислить $\pi/18$, а вторая – синус этой величины, и обе машины фактически реализуются в одной «мета-машине». Вторая машина не может ждать окончания работы первой машины, чтобы начать свои вычисления! Две машины должны работать в тандеме, программный метод, иногда называемый *сцеплением* (*dovetailing*): как только машина, вычисляющая угол, выдает очередную цифру, машина, вычисляющая синус этого угла, должна принять ее и вычислить очередную цифру результата. Такое челночное (back-and-forth) взаимодействие двух машин продолжается бесконечно.

Теперь вас, наверное, не удивит то, что машины Тьюринга, переопределенные Стивеном Клини и Мартином Дэвисом, вычисляют только *теоретико-числовые* функции, то есть функции, область определения и область изменения которых ограничивается неотрицательными целыми числами. Оба автора кодируют ввод функции вертикальными штрихами или «галочками». Например, число 7 – это просто 8 вертикальных штрихов в восьми последовательных клетках ленты. (Число 0 требует одного штриха.)

Вообще говоря, нам не хотелось бы, чтобы вычисление функции тянулось вечно. Желательно, чтобы вычисление функции закончилось и можно было проверить ее результат. Исходя из этих соображений, переопределенные Клини и Дэвисом машины Тьюринга *останавливаются* по окончании вычисления. Очевидно, что машина, предназначенная для сложения или умножения неотрицательных целых чисел, не может работать вечно. В формулировке Клини и Дэвиса машины, которые не останавливаются, считаются *плохими* (*bad*). Определение того факта, что машина Тьюринга должным образом закончит свои вычисления и остановится, названо Дэвисом *проблемой останова*. Позже проблему останова стали тесно связывать с машинами Тьюринга, однако это понятие чуждо оригинальной статье Тьюринга.

Теперь, когда мы немного поразмышляли о проблемах, возникающих при создании машин Тьюринга, которые работают с вещественными функциями, мы готовы изучить подходы, которые предлагает Тьюринг в разделе 10 своей статьи.

Раздел 10 фактически продолжает раздел 9, в начале которого Тьюринг приводит три аргумента, касающихся вычислительных способностей его машин. Третий из них «Примеры больших классов вычислимых чисел» – тема раздела 10. В то же время Тьюринг несколько отклоняется от своей первоначальной темы – вычислимых чисел – и отправляется в мир вычислимых функций.

Раздел 10 – это та часть статьи Тьюринга, которая, видимо, меньше всего подвергалась анализу и зачастую слишком трудна для разбора. Она немногословна и местами туманна, поэтому у меня нет уверенности в том, что я всегда улавливал его аргументы абсолютно точно.

Наверное, неудивительно, что Тьюринг начинает с обсуждения «вычислимой функции целой переменной» и что «простейший» способ определить такую функцию требует, чтобы и область определения, и область изменения были неотрицательными целыми числами.

10. *Примеры больших классов вычислимых чисел.*

Будет полезно начать с определения вычислимой функции целой переменной, вычислимой переменной и т. д. Есть множество равнозначных способов определения вычислимой функции целочисленной переменной. Простейший, наверное, таков. Если γ – вычислимая последовательность, в которой 0 появляется бесконечно† часто, а n – целое число, то давайте определим $\xi(\gamma, n)$ так, чтобы она была номером символа 1 между n-м и $(n + 1)$-м символом 0 в γ. Тогда $\phi(n)$ вычислима, если для всех n и некоторой γ $\phi(n) = \xi(\gamma, n)$.

† Если машина M вычисляет γ, то задача, печатает ли M символ 0 бесконечно часто, того же свойства, что задача, свободна ли машина M от циклов.

Здесь нужен пример. Пусть наша функция ϕ (греческая буква фи) – что-то простое вроде

$$\phi(n) = 2n + 1$$

для неотрицательного целого числа n. Тогда

$$\phi(0) = 1$$
$$\phi(1) = 3$$
$$\phi(2) = 5$$
$$\phi(3) = 7$$
$$\ldots$$

Соответствующая этой функции последовательность γ (греческая буква гамма) такова:

0101110111111011111110...

Обратите внимание, что в ней между каждой очередной парой нулей – сначала одна единица, потом три, потом пять, семь и т. д., что соответствует значениям функции $\phi(n)$ для каждого очередного неотрицательного целого n.

Действительно ли γ – вычислимая последовательность? На мой взгляд, определенно вычислимая. Это значит, что $\phi(n)$ – вычислимая функция, а машина, которая вычисляет γ, работает бесконечно, вычисляя все значения функции.

Тьюринг подходит к этой вычислимой функции с другой стороны: он указывает на связь последовательности γ, которая содержит 0 бесконечно часто, с функцией $\xi(\gamma, n)$ (греческая буква кси), которая определяет количество единиц между каждой очередной парой нулей, и приравнивает $\phi(n)$ к $\xi(\gamma, n)$.

В таком случае любая машина, вычисляющая последовательность, в которой 0 появляется бесконечно часто, также вычисляет функцию положительных целых чисел, хотя, вообще говоря, мы не можем определить, какие машины действительно отвечают этому критерию.

Теперь Тьюринг предполагает связанный с функцией ϕ предикат, так что вычисление функции становится похожим на вычисление числа с опорой на логику, которое Тьюринг приводит в разделе 9 своей статьи (см. главу 12 этой книги).

Равносильное определение таково. Пусть $H(x, y)$ означает $\phi(x) = y$.

Давайте воспользуемся тем же примером. После замены независимой переменной функции становится такой:

$\phi(x) = 2x + 1$.

Можно определить $H(x, y)$, используя предикаты из предыдущей главы:

$(\exists z)$ $(Product(u'', x, z)$ & $Sum(z, u', y))$.

Эта формула истинна, если существует такое z, что истинно и то, что удвоенное x равно z, и то, что z плюс 1 равно y.

Тогда, если мы можем найти непротиворечивую (неопровержимую) аксиому $\mathfrak{A}_\phi$, такую, что $\mathfrak{A}_\phi \rightarrow P$,

$\mathfrak{U}_\phi$ должна быть конъюнкцией аксиом для предикатов, необходимых для определения H (в данном случае *Product* и *Sum*) и P, тогда очевидно, что импликация имеет силу.

и если для каждого целого числа n существует целое число N, такое, что

$$\mathfrak{U}_\phi \,\&\, F^{(N)} \rightarrow H(u^{(n)}, u^{(\phi(n))}),$$

Вы помните, что $F^{(N)}$ – это сокращение конъюнкции функций преемника. N должно быть не меньше (а то и больше) наибольшего из n и $\phi(n)$. В нашем примере $\phi(10) = 21$. Предикату H нужны числа $u' = 1$, $u'' = 2$ и $z = 20$. Поэтому для достаточности N оно должно быть не меньше 21, а в некоторых случаях и больше, чем n и $\phi(n)$.

и таким, что если $m \neq \phi(n)$, то для некоторого N'

$$\mathfrak{U}_\phi \,\&\, F^{(N')} \rightarrow (-H(u^{(n)}, u^{(m)}),$$

Правая круглая скобка в конце отсутствует. В итоге предикат H истинен, когда первый аргумент – n, а второй – $\phi(n)$, иначе он ложен. Для этой второй формулы N' тоже должно быть не меньше m, но вскоре мы увидим, что значения m, большие n, фактически не рассматриваются.

тогда можно сказать, что ϕ – вычислимая функция.

Здесь Тьюринг заканчивает свои рассуждения, ничего, по сути, не объясняя, как это должно работать. Это еще одна модификация машины $\mathcal{K}$, которая перечисляет все доказуемые формулы. Описанная Тьюрингом в разделе 9 машина перечисляет эти доказуемые формулы, печатая 1 или 0 для последовательных значений n, соответствующих значению истинности предиката G.

Для этой новой задачи нужна машина $\mathcal{K}_\gamma$, которая будет вычислять последовательность γ, описанную выше:

01011101111011111110...,

где каждый ряд единиц – это значение функции $\phi(n)$ для последовательных значений n. Главное отличие – в том, что эта машина перечисляет все доказуемые формулы не только для последовательных значений n, но и для произвольных значений n и m.

Для каждого нового значения n и m машина начинает работу с записи формул A и B (в терминологии предыдущей машины)

A $\mathfrak{U}_\phi$ & $F^{(N)} \to H(u^{(n)}, u^{(m)})$ **B** $\mathfrak{U}_\phi$ & $F^{(N')} \to (-H(u^{(n)}, u^{(m)}))$,

а затем при генерации всех доказуемых формул пытается сопоставить их с каждой из этих двух.

Машина начинает с n, равного нулю. Это – аргумент функции $\phi(n)$ и первый аргумент предиката $H(n, m)$. Для каждого нового значения n машина печатает 0 и устанавливает m в нуль. Этот m – возможный результат функции $\phi(n)$ и второй аргумент предиката $H(n, m)$.

Для каждого нового значения m машина начинает генерировать все доказуемые формулы. Если формула совпала с B, машина печатает 1, потому что это значение m не является результатом функции $\phi(n)$. Машина увеличивает m, печатает новые варианты A и B и снова начинает генерировать доказуемые формулы. Если формула совпала с A, то значение m – результат $\phi(n)$, и машина переходит к следующему значению n. Для следующего значения n машина начинает с печати нуля и установки m в нуль.

Таким образом, машина печатает последовательность γ, в которой каждый ряд единиц – это значение функции для возрастающих значений целочисленного аргумента.

Мы не можем определить общие вычислимые функции действительной переменной, так как нет никакого общего метода описания вещественного числа,

Это обсуждавшаяся ранее проблема кодирования вещественного числа на ленте для обращения к нему функции.

но мы можем определить вычислимую функцию вычислимой переменной.

Для этого необходимо, чтобы вычисление вычислимого числа и вычислимой функции шли в связке. Наверное, в простейшем случае после вычисления каждой новой цифры переменной вычислимая функция перехватывает её и печатает новую цифру результата. И вычислимая переменная, и вычислимая функция поддерживают один и тот же уровень точности по количеству значащих разрядов.

Пример такой функции у Тьюринга – тригонометрический тангенс. Тьюринг хочет вычислить тангенс для разных (фактически для всех) вычислимых чисел, но у него нет никакого критерия прекращения вычислений для одного числа и начала вычислений для следующего.

Если n – приемлемое число, то пусть γ_n – число, вычисленное $\mathcal{M}(n)$, и пусть

$$\alpha_n = \tan\left(\pi\left(\gamma_n - \tfrac{1}{2}\right)\right),$$

[255]

пока не окажется, что $\gamma_n = 0$ или $\gamma_n = 1$, для каждого из которых $\alpha_n = 0$.

Следующая сноска указывает, что это не единственно возможная функция, но для целей Тьюринга она достаточно проста. Поскольку машины Тьюринга вычисляют действительные числа от 0 до 1, число γ_n будет от 0 до 1 независимо от машины. Аргумент функции тангенса – величина угла от $-\pi/2$ до $\pi/2$ радиан или от -90° до 90°.

Тогда если n пробегает приемлемые числа, то α_n пробегает вычислимые числа†.

† Функция α_n может быть определена любым другим способом, чтобы пробегать вычислимые числа.

Тангенс значений от -90° до 90° меняется от $-\infty$ до $+\infty$, охватывая таким образом весь континуум действительных чисел. (Строго говоря, тангенсы -90° и 90° не определены, поэтому Тьюринг рассматривает эти случаи отдельно.) За несколькими исключениями значения функции тангенса будут трансцендентными числами.

И только после этого Тьюринг фактически предлагает определить машину, которая вычисляет тангенс угла. Подобно синусу, функция тангенса вычисляется с помощью бесконечного ряда

$$\tan(x) = x + \frac{x^3}{3} + \frac{x^5}{3\cdot 5} + \frac{x^7}{3\cdot 5\cdot 7} + \ldots,$$

где x меняется от $-\pi/2$ до $\pi/2$.

Теперь пусть $\phi(n)$ – вычислимая функция, которая, как может быть показано, такова, что для каждого приемлемого аргумента ее значение приемлемо‡. Тогда функция f, определенная как $f(\alpha_n) = \alpha_{\phi(n)}$, – вычислимая функция, и все вычислимые функции вычислимой переменной могут быть выражены таким же образом.

‡ Хотя невозможно найти общий процесс для установления приемлемости числа, часто можно показать, что определенные классы чисел приемлемы.

Функция $\phi(n)$ имеет область определения и диапазон Описательных номеров приемлемых машин. Возможно, мы ограничиваемся машинами определенного формата, достаточно простого, чтобы быть приемлемыми. Функция $\phi(n)$ меняет Описатель, по сути дела перепрограммируя машину так, что она вычисляет нечто иное. Например, $\phi(n)$ могла бы перепрограммировать машину так, чтобы она вместо вычисления x удваивала бы его значение и добавляла бы к нему 1, реализуя на самом деле функцию $2x + 1$.

Похожие определения могут быть даны вычислимым функциям нескольких переменных, функциям с вычислимыми значениями от целочисленной переменной и т. д.

Я сформулирую ряд теорем о вычислимости, но докажу только (ii) и теорему, подобную (iii).

Следующие десять теорем обозначены малыми римскими цифрами. Доказательства (ii) и (iii) занимают последние две страницы раздела 10 статьи.

(i) Вычислимая функция вычислимой функции целочисленной или вычислимой переменной вычислима.

Иными словами, мы можем помещать функции в стек. Можно начать с машины, вычисляющей число, затем применить другую машину, которая реализует функцию от этого числа, а потом применить третью машину, которая реализует еще одну функцию от результата первой функции. Как и машина тригонометрического тангенса, эти машины не могут ждать окончания предыдущего этапа вычислений. Машины должны работать в тандеме, передавая информацию от одной к другой, как только вычислена очередная цифра.

(ii) Любая функция целочисленной переменной, определяемая рекурсивно через вычислимые функции, вычислима. *То есть* если $\phi(m, n)$ вычислима и r – некоторое целое число, то $\eta(n)$ вычислима, где

$$\eta(0) = r,$$
$$\eta(n) = \phi(n, \eta(n-1)).$$

Внимание: греческая буква η слегка напоминает наклонную n. «Функция целочисленной переменной» – это η. «Вычислимая функ-

ция» – это $\phi(m, n)$. Для примера давайте предположим, что r равняется 1, а функция $\phi(m, n)$ определяется просто как

$$\phi(m, n) = m \cdot n,$$

которая, я уверен, вычислима. Давайте вычислим несколько значений η.

$$\eta(0) = r = 1$$
$$\eta(1) = \phi(1, \eta(0)) = \phi(1, 1) = 1 \cdot 1 = 1$$
$$\eta(2) = \phi(2, \eta(1)) = \phi(2, 1) = 2 \cdot 1 = 2$$
$$\eta(3) = \phi(3, \eta(2)) = \phi(3, 2) = 3 \cdot 2 = 6$$
$$\eta(4) = \phi(4, \eta(3)) = \phi(4, 6) = 4 \cdot 6 = 24$$
$$\eta(5) = \phi(5, \eta(4)) = \phi(5, 24) = 5 \cdot 24 = 120$$
. . .

Функция $\eta(n)$ – рекурсивное определение факториала. В конце этого раздела Тьюринг докажет, что целочисленная функция $\eta(n)$, определенная через вычислимую функцию $\phi(m, n)$, сама вычислима.

(iii) Если $\phi(m, n)$ – вычислимая функция двух целочисленных переменных, то $\phi(n, n)$ вычислимая функция n.

Это кажется тривиальным, но Тьюринг пользуется возможностью, чтобы сделать кое-что интересное. Доказательство завершает эту главу.

(iv) Если $\phi(n)$ – вычислимая функция, значение которой всегда 0 или 1, то последовательность, n-я цифра которой есть $\phi(n)$, вычислима.

Например, допустим, что функция *IsPrime*(n) возвращает 1, если n – простое число, и 0 – в противном случае. Тьюринг утверждает, что следующая последовательность вычислима:

00110101000101000101000010000010...

Я показал только первые 31 цифру последовательности, начинающейся с нуля. Номер цифры 1 соответствует простым числам 2, 3, 5, 7, 11, 13, 17, 19, 23 и 29.

Затем Тьюринг обращается к теореме Дедекинда. Вот формулировка этой теоремы из первой главы книги Дж. Х. Харди *Курс чистой математики*, где Тьюринг, наверное, впервые столкнулся с теоремой,

когда начал читать книгу в 1931 году при подготовке к неудачному поступлению в Кембридж.

Теорема Дедекинда. Если вещественные числа делятся на два класса L и R таким образом, что

(i) каждое число принадлежит одному или другому из двух классов,
(ii) каждый класс содержит, по крайней мере, одно число,
(iii) любой элемент L меньше любого элемента R,

то существует число α, которое обладает тем свойством, что все числа, меньшие него, принадлежат L, а все числа, большие него, – к R. Само число α может принадлежать любому классу[1].

На первый взгляд, теорема похожа на маленькую хитрость, но она раскрывает фундаментальное различие между рациональными и вещественными числами, и особенно то, что вещественные числа образуют континуум, тогда как рациональные – нет.

Представьте числовую ось, которая простирается от отрицательной бесконечности слева до положительной бесконечности справа. Если хотите, можете рассмотреть только отрезок этой числовой прямой. Разобьем ее на две части (сечение Дедекинда). Часть чисел – слева (L в теореме), а часть – справа (R).

Например, можно разбить прямую в числе 1,5 так, чтобы все числа из L были меньше 1,5, а все числа из R были больше 1,5. Что можно сказать о самом числе 1,5? На ваше усмотрение – отнести его к L или к R. Его можно отнести к L, и тогда оно стало бы наибольшим в L. В этом случае в R не было бы наименьшего числа. Иными словами,

---

[1] G. H. Hardy, *A Course of Pure Mathematics*, 10th edition (Cambridge University Press, 1952), 30. Теорема Дедекинда не охвачена первым изданием (1908) книги Харди, а появляется только в шестом издании (1933). У меня не было возможности сравнить эти издания. Тьюринг читал книгу Харди перед поступлением в Кембридж в 1931 году и, возможно, приобрел тогда пятое издание (1928). Книга Харди не единственная, где Тьюринг мог найти теорему Дедекинда. Она обсуждалась в главе 34 книги Bertrand Russell, *The Principles of Mathematics* (Cambridge University Press, 1903), в главе 7 книги Bertrand Russell, *Introduction to Mathematical Philosophy* (George Allen & Unwin, 1919, 1920), а также в первой главе книги E. W. Hobson, *The Theory of Functions of a Real Variable and the Theory of Fourier's Series*, которую Тьюринг упоминает в §8 своей статьи. Брошюра Дедекинда «*Stetigkeit und Irrationale Zahlen*», описывающая это понятие, была издана в 1872 году; первый английский перевод появился в книге Richard Dedekind, *Essays on the Theory of Numbers*, trans. Wooster W. Beman (Open Court Press, 1901; Dover, 1963) под заголовком «Continuity and Irrational Numbers». Перевод был полностью пересмотрен в книге под редакцией Уильяма Эвалда *From Kant to Hilbert: A Source Book in the Foundations of Mathematics* (Oxford University Press, 1996), Vol. II, 765–779.

в R не было бы числа, меньшего, чем все остальные. Или 1,5 можно отнести к R, тогда оно стало бы наименьшим в R, а в L не стало бы наибольшего числа.

Теперь разобьем прямую в числе $\sqrt{2}$ (квадратный корень из двух). Если числовая ось состоит только из *рациональных чисел*, то все числа из L меньше $\sqrt{2}$, а все числа из R больше $\sqrt{2}$. Поскольку $\sqrt{2}$ не рациональное число, оно не принадлежит ни L, ни R. Кроме того, в L нет наибольшего числа, а в R – наименьшего. Прямая имеет разрыв в точке $\sqrt{2}$.

Однако если числовая ось состоит из *вещественных чисел*, то $\sqrt{2}$ должно принадлежать либо L, либо R. Невозможно задать такое сечение вещественных чисел, чтобы точка разбиения не принадлежала ни L, ни R. Именно поэтому вещественные числа образуют континуум, а рациональные – нет.

Теорема Дедекинда не справедлива в ее обычном виде, если мы всюду заменим «вещественный» на «вычислимый».

Вычислимые числа не образуют континуум, потому что можно провести сечение Дедекинда на невычислимом числе. Возможно, это число – слишком сложное (то есть слишком случайное), чтобы иметь алгоритмическое определение, или оно имеет *определение* – например, число содержит по одной цифре из каждого вычислимого числа, – но вычислить его нельзя. Это сечение делит вычислимые числа на две части так, что в L нет наибольшего числа, а в R нет наименьшего.

Но она справедлива в следующем виде:

Обратите внимание на последующую римскую цифру: это – новая теорема Тьюринга (v). Логическая функция $G(\alpha)$, которую здесь вводит Тьюринг, истинна, если аргумент – вычислимое число – меньше (или, возможно, меньше или равен) некоторого постоянного числа ξ (греческая буква кси).

Функция $G(\alpha)$ делит вычислимые числа на два класса: L – вычислимые числа, для которых $G(\alpha)$ возвращает истину, и R – числа, для которых $G(\alpha)$ возвращает ложь. Для теоремы Дедекинда в формулировке Харди требование (i) выполняется, потому что $G(\alpha)$ или истинна, или ложна для каждого вычислимого числа.

(v) Если $G(\alpha)$ – логическая функция вычислимых чисел и

$$(a) \quad (\exists\alpha)\,(\exists\beta)\,\{G(\alpha)\ \&\ (-G(\beta))\},$$

Формула (*a*) Тьюринга равносильна требованию (ii) у Харди, что каждый класс содержит хотя бы один элемент.

$$(b) \quad G(\alpha)\ \&\ (-G(\beta)) \rightarrow (\alpha < \beta)$$

Формула (*b*) Тьюринга равносильна требованию (iii) у Харди. Если $G(\alpha)$ истинна, то α находится в L, а если $G(\beta)$ ложна, то β находится в R, и α меньше β.

и существует общий процесс определения истинности значения $G(\alpha)$, то существует вычислимое число ξ такое, что [256]

$$G(\alpha) \rightarrow \alpha \leq \xi,$$
$$-G(\alpha) \rightarrow \alpha \geq \xi.$$

«Общий процесс определения истинности значения $G(\alpha)$» крайне важен. Поскольку ξ – вычислимое число, в самом общем случае оно не может явно храниться где-то внутри определения $G(\alpha)$. Тем не менее машина могла бы вычислить это число, сжимая границы вокруг его значения. Фактически для вычисления ξ машина должна все время повышать точность проверяемых в $G(\alpha)$ значений.

Другими словами, теорема справедлива для любого сечения вычислимых чисел, то есть существует общий процесс определения, к какому классу принадлежит данное число.

В следующем предложении термин «последовательность вычислимых чисел» может слегка сбить с толку, потому что чуть ли не с самого начала статьи для обозначения генерируемых машиной цифр Тьюринг употреблял термин «вычислимая последовательность». «Последовательность», о которой он говорит здесь, – это упорядоченный набор вычислимых чисел.

Вследствие этого ограничения теоремы Дедекинда мы не можем сказать, что у вычислимой ограниченной возрастающей последовательности вычислимых чисел есть вычислимый предел. Это можно понять, рассматривая такую последовательность:

$$-1,\ -\frac{1}{2},\ -\frac{1}{4},\ -\frac{1}{8},\ -\frac{1}{16},\ \frac{1}{2},\ \ldots$$

Этот небольшой пассаж был предметом обсуждения на телеконференции sci.logic несколько лет назад. Архив обсуждений начина-

ется в http://sci.techarchive.net/Archive/sci.logic/2004-08/2244.html. Одно предположение состояло в том, что 1/2 в конце – это ошибка, а на самом деле тут должна быть –1/32. Данная поправка делает последовательность более привлекательной, но не может быть тем, что имел в виду Тьюринг, потому что теперь становится очевидным, что последовательность стремится к нулю, который, конечно же, является вычислимым пределом.

Более правдоподобная догадка – в том, что Тьюринг предлагает последовательность, которая как будто бы стремится к вычислимому пределу, но на самом деле – нет. Возможно, последовательность можно было даже ограничить –1 и 1, но и это не говорит нам о действительной вычислимости предела.

С другой стороны, (v) позволяет нам доказать

(vi) Если α и β вычислимы и $\alpha < \beta$ и $\phi(\alpha) < 0 < \phi(\beta)$, где $\phi(\alpha)$ – вычислимая возрастающая непрерывная функция, то существует единственное вычислимое число γ, удовлетворяющее условию $\alpha < \gamma < \beta$ и $\phi(\gamma) = 0$.

Тьюринг, видимо, подразумевает полиномиальную вычислимую функцию:

$\phi(x) = 5x^3 - 3x^2 - 7$.

Известно, что любой действительный многочлен нечетной степени (наибольший показатель степени) имеет не менее одного действительного корня, то есть значение x, для которого многочлен обращается в нуль.

Границы корня многочлена $\phi(x)$ можно сузить, заметив, что $\phi(1)$ равняется –5, а $\phi(2)$ равняется 21. Корень должен быть между 1 и 2.

Числа 1 и 2 – это α и β в утверждении Тьюринга; можно видеть, что $\alpha < \beta$ и $\phi(\alpha) < 0 < \phi(\beta)$. Поскольку функция непрерывна (то есть у нее нет разрывов), существует значение γ между α и β, для которого $\phi(\gamma)$ равняется нулю. Цель – вычислить его. Мы можем выбрать число на полпути между α и β, которое в данном случае равно 1,5, и найти $\phi(1{,}5)$, равное 3,125. Поскольку это значение больше нуля, пусть 1,5 будет новым β. Теперь мы знаем, что корень – где-то между 1 и 1,5. Теперь пробуем 1,25; $\phi(1{,}25)$ равно –1,921875, что меньше нуля, поэтому 1,25 – это новое α. Теперь мы знаем, что корень – между 1,25 и 1,5. Каждый шаг ограничивает корень все меньшим интервалом и, по сути, вычисляет его.

Грубо говоря, последовательность чисел *сходится*, если абсолютная величина разности между соседними числами последовательности становится все меньше и меньше. (Я говорю «грубо», потому что в самом начале эта разность может получиться немалой.) Любое вещественное число может быть представлено сходящейся последовательностью рациональных чисел. Проще всего это показать, используя последовательность из рациональных чисел вроде этой:

$a_0 = 3$
$a_1 = 3{,}1$
$a_2 = 3{,}14$
$a_3 = 3{,}141$
$a_4 = 3{,}1415$
$a_5 = 3{,}14159$
...

Все они – рациональные числа, хотя становятся все ближе и ближе к иррациональному числу, известному как число π. Последовательность сходится, потому что разность между соседними числами уменьшается. Разность между a_3 и a_4 равна 0,0005, а разность между a_4 и a_5 – 0,00009.

В математике эта разность часто обозначается малой буквой ε. Вы можете выбрать произвольное значение ε, настолько малое, насколько пожелаете, например 0,0001. Последовательность сходится, если для заданного ε существует такое число N, что для любых $n > N$ и $m > N$ абсолютная величина разности между числами меньше произвольно выбранного числа: $|a_n - a_m| < \varepsilon$. В приведенном выше примере для ε, равного 0,0001, $N = 3$, потому что $|a_4 - a_5| < 0{,}0001$, как и разность между любыми a_n и a_m, где n и m больше 3.

Тьюринг определяет вычислимую сходимость аналогично.

Вычислимая сходимость.

Будем говорить, что последовательность β_n вычислимых чисел *вычислимо сходится*, если существует вычислимая целочисленная функция $N(\varepsilon)$ вычислимой переменной ε, такая, что мы можем показать, что если $\varepsilon > 0$ и $n > N(\varepsilon)$ и $m > N(\varepsilon)$, то $|\beta_n - \beta_m| < \varepsilon$.

Числа последовательности должны быть вычислимыми, но Тьюринг требует также, чтобы ε было вычислимым и $N(\varepsilon)$ была вычислимой функцией.

Тогда мы можем показать, что

(vii) Степенной ряд, коэффициенты которого образуют вычислимую последовательность вычислимых чисел, вычислимо сходится во всех вычислимых точках внутри его интервала сходимости.

Степенной ряд – это бесконечная сумма вида:

$a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4 + ...$

Как я показал ранее, тригонометрическую функцию синуса можно представить в виде степенного ряда:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$

Его коэффициенты (то есть значения a_i) 1, 0, –1/3!, 0, 1/5!, 0, –1/7! и т. д., конечно же, образуют вычислимую последовательность. Некоторые степенные ряды сходятся только для x, равного нулю. Другие степенные ряды сходятся в некотором диапазоне значений x, называемом *интервалом сходимости*. Известно, что функция синуса сходится для всех x. Поскольку коэффициенты вычислимы, машина может определить, является ли ряд сходящимся.

(viii) Предел вычислимо сходящейся последовательности вычислим.

Возможна также последовательность функций, сходящаяся к определенной функции. Если такая сходимость имеет место для конкретного значения функции, то говорят, что это – *точечная* сходимость. Гораздо более строгая сходимость функций – *равномерная*, которая не зависит от значения функции.

И при очевидном определении «вычислимо равномерной сходимости»:

(ix) Предел вычислимо равномерной сходящейся вычислимой последовательности вычислимых функций – вычислимая функция. Отсюда

(x) Сумма степенного ряда, коэффициенты которого образуют вычислимую последовательность, – вычислимая функция в пределах ее интервала сходимости.

От этих теорем Тьюринг делает вывод, что все алгебраические числа вычислимы, как и некоторые популярные трансцендентные числа:

Из (viii) и $\pi = 4\,(1-\frac{1}{3}+\frac{1}{5}-...)$ мы заключаем, что π вычислимо.
Из $e = 1+1+\frac{1}{2!}+\frac{1}{3!}+...$ мы заключаем, что e вычислимо.
[257]
Из (vi) мы заключаем, что все действительные алгебраические числа вычислимы.

Напомню, что алгебраические числа – это корни полиномиальных уравнений.

Из (vi) и (x) мы заключаем, что действительные нули функций Бесселя вычислимы.

Функции Бесселя – это решения дифференциальных уравнений общего вида, а нули – это те значения, в которых функции обращаются в нуль. Последнее заключение относится также к тригонометрическим, логарифмическим, показательным функциям и множеству других менее известных функций.

Тьюринг пообещал доказать теорему (ii), которая утверждает, что любая функция целочисленной переменной, определенная рекурсивно через вычислимые функции, вычислима.

Доказательство (ii).

В начале раздела 10 Тьюринг определил предикат $H(x, y)$, который истинен, если $\phi(x) = y$, а затем показал, как машина может доказать формулы, включающие $H(u^{(n)}, u^{(\phi(n))})$ и $-H(u^{(n)}, u^{(m)})$, где $m \neq \phi(n)$. То доказательство установило вычислимость функции $\phi(x)$.

Настоящее доказательство основано на более раннем, только вместо функции $\phi(x)$ в нем присутствует функция $\eta(x)$, где

$\eta(0) = r$
$\eta(n) = \phi(n, \eta(n-1))$

Пусть $H(x, y)$ значит «$\eta(x) = y$» и пусть $K(x, y, z)$ значит «$\phi(x, y) = z$».

В примере с факториалом

$\phi(x, y) = x \cdot y$,

таким образом, $K(x, y, z)$ – это просто $Product(x, y, z)$.

$\mathfrak{A}_\phi$ – аксиома для $\phi(x, y)$.

Аксиома $\mathfrak{A}_\phi$ требует поддержки предиката $K(x, y, z)$. В примере с факториалом аксиома должна включать аксиомы для предикатов преемника, *Sum* и *Product*. Аксиома для $\eta(x)$ сложнее:

Мы принимаем за $\mathfrak{A}_\eta$

$$\mathfrak{A}_\phi \,\&\, P \,\&\, (F(x, y) \to G(x, y)) \,\&\, (G(x, y) \,\&\, G(y, z) \to G(x, z))$$
$$\,\&\, (F^{(r)} \to H(u, u^{(r)})) \,\&\, (F(v, w) \,\&\, H(v, x) \,\&\, K(w, x, z) \to H(w, z))$$
$$\,\&\, [H(w, z) \,\&\, G(z, t) \vee G(t, z) \to (-H(w, t))].$$

Это третье применение Тьюрингом предиката с именем G, каждый из которых определяется по-разному. Но этот – фундаментальный: его функция – «больше», и удивительно, что Тьюринг фактически избегает объяснений того, что она делает, пока не опубликует в *Трудах Лондонского математического общества* Исправления к своей статье[1]. (Эти Исправления представлены в главе 16 этой книги.) В этих Исправлениях Тьюринг говорит, что $G(x, y)$ нужно понимать как «x предшествует y» или $y > x$. Как обычно, предикат F – функция преемника. $\mathfrak{A}_\eta$ состоит из конъюнкции семи членов, начиная с $\mathfrak{A}_\phi$ и P. Третий член означает, что если y – преемник x, то y больше x, а последний член в первой строке устанавливает транзитивность функции G (если y больше x и z больше y, то z больше x).

Первый член во второй строке утверждает, что $H(u, u^{(r)})$ истинно, что означает $\eta(0) = r$. Второй член – это:

$$(F(v, w) \,\&\, H(v, x) \,\&\, K(w, x, z) \to H(w, z)).$$

Переводя с языка функций на язык предикатов, если $w = v + 1$, $\eta(v) = x$ и $\phi(w, x) = z$, то $\eta(w) = z$, или:

$$\eta(v + 1) = \phi(v + 1, \eta(v)),$$

что представляет собой иной способ задания общей формулы для $\eta(n)$ при $n > 0$.

Заключительный член $\mathfrak{A}_\eta$ таков:

$$[H(w, z) \,\&\, G(z, t) \vee G(t, z) \to (-H(w, t))].$$

---

[1] Alan Turing, «On Computable Numbers, with an Application to the Entscheidungsproblem. A Correction», *Proceedings of the London Mathematical Society*, 2nd Series, Vol. 43 (1937), 544–546.

Обратите внимание на пару предикатов G с перестановкой аргументов. Это единственная причина включения функции G в эту аксиому: либо один, либо другой из этих членов истинен, если z не равна t. В целом член утверждает, что если $H(w, z)$ истинен, то для всех значений t, не равных z, $H(w, t)$, не истинен, другими словами, $\eta(w) \neq t$.

Было бы гораздо лучше, если бы формула $\mathfrak{A}_\eta$ состояла из цепочки кванторов общности, но при их отсутствии они подразумеваются.

Я не буду приводить доказательство непротиворечивости $\mathfrak{A}_\eta$. Такое доказательство может быть проведено методами, применяемыми в книге Гильберта и Бернайса *Grundlagen der Mathematik* (Берлин, 1934), с. 209 и *далее*. Кроме того, непротиворечивость не имеет значения.

Это первый том книги, написанной в основном швейцарским математиком Паулем Бернайсом, которую он начал писать еще в Гёттингене. Будучи евреем, Бернайс в 1933 году был лишен профессуры в университете и в 1934 году уехал в Цюрих. Второй том *Grundlagen der Mathematik* (*Основания математики*) вышел в 1939 году. Книга высоко ценилась в то время, но никогда не переводилась на английский язык. Роль Бернайса в статье Тьюринга была иной. В опубликованном Исправлении к своей статье Тьюринг отмечает, что «обязан П. Бернайсу за указанные ошибки». Короткая статья Алонзо Чёрча, показавшая отсутствие решения для Entscheidungsproblem[1], тоже подверглась подобному критическому разбору и тоже сопровождалась исправлениями[2], содержащими сноску «автор обязан Паулю Бернайсу, указавшему на эту ошибку...».

Хотя с 1935 по 1936 год Бернайс находился в Институте перспективных исследований, он вернулся в Цюрих до прибытия в Принстон Тьюринга, и, по всей видимости, они никогда не встречались.

Страница, на которую указывает Тьюринг, – это начало раздела по теории чисел. Именно на этой странице Бернайс вводит предикат R, который является тем же самым, что и предикат G у Тьюринга. Аксиомы, которые приводит Бернайс для своего предиката «больше» (который я слегка подогнал к нотации Тьюринга), демонстрируют, насколько Тьюринг пренебрегает деталями:

---

[1] Alonzo Church, «A Note on the Entscheidungsproblem», *The Journal of Symbolic Logic*, Vol. 1, No. 1 (Mar. 1936), 40–41.

[2] Alonzo Church, «Correction to *A Note on the Entscheidungsproblem*», *The Journal of Symbolic Logic*, Vol. 1, No. 3 (Sept. 1936), 101–102.

$(x)-R(x, x)$
$(x)(y)(z)(R(x, y)\ \&\ R(y, z) \rightarrow R(x, z))$
$(x)(\exists y)R(x, y)$

Третья аксиома напоминает нам, что существует число, которое не больше любого другого числа. Обычно это число – 0 или 1 в зависимости от определения натурального ряда. На той же самой странице в *Grundlagen der Mathematik* приводятся аксиомы Бернайса для функции S преемника:

$(x)(\exists y)S(x, y)$
$(\exists x)(y)-S(x, y)$
$(x)(y)(r)(s)(S(x, r)\ \&\ S(y, r)\ \&\ S(s, x) \rightarrow S(s, y))$

Довольно странно, что Тьюринг указывает на страницу книги с определением предикатов, которыми он пользуется в своей статье, но при этом игнорирует их аксиомы.

Здесь Тьюринг применяет индуктивное доказательство – вид доказательства, который особенно удобен в теории чисел и других приложениях, где используются только натуральные числа. В индуктивном доказательстве формула (или что-то иное) сначала доказывается для нуля. Обычно это просто. Затем предполагается, что формула верна для n, и на основании этого предположения доказывается формула для $n + 1$. Здесь нет нужды доказывать формулу для всех n – истинность для $n + 1$ следует из истинности только для n. Раз формула была доказана сначала для 0, то она верна и для $0 + 1 = 1$, а раз она верна для 1, она верна и для $1 + 1 = 2$ и т. д.

Индуктивное доказательство у Тьюринга несколько иное. Он откладывает доказательство для случая нуля на потом и начинает со второго шага индукции: если предположить, что формула верна для $n - 1$, то она верна и для n. То есть сначала идет предположение.

Допустим, что для некоторых n, N, мы показали, что

$$\mathfrak{U}_{\eta}\ \&\ F^{(N)} \rightarrow H(u^{(n-1)}, u^{(\eta(n-1))}),$$

Следующая формула – прямое следствие аксиомы $\mathfrak{U}_{\phi}$, которая поддерживает логическую функцию K.

тогда для некоторого M

$$\mathfrak{U}_{\phi}\ \&\ F^{(M)} \rightarrow K(u^{(n)}, u^{(\eta(n-1))}, u^{(\eta(n))}),$$

Кроме того, так как $\mathfrak{A}_\eta \to \mathfrak{A}_\phi$ тривиально,

$\mathfrak{A}_\eta \,\&\, F^{(M)} \to K\,(u^{(n)}, u^{(\eta(n-1))}, u^{(\eta(n))})$.

Эта формула также останется истинной, если мы соединим ее конъюнкцией с парой членов в правой части, которые, как нам известно, должны быть истинными. Один из них – тривиальная функция преемника, которая подразумевается функцией $F^{(M)}$, а другой – первоначальное предположение о предикате H:

$$\mathfrak{A}_\eta \,\&\, F^{(M)} \to F(u^{(n-1)}, u^{(n)}) \,\&\, H(u^{(n-1)}, u^{(\eta(n-1))}) \,\&\, K(u^{(n)}, u^{(\eta(n-1))}, u^{(\eta(n))}),$$

Это уже начинает в чем-то напоминать аксиому $\mathfrak{A}_\eta$, и Тьюринг вскрывает это:

и

$$\mathfrak{A}_\eta \,\&\, F^{(M)} \to [F(u^{(n-1)}, u^{(n)}) \,\&\, H(u^{(n-1)}, u^{(\eta(n-1))}) \,\&\, K(u^{(n)}, u^{(\eta(n-1))}, u^{(\eta(n))}) \to H(u^{(n)}, u^{(\eta(n))})].$$

Справа от первой импликации – предпоследний член аксиомы $\mathfrak{A}_\eta$ со значениями $u^{(n)}$, $u^{(n-1)}$, $u^{(\eta(n))}$ и $u^{(\eta(n-1))}$ соответственно вместо w, v, z и x. Объединяя эти две формулы,

Следовательно, $\mathfrak{A}_\eta \,\&\, F^{(M)} \to H(u^{(n)}, u^{(\eta(n))})$.

Этим заканчивается индуктивная часть доказательства. Тьюринг еще не показал, что формула истинна для нуля. Теперь он делает это.

Также $\mathfrak{A}_\eta \,\&\, F^{(r)} \to H(u, u^{(\eta(0))})$.

Это лишь член аксиомы с $\eta(0)$ вместо r. Теперь мы знаем, что формула истинна для всех n.

Следовательно, для всякого n формула вида

$$\mathfrak{A}_\eta \,\&\, F^{(M)} \to H(u^{(n)}, u^{(\eta(n))})$$

доказуема.

Теперь необходимо показать, что для $m \neq \eta(n)$ – не $\eta(u)$, как у Тьюринга в следующем предложении, – из аксиом следует отрицание H.

Кроме того, если $M' \geq M$ и $M' \geq m$ и $m \neq \eta(u)$, то

$$\mathfrak{U}_\eta \;\&\; F^{(M')} \rightarrow G(u^{(\eta(n))}, u^{(m)}) \vee G(u^{(m)}, u^{(\eta(n))})$$

Таким образом, если $m \neq \eta(n)$, то m либо больше, либо меньше $\eta(n)$.

[258]

и

$$\mathfrak{U}_\eta \;\&\; F^{(M')} \rightarrow [\{G(u^{(\eta(n))}, u^{(m)} \vee G(u^{(m)}, u^{(\eta(n))} \;\&\; H(u^{(n)}, u^{(\eta(n))})\} \rightarrow (-H(u^{(n)}, u^{(m)}))].$$

Справа от первой импликации – повторное объявление последнего члена аксиомы $\mathfrak{U}_\eta$, несколько перестроенной и со значениями $u(n)$, $u(m)$ и $u(\eta(n))$ соответственно вместо w, t, и z. Объединяя эти две формулы,

Следовательно, $\mathfrak{U}_\eta \;\&\; F^{(M')} \rightarrow (-H(u^{(n)}, u^{(m)}))$.

Таким образом, условия нашего второго определения вычислимой функции выполнены.

Под «вторым определением» Тьюринг подразумевает то, что на странице 269 этой книги начинается со слов «Равносильное определение таково». Он установил, что может создать два утверждения следующего вида:

$\mathfrak{U}_\eta \;\&\; F^{(M)} \rightarrow H(u^{(n)}, u^{(\eta(n))})$
$\mathfrak{U}_\eta \;\&\; F^{(M')} \rightarrow (-H(u^{(n)}, u^{(m)}))$

Для каждого значения n и m одно или другое доказуемо.

Следовательно, η – вычислимая функция.

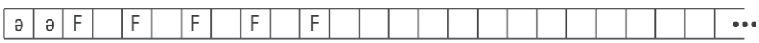
В следующем доказательстве таблицы машины Тьюринга появляются последний раз в этой статье (и в этой книге). Эта машина предназначена для доказательства теоремы Тьюринга (iii), которая утверждает, что если $\phi(m, n)$ – вычислимая функция двух целочисленных переменных, то $\phi(n, n)$ – вычислимая функция n.

Это доказательство иллюстрирует метод вычисления функции, область определения которой – натуральные числа, а область изменения – вещественные числа. Функция вычисляется для множества последовательных аргументов ($n = 0, 1, 2, ...$), но вычисление функции останавливается, когда вычислено n цифр.

Доказательство модификации (iii).

Пусть дана машина $\mathcal{N}$, которая, начав с ленты с символами əə, сопровождаемыми последовательностью из произвольного числа символов «*F*» в *F*-клетках, и *m*-конфигурации *b*, вычислит последовательность γ_n в зависимости от числа n символов «*F*».

Лента выглядит примерно так:



Эта машина интересна тем, что по описанию и принципу работы она больше других в этой статье похожа на переработанную машину Тьюринга, описанную Клини и Дэвисом. Эта машина читает неотрицательное целое число, записанное на ленте в виде ряда символов «F», а затем вычисляет функцию этого целого числа. Машины Клини и Дэвиса реализуют только теоретико-числовые функции, но Тьюринг в духе вечно работающих машин неявно подразумевает, что последовательность γ_n – это нечто более сложное, например кубический корень из *n*. Если машина начнет с чтения пяти символов «F» (как показано на ленте выше), она вычислит кубический корень из 5.

Если $\phi_n(m)$ – *m*-я цифра γ_n, то последовательность β, чья *n*-я цифра – $\phi_n(n)$, вычислима.

Например, функция $\phi_5(12)$ возвращает двенадцатую двоичную цифру кубического корня из 5. Тьюринг вводит новую вычислимую последовательность β, которая содержит одну цифру функции ϕ_n для каждого *n*. В моем примере первая цифра последовательности β – это первая цифра кубического корня из 1, вторая цифра – вторая цифра кубического корня из 2 и т. д.

Тьюринг начал с предположения, что такая машина (будь то машина кубического корня или какая-то другая) уже существует. Он собирается модифицировать эту машину частично путем изменения некоторых из существующих команд и частично путем добавления некоторых новых команд, чтобы получить несколько иную машину.

Новая машина должна работать с переменным числом символов «F»: сначала ни одного символа, затем один, затем два символа «F» и т. д. Вычислив *n* цифр (где *n* – число символов «F» плюс 1), она,

в сущности, должна прекратить вычисление, а затем снова начать его с дополнительным символом «F», управляющим ее действиями. Эта новая машина *не* будет последовательно печатать свои цифры 0 и 1 в *F*-клетках. Они будут печататься слева направо, но относиться к другому выводу машины.

Тьюрингу нужна его первоначальная машина в стандартном виде, чтобы ее можно было легко изменить.

Мы предполагаем, что таблица для $\mathcal{N}$ записывается таким образом, что в каждой строке в столбце операции оказывается только одна операция.

Для этих модификаций Тьюрингу нужно ввести некоторые новые символы, включая прописные греческие буквы Ξ (кси) и Θ (тета). Ему также нужно заменить три символа всюду, где они появляются в таблице этой машины.

Мы также предполагаем, что Ξ, Θ, $\bar{0}$ и $\bar{1}$ не появляются в таблице, и мы всюду заменяем ə на Θ, 0 на $\bar{0}$ и 1 на $\bar{1}$.

Когда Тьюринг говорит «всюду заменяем ə на Θ», он не имеет в виду ленту. Лента все еще должна начинаться с ää. Он имеет в виду, что любая строка таблицы, которая читает ə, должна быть изменена так, чтобы читать Θ.

Этого Тьюринг здесь не говорит, но модификация машины также требует, чтобы она не использовала символы *h* и *k* и что конфигурации $\mathfrak{c}$, $\mathfrak{u}$, $\mathfrak{u}_1$, $\mathfrak{u}_2$, $\mathfrak{u}_3$, $\mathfrak{v}$, $\mathfrak{v}_1$, $\mathfrak{v}_2$, $\mathfrak{v}_3$ доступны для новых определений.

Затем производятся дальнейшие подстановки. Каждую строку вида

$\mathfrak{A}$	α	$P\bar{0}$	$\mathfrak{B}$

мы заменяем на

$\mathfrak{A}$	α	$P\bar{0}$	$\mathfrak{re}(\mathfrak{B}, \mathfrak{u}, h, k)$

Заменяемая строка изначально печатала 0, а не $\bar{0}$. Конфигурации $\mathfrak{A}$ и $\mathfrak{B}$ и текущий символ α – здесь лишь метки-заполнители. Можно переименовать функцию $\mathfrak{re}$ в «replace»; здесь $\mathfrak{re}$ заменяет на ленте первый символ *h* на *k*, а затем переходит в конфигурацию $\mathfrak{B}$. Если она не находит *h*, то переходит в конфигурацию $\mathfrak{u}$.

Как определено в разделе 4, функция $\mathfrak{re}$ использует функцию $\mathfrak{f}$, которая использует ограничитель $\mathfrak{d}$. Эта функция *не* должна меняться, чтобы вместо $\mathfrak{d}$ не найти ограничитель Θ.

Эта новая машина, по существу, перехватывает операцию исходной машины каждый раз, когда та печатает цифру. Когда машина начинает работать с новым числом символов «F», число символов h на ленте будет на один больше, чем число символов «F». Таким образом, машина, прежде чем перейти в конфигурацию $\mathfrak{u}$, печатает h раз $\bar{0}$, а не 0.

а каждую строку вида

$\mathfrak{A}$ α $P\bar{1}$ $\mathfrak{B}$

на

$\mathfrak{A}$ α $P\bar{1}$ $\mathfrak{re}(\mathfrak{B}, \mathfrak{v}, h, k)$

В конфигурациях $\mathfrak{u}$ и $\mathfrak{v}$ должны печататься просто символы 0 и 1 без надчеркивания, после чего лента подготавливается к следующему числу символов «F», а машина фактически перезапускается. Тьюринг покажет только конфигурацию $\mathfrak{u}$, а $\mathfrak{v}$ очень похожа на нее.

и добавим в таблицу следующие строки:

$\mathfrak{u}$		$\mathfrak{pe}(\mathfrak{u}_1, 0)$
$\mathfrak{u}_1$	$R, Pk, R, P\Theta, R, P\Theta$	$\mathfrak{u}_2$
$\mathfrak{u}_2$		$\mathfrak{re}(\mathfrak{u}_3, \mathfrak{u}_3, k, h)$
$\mathfrak{u}_3$		$\mathfrak{pe}(\mathfrak{u}_2, F)$

После того как в конфигурации $\mathfrak{u}$ печатается обычный 0, в конфигурации $\mathfrak{u}_1$ печатаются k и два символа Θ, которые представляют собой ограничитель для новой машины. В конфигурации $\mathfrak{u}_2$ и $\mathfrak{u}_3$ каждый k замещает h. (Напомню, что h замещает k в предыдущем замещении.) Для каждого k, замененного на h, в конце печатается еще и F. В таблице Тьюринга есть бесконечный цикл, потому что $\mathfrak{u}_2$ всегда переходит в $\mathfrak{u}_3$, а $\mathfrak{u}_3$ всегда переходит в $\mathfrak{u}_2$. Функция замены должна быть такой:

$\mathfrak{re}(\mathfrak{u}_3, \mathfrak{b}, k, h)$.

Если больше нет символов k для замены на h, машина должна стартовать в конфигурации $\mathfrak{b}$ – начальной конфигурации исходной машины.

Конфигурация $\mathfrak{v}$ подобна $\mathfrak{u}$, за тем лишь исключением, что в $\mathfrak{v}$ печатается 1, а не 0.

а также подобные строки с v вместо u и 1 вместо 0 вместе со следующей строкой

𝔠 *R*, *P*Ξ, *R*, *Ph* 𝔟.

Еще одна ошибка, к сожалению. Здесь на самом деле должен печататься не *h*, а новый ограничитель:

𝔠 *R*, *P*Ξ, *R*, *P*Θ, *R*, *P*Θ

Тогда мы имеем таблицу для машины $\mathcal{N}'$, которая вычисляет β. Ее исходная *m*-конфигурация – 𝔠, а исходный прочитанный символ – второй 𝔞.

Давайте посмотрим, работает ли это на самом деле. У нас есть машина $\mathcal{N}'$ – модифицированная версия $\mathcal{N}$, которая вычисляет кубический корень числа, состоящего, однако, из многих символов «F», которые он считывает с ленты. Машина $\mathcal{N}'$ начинает работу с лентой лишь с двумя 𝔞𝔞 в конфигурации 𝔠:

В конфигурации 𝔠 печатаются Ξ и два Θ:

Ξ фактически нигде не используется. Теперь машина переходит в конфигурацию 𝔟 и продолжает работать обычным образом, но использует ограничитель ΘΘ вместо 𝔞𝔞. Ни одного символа «F» для чтения с ленты нет, поэтому она вычисляет кубический корень из нуля. Когда приходит время для печати первого $\bar{0}$, она также пытается заменить первый *h* на *k*. Символов *h* нет, поэтому машина переходит в конфигурацию 𝔲.

В этой конфигурации на ленте печатается обычный 0, сопровождаемый *k* и двумя Θ:

Конфигурации $\mathfrak{u}_2$ и $\mathfrak{u}_3$ образуют небольшой цикл для замены каждого *k* на *h* и печати *F*:

Символов *k* для замены на *h* больше нет, поэтому теперь машина переходит в *m*-конфигурацию 𝔟, и, в сущности, стартует заново для вычисления кубического корня из 1. Первая цифра – 1, поэтому машина печатает $\bar{1}$ и заменяет *h* на *k*. И продолжает дальше. Следующая цифра – 0, поэтому печатается $\bar{0}$. Символов *h* больше нет, поэтому машина снова переходит в конфигурацию 𝔲, чтобы напечатать обычный 0, сопровождаемый *k* и двумя Θ:

Теперь каждый *k* заменен на *h* и напечатан F:

Здесь есть еще одна ошибка, но я не стал ее исправлять. Необходимо, чтобы символы «F» находились в *F*-клетках так, чтобы между ними были промежутки. Несмотря на это, раз символов *k* для замены на *h* больше нет, машина переходит в *m*-конфигурацию 𝔟, чтобы начать вычисление кубического корня из 2.

Закончив раздел 10, Тьюринг убежден, что определил модель вычислений, которая формально охватывает все, что может потребоваться алгоритму. Теперь он готов показать, что Entscheidungsproblem не может иметь решения.

Глава 14

Главное доказательство

Некоторые математические доказательства – прямые; иным нужно зайти с черного хода. Эта вторая категория, конечно, включает доказательства *от противного* (*сведение к абсурду*), которые начинаются с предположения, обратного тому, что требуется доказать, а затем показывают, что исходное предположение приводит к противоречию.

Еще есть доказательства, которые не стремятся проникнуть через парадную дверь *или* черный ход. Наоборот, кажется, что такие доказательства вообще уклоняются от темы, выстраивая сложную конструкцию, которая временами кажется и таинственной, и не строгой. И когда цель этого занятия становится совершенно непонятной и вы уже теряете всякую надежду увидеть решение, ловкий математик вываливается из дымохода (разве что без сажи на ковре) и бодро восклицает «Оба-на!» – а доказательство-то уже закончено.

В определенном смысле заключительный раздел статьи Тьюринга – самый важный, потому что именно здесь он показывает, что «Entscheidungsproblem не может быть решена». Для того времени это было важным выводом, но построенная Тьюрингом для получения этого результата конструкция – воображаемое устройство, известное теперь как машина Тьюринга, – в конечном счете станет интересней и плодотворней, нежели само доказательство, которое должно теперь стать предметом нашего внимания.

Тьюринг положил начало этому доказательству в разделе 8. Тогда это казалось не слишком важным, но он тщательно доказал, что нельзя спроектировать машину, реализующую общий конечный процесс определения того, что другая машина Тьюринга когда-либо напечатает цифру 0. Два промежуточных раздела (9 и 10) служили для того, чтобы обосновать, что понятие машинной вычислимости по Тьюрингу равносильно нашему общепринятому пониманию вычислимости.

В разделе 11 Тьюринг показывает, как может быть выражена функциональность вычислительной машины на языке логики предикатов первого порядка. Затем он создает в этой логике формулу, которая доказуема тогда и только тогда, когда машина когда-либо печатает цифру 0. Если эта формула разрешима, то есть если можно установить ее доказуемость, то мы бы имели общий процесс определения того, что машина когда-либо напечатает 0, а мы уже знаем, что его не может быть.

[259]

11. *Применение к Entscheidungsproblem*.

Результаты §8 имеют ряд важных применений. В частности, они могут быть применены для того, чтобы показать, что Entscheidungsproblem Гильберта не может иметь решения. Пока я ограничусь доказательством этой частной теоремы. За постановкой этой проблемы я должен отослать читателя к книге Гильберта и Аккермана *Grundz ge der Theoretischen Logik* (Берлин, 1931), глава 3.

На самом деле книга вышла в 1928 году. Глава 3 – около трети книги из четырех глав на 120 страницах – называется «Der engere Funktionenkalkül» или «Ограниченное функциональное исчисление», которое известно нам сегодня как логика предикатов первого порядка. Авторы утверждают:

> *Das Entscheidungsproblem ist gelöst, wenn man ein Verfahren kennt, das bei einem vorgelegten logischen Ausdruck durch endlich viele Operationen die Entscheidung über die Allgemeingültigkeit bzw. Erfüllbarkeit erlaubt… [Das] Entscheidungsproblem muβ als das Hauptproblem der mathematischen Logik bezeichnet werden*[1].
>
> Проблема разрешимости решается, если нам известна процедура с конечным числом операций, устанавливающая справедливость или выполнимость любого заданного выражения… Проблема разрешимости должна считаться главной проблемой математической логики.

Использование Гильбертом и Аккерманом слов *справедливость* и *выполнимость* указывает на так называемую семантическую формулировку проблемы разрешимости. Спустя 25 лет Вильгельм Аккерман продолжил исследование Entscheidungsproblem с семантической точки зрения в своей небольшой книжке *Разрешимые случаи проблемы разрешимости* (North-Holland Publishing Company, 1954).

---

[1] David Hilbert and Wilhelm Ackermann, *Grundz ge der Theoretischen Logik* (Springer, 1928), 73, 77.

Ссылаясь на гильбертово ограниченное функциональное исчисление **K** (видимо, от *Kalkül*), Тьюринг для описания решения задачи пользуется несколько иным языком.

> Поэтому я предполагаю показать, что не может существовать общего процесса для определения доказуемости заданной формулы $\mathfrak{A}$ функционального исчисления **K**, *то есть* не может существовать машины, которая, отвечая любой из таких формул $\mathfrak{A}$, говорит в итоге, доказуема ли эта формула.

Используя слово *доказуемость* вместо *справедливость* или *выполнимость*, Тьюринг показывает, что он подходит к проблеме разрешимости с синтаксической точки зрения. Синтаксический подход к логике связан с системой аксиом и правил вывода. Считается, что формула доказуема (а формула – это теорема), если формула есть аксиома или же она выводится из аксиом путем применения правил вывода. Равносильность семантического и синтаксического подходов к логике первого порядка была установлена в 1930 году теоремой Гёделя о полноте.

Опираясь на рассуждения из разделов 9 и 10, Тьюринг получил право утверждать, что если для реализации общей разрешающей процедуры невозможно создать машину, то не существует и общей разрешающей процедуры, которая могла быть осуществлена человеком.

Тогда, в 1936 году, читатели статьи Тьюринга, не искушенные в тонкостях полноты, неполноты и разрешимости, могли бы прийти в замешательство, соотнося доказательство Гёделя о неполноте – в статье, чей заголовок на самом деле означал «unentscheidbare Sätze» или «неразрешимые утверждения», – и доказательство Тьюринга. Действительно, на первой же странице своей статьи Тьюринг пишет: «получены результаты, которые внешне похожи на таковые у Гёделя» (стр. 88 этой книги). Ему нужно развить эту тему несколько дальше.

> Наверное, стоит отметить: то, что я буду доказывать, совершенно отличается от известных результатов Гёделя†. Гёдель показал, что (в формализме Principia Mathematica) существуют такие утверждения $\mathfrak{A}$, что ни $\mathfrak{A}$, ни $-\mathfrak{A}$ не доказуемы. Как следствие этого показывается, что в рамках такого формализма невозможно доказать непротиворечивость Principia Mathematica (или **K**). С другой стороны, я покажу, что не существует общего метода, который говорит, доказуема ли в **K** данная формула $\mathfrak{A}$, или, что сводится к тому же,

непротиворечива ли система, включающая в себя **K**, в которую добавлена дополнительная аксиома $-\mathfrak{A}$.

† *Там же.*

Теорема Гёделя и теорема Тьюринга подходят к разрешимости с разных сторон. Теорема Гёделя показывает наличие утверждений, которые не могут быть ни доказаны, ни опровергнуты; эти утверждения, как говорится, неразрешимы.

«Общий метод», о котором говорит Тьюринг, – это разрешающая процедура или алгоритм, который анализирует любую произвольную формулу и устанавливает, доказуема она или недоказуема. Тьюринг докажет, что такой общей разрешающей процедуры не может быть.

Даже при наличии неразрешимых утверждений разрешающая процедура, очевидно, все еще могла бы существовать. Анализируя неразрешимое, по Гёделю, утверждение, нужно установить недоказуемость и самого утверждения, и его отрицания.

Если бы было доказано отрицание того, что показал Гёдель, то есть если для каждого $\mathfrak{A}$ доказуемо либо $\mathfrak{A}$, либо $-\mathfrak{A}$, то мы имели бы непосредственное решение Entscheidungsproblem. Но мы можем изобрести машину $\mathcal{K}$, которая последовательно докажет все доказуемые формулы. Рано или поздно $\mathcal{K}$ приведет либо к $\mathfrak{A}$, либо к $-\mathfrak{A}$. Если она приводит к $\mathfrak{A}$, то мы знаем, что $\mathfrak{A}$ доказуема. Если она приводит к $-\mathfrak{A}$, то, так как **K** непротиворечива (Гильберт и Аккерман, с. 65), мы знаем, что $\mathfrak{A}$ недоказуема.

Что ж, хорошо, но ведь очевидно, что мастерски исполненная машина Тьюринга $\mathcal{K}$ (это ее последнее появление в этой статье) – это совсем *не то*, что имел в виду Гильберт, формулируя Entscheidungsproblem. Независимо от того, какой – «механической» или «машиноподобной» – должна быть гипотетическая разрешающая процедура, ее тем не менее нужно рассматривать как нечто, посильное человеку, а не компьютеру, требующему тысяч лет работы и всех на свете ресурсов памяти.

Результат Гёделя не дал оснований для общей разрешающей процедуры. И в доказательстве Тьюринга все еще была необходимость.

Из-за отсутствия целых чисел в **K** доказательства получаются несколько длинными. Но основные идеи весьма просты.

Для каждой вычислительной машины $\mathcal{M}$ мы создаем формулу Un($\mathcal{M}$) и показываем, что если существует общий метод определения доказуемости Un($\mathcal{M}$), то существует и общий метод определения, печатает ли $\mathcal{M}$ когда-либо 0.

Раскрывает ли Тьюринг изюминку, называя формулу Un (от Undecidable – Неразрешимый)? Формула Un($\mathcal{M}$) играет роль контрпримера – формулы, которую ни одна общая разрешающая процедура не может успешно проанализировать.

Как вы помните, машина состоит из последовательности конфигураций, которым сопоставлены операции. С того момента, как Тьюринг представил свою Универсальную машину, для обозначения этих ее элементов он использует слово *команды* (*instructions*).

Тьюринг должен представить эту вычислительную машину на языке логики первого порядка. Каждая команда будет преобразована в формулу, которая указывает, как команда влияет на полную конфигурацию машины. Если помните, полная конфигурация – это снимок ленты после каждого движения машины. Полная конфигурация включает также следующую *m*-конфигурацию машины и следующий прочитанный символ.

Сначала Тьюринг представляет несколько логических функций (известных в современной терминологии как предикаты). Как все логические функции, они имеют значения истины или лжи. Во всех случаях аргументы этих функций – неотрицательные целые числа, обозначающие клетки ленты и полные конфигурации. Предполагается, что клетки ленты нумеруются с нуля, из чего следует, что лента имеет начало и бесконечна только в одном направлении.

Предполагается также, что полные конфигурации последовательно нумеруются. Пара чисел (x, y) задает определенную клетку y в полной конфигурации x. В своих примерах я буду использовать конкретные номера, хотя у Тьюринга нет ни примеров, ни номеров.

Давайте вспомним пример I машины Тьюринга, которая поочередно печатала в каждой второй клетке цифры 0 и 1. Вот первые 10 полных конфигураций. Я пронумеровал эти полные конфигурации числами в крайнем левом столбце. (Заголовок «ПК x» значит «Полная Конфигурация x».) Числами вверху я пронумеровал клетки ленты. Я также заменил оригинальные буквенные обозначения конфигураций у Тьюринга на буквы q с индексами.

КЛЕТКА *Y*

ПК *x*	0	1	2	3	4	5	6	7	...
0	q_1								
1	0	q_2							
2	0		q_3						
3	0		1	q_4					
4	0		1		q_1				
5	0		1		0	q_2			
6	0		1		0		q_3		
7	0		1		0		1	q_4	
8	0		1		0		1		
9	0		1		0		1		
...	0		1		0		1		

Эта машина заносит цифры только в каждую вторую клетку так, что числа оказываются только в клетках с четными номерами. Появляющиеся в клетке *m*-конфигурации указывают на то, что эта клетка просматривается в этой полной конфигурации.

Многие из логических функций Тьюринга имеют индексы, которые являются частью имени функции. Эти логические функции определяются заранее, но вы вскоре увидите, как Тьюринг использует их для описания всей машины.

Смысл используемых логических функций состоит в следующем:

$R_{S_l}(x, y)$ нужно понимать как «в полной конфигурации x (для $\mathcal{M}$) в клетке y находится символ S».

У последней перед заключительной кавычкой буквы S опущен индекс l. Как вы помните, символ S_0 – это пробел, S_1 – это 0, а S_2 – это 1. Для машины примера I $R_{S_2}(5, 2)$ истинна, потому что цифра 1 появляется в полной конфигурации 5 в клетке 2, но $R_{S_2}(5, 6)$ ложна, потому что цифра 1 не появляется в полной конфигурации 5 в клетке 6.

[260]

$I(x, y)$ нужно понимать как «в полной конфигурации x просматривается клетка y».

Для машины примера I функция $I(6, 6)$ истинна, потому что в полной конфигурации 6 следующая просматриваемая клетка – 6, но $I(6, y)$ ложна для любой другой клетки y.

$K_{q_m}(x)$ нужно понимать как «в полной конфигурации x m-конфигурация есть q_m».

В конце этого предложения отсутствует кавычка. Для машины примера I функция $K_{q_2}(5)$ истинна, но $K_{q_3}(5)$ и $K_{q_2}(7)$ ложны.

$F(x, y)$ нужно понимать как «y – непосредственный преемник x».

Или в обычной системе обозначений арифметики $y = x + 1$. Используя представленную ранее Тьюрингом систему обозначений для натуральных чисел, предикат $F(u''', u'''')$ истинен, но при надлежащей аксиоматизации $F(u''', u'''''')$ должен быть ложным.

Пока что Тьюринг создал только логические функции, которые описывают полные конфигурации машины в действии. Он не сопоставил им таблицы, которые описывают фактические машины. Стандартная машина содержит только одну команду печати и одну команду движения головки на строку таблицы. Каждая строка таблицы состоит из m-конфигурации q_i, просмотренного символа S_j, напечатанного символа S_k (который может совпадать с S_j), движения головки (влево, вправо или никуда) и новой m-конфигурации q_l.

Затем Тьюринг дает определение тому, что он называет Inst (от «instruction» – «команда»). Это не логическая функция, а сокращение для выражения, построенного из только что приведенных логических функций. Каждое выражение Inst соответствует строке машинной таблицы. Выражение описывает, как эти отдельные комбинации m-конфигураций, символов и движений головки влияют на полную конфигурацию:

Inst$\{q_i\, S_j\, S_k\, L\, q_l\}$ должно быть сокращением для

$$(x,y,x',y')\{(R_{S_j}(x,y) \,\&\, I(x,y) \,\&\, K_{q_i}(x) \,\&\, F(x,x') \,\&\, F(y',y)) \rightarrow (I(x',y') \,\&\, R_{S_k}(x',y) \,\&\, K_{q_l}(x') \,\&\, (z)[F(y',z) \vee (R_{S_j}(x,z) \rightarrow R_{S_k}(x',z))])\}.$$

Это – одно из трех возможных выражений Inst для случая движения головки влево, на что указывает буква L в списке аргументов Inst.

Начальная четверка (x, y, x', y') – это сокращение для четырех кванторов общности $(x)(y)(x')(y')$. Связанные переменные x и x' – две последовательные полные конфигурации; в конце первой строки вы видите предикат F (первый из двух), означающий, что x' – преемник x. Связанные переменные y и y' – это две соседние клетки. Для команды, в которой головка движется влево, y' равна y минус 1, или y – преемник y', на что указывает второй предикат F в первой строке.

Остальные три предиката в первой строке задают условия для этой команды. Полная конфигурация – x, просмотренная клетка – y, просмотренный символ – S_j и конфигурация – q_i.

Вторая строка начинается с операции импликации, относящейся к остатку выражения. Это предикаты, описывающие полную конфигурацию – результат этой команды. В следующей полной конфигурации x' просматривается клетка y'; в клетке y теперь находится символ S_k, а новая m-конфигурация – q_l.

Inst заканчивается выражением в третьей строке, которое *должно* означать, что все остальные клетки, кроме клетки y, остаются без изменений. Эти остальные клетки обозначаются связанной переменной z. Либо z – преемник y' (и в этом случае она равна y и является адресом операции печати), либо... и здесь оставшаяся часть формулы Тьюринга неверна. Все, что она говорит, – что во всех остальных клетках вместо S_j будет S_k, что бессмысленно.

Усовершенствованная версия выражения Inst – в исправлениях, которые Тьюринг привел в следующей статье, опубликованной в *Трудах Лондонского математического общества,* спустя примерно год после публикации оригинала. Это исправление (страница 349 этой книги) тоже не совсем правильное, и вот какой должна быть последняя строка:

$$\&(z)[F(y',z)\mathrm{v}([R_{S_0}(x,z)\rightarrow R_{S_0}(x',z)]\&[R_{S_1}(x,z)\rightarrow R_{S_1}(x',z)]\\ \&\ldots\&[R_{S_M}(x,z)\rightarrow R_{S_M}(x',z)]))]\}.$$

Индексы S_0, S_1, ..., S_M в функции R – это все символы, которые может печатать $\mathcal{M}$. Иными словами, для всех клеток z либо каждая из них – преемник y' (это значит, что z – измененная клетка y), либо символ в этой клетке остается без изменений от полной конфигурации x до полной конфигурации x'.

Выражение Inst, которое только что привел Тьюринг, – для головки, движущейся влево.

Inst $\{q_i\, S_j\, S_k\, R\, q_l\}$ и Inst $\{q_i\, S_j\, S_k\, N\, q_l\}$

должны быть сокращениями для двух других подобных выражений.

Для разнообразия (и удобства ссылок в дальнейшем) следующее полное правильное выражение для команды с движением головки вправо показано в рамке:

Inst $\{q_i\, S_j\, S_k\, R\, q_l\}$ – сокращение для:

$$(x,y,x',y')\{(R_S(x,y)\,\&\,I(x,y)\,\&\,K_{q_i}(x)\,\&\,F(x,x')\,\&\,F(y,y'))$$
$$\rightarrow (I(x',y')\,\&\,R_{S_k}(x',y)\,\&\,K_{q_l}(x')$$
$$\&\,(z)[F(z,y')\vee([R_{S_0}(x,z)\rightarrow R_{S_0}(x',z)]\,\&$$
$$[R_{S_1}(x,z)\rightarrow R_{S_1}(x',z)]\,\&$$
$$\ldots\,\&$$
$$[R_{S_m}(x,z)\rightarrow R_{S_m}(x',z)])]\}$$

Это выражение в основном такое же, как для команды движения головки влево, за тем исключением, что вместо $F(y', y)$ стоит $F(y, y')$, означающая, что новая просматриваемая клетка – справа от последней просмотренной, а в последней части выражения вместо $F(y', z)$ стоит $F(z, y')$, означающая, что z равна y.

Для машины примера I индекс m в функции R_{S_m} имеет максимальное значение 2, потому что машина печатает только символы S_0 (пробел), S_1 (нуль) и S_2 (один).

Давайте приведем описание $\mathcal{M}$ к первой стандартной форме из §6. Это описание состоит из множества выражений вида «$q_i\, S_j\, S_k\, L\, q_l$» (или с R или с N вместо L).

На самом деле Тьюринг описывает эту стандартную форму в разделе 5. Пример I машины появляется на странице 241 его статьи (страница 161 этой книги) примерно в таком виде:

$q_1 S_0 S_1 R q_2$; $q_2 S_0 S_1 R q_3$; $q_3 S_0 S_2 R q_4$; $q_4 S_0 S_0 R q_1$;

Каждая из этих четырех пятерок становится выражением Inst.

Давайте построим все выражения вида Inst $\{q_i S_j S_k L q_l\}$ и возьмем их логическую сумму. Назовем ее Des($\mathcal{M}$).

То есть ***Des**cription* (*Описание*) машины $\mathcal{M}$. Термин «логическая сумма» – несколько сомнительный, поэтому в Исправлениях к статье (страница 350 этой книги) Тьюринг заменяет его словом «конъюнкция». Иными словами, чтобы представить всю машину на языке логики первого порядка, все члены Inst нужно соединить значком &.

Des($\mathcal{M}$) для машины из примера I – это сокращение для:

$$\text{Inst}\{q_1S_0S_1Rq_2\}\ \&\ \text{Inst}\{q_2S_0S_1Rq_3\}\ \&$$
$$\text{Inst}\{q_3S_0S_2Rq_4\}\ \&\ \text{Inst}\{q_4S_0S_0Rq_1\}$$

Теперь Тьюринг включит формулу Des($\mathcal{M}$) в большую формулу, которую он называет Un($\mathcal{M}$) от *Undecidable* (*Неразрешимая*). Эта формула Un – импликация вида:

Некоторая машина → Печатает нуль.

Формула использует функцию преемника *F*, а также логическую функцию *N*, которая истинна, когда ее аргумент – натуральное число.

Формула Un($\mathcal{M}$) должна быть такой

$$\begin{aligned}(\exists u)[N(u)\,\&\,(x)(N(x) \rightarrow\ & (\exists x')F(x,x'))\\ & \&\,(y,z)(F(y,z) \rightarrow N(y)\,\&\,N(z))\,\&\,(y)R_{S_0}(u,y)\\ & \&\,I(u,u)\,\&\,K_{q_1}(u)\,\&\,\text{Dcs}(\mathcal{M})]\\ & \rightarrow (\exists s)(\exists t)[N(s)\,\&\,N(t)\,\&\,R_{S_1}(s,t)].\end{aligned}$$

Знак импликации в начале четвертой строки делит формулу на две части. В каждой части есть выражение в квадратных скобках, которому предшествует один или два квантора существования.

Последняя строка – самая несложная: она просто говорит, что существуют такие два числа *s* и *t*, что символ S_1 (нуль) появляется в полной конфигурации *s* в клетке *t* ленты. В разделе 8 Тьюринг доказал, что не существует алгоритма, который скажет нам, напечатает ли когда-нибудь произвольная машина символ нуль, так что я полагаю, что вы уже начинаете видеть Тьюринга, спускающегося вниз по математическому дымоходу.

Формула Un($\mathcal{M}$) начинается с утверждения о существовании числа *u* (числа нуль), которое служит номером и первой полной конфигурации, и первой просмотренной клетки. Остаток первой строки оз-

начает лишь то, что для любого числа x существует другое число x', которое является преемником x.

Вторая строка чуть многословнее, но утверждает только то, что для полной конфигурации u (нуль) и в каждой клетке y на ленте находится символ S_0 или пробел. Это начальное состояние ленты. Третья строка содержит функцию I, подтверждающую, что в нулевой полной конфигурации в читаемой клетке – нуль, и функцию K, устанавливающую начальную m-конфигурацию в q_1. Дальше идет выражение Des($\mathcal{M}$), которое описывает саму машину.

Из предыдущей формулы Тьюринг извлекает лишь фрагмент в квадратных скобках и вводит еще одно сокращение:

[$N(u)$ & ... & Des($\mathcal{M}$)] может быть сокращено до $A(\mathcal{M})$.

Утверждение $A(\mathcal{M})$ заключает в себе начальные условия машины и описание машины, но имеет свободную переменную u.

В опубликованном Исправлении к статье Тьюринг учел шероховатости этого доказательства, о которых я говорил в главе 12. Он не проверял единственность преемников. По этой причине он определил логическую функцию $G(x, y)$, которая истинна, если y больше x, и выражение $\mathcal{Q}$, которым предполагалось заменить представление P аксиом Пеано:

$\mathcal{Q}$ – сокращение для:

$$(x)(\exists w)(y,z)\{F(x,w)\,\&\,(F(x,y)\rightarrow G(x,y))$$
$$\&\,(F(x,z)G(z,y)\rightarrow G(x,y))$$
$$\&\,[G(z,x)\vee(G(x,y)\,\&\,F(y,z)\vee$$
$$(F(x,y)\,\&\,F(z,y))\rightarrow(-F(x,z))]\}.$$

С таким определением, опекающим натуральные числа, сокращение $A(\mathcal{M})$ становится гораздо проще. Это – конъюнкция $\mathcal{Q}$, (начальных условий машины) и описания машины:

$A(\mathcal{M})$ – сокращение для:

$$\mathcal{Q}\,\&\,(y)R_{S_0}(u,y)\,\&\,I(u,u)\,\&\,K_{q_1}(u)\,\&\,\mathrm{Des}(\mathcal{M})$$

В $A(\mathcal{M})$ переменная u – свободная. Эта переменная становится связанной в формуле Un($\mathcal{M}$):

Un($\mathcal{M}$) – это формула:
$(\exists u)A(M) \rightarrow (\exists s)(\exists t)R_{S_1}(s,t)$

Я удалил предикаты $N(x)$, потому что по умолчанию считается, что область определения всех этих логических функций – натуральные числа. Относительно определений R, I, K, предикатов F и формул Inst и Des Тьюринг говорит:

> Подставив предполагаемые на с. 259–60 значения, мы обнаружим, что Un($\mathcal{M}$) имеет значение «в некоторой полной конфигурации $\mathcal{M}$ на ленте появляется S_1 (то есть 0)».

Поскольку выражение слева от импликации в Un($\mathcal{M}$) включает описание машины и ее начальные условия, мы считаем их истинными. Они – аксиомы. Справа от импликации – выражение, которое истинно, если машина когда-нибудь во время своей работы напечатает нуль. Поэтому вся формула Un($\mathcal{M}$) истинна, если истинна правая часть (то есть машина когда-нибудь напечатает нуль), и ложна, если машина никогда не напечатает нуль. Существует ли алгоритм, устанавливающий доказуемость Un($\mathcal{M}$)? Если это так, то существует и алгоритм, который скажет нам, что произвольная машина когда-либо напечатает нуль.

Заметьте, что Тьюринг ссылается на «предполагаемые» значения логических функций. Оставшаяся, большая часть доказательства будет основана исключительно на синтаксической интерпретации формул без учета точного смысла этих функций.

Теперь Тьюринг хочет показать, что Un($\mathcal{M}$) доказуема тогда и только тогда, когда на ленте появляется S_1. Каждую половину этого доказательства он снабжает леммами (вспомогательными теоремами), которые вскоре назовет Лемма 1 и Лемма 2, но сначала называет их (*a*) и (*b*):

> В соответствии с этим я доказываю, что
>
> (*a*) Если в некоторой полной конфигурации $\mathcal{M}$ на ленте появляется S_1, то Un($\mathcal{M}$) доказуема.
>
> (*b*) Если Un($\mathcal{M}$) доказуема, то в некоторой полной конфигурации $\mathcal{M}$ на ленте появляется S_1.
>
> Если это сделано, остальная часть теоремы тривиальна.

Самое сложное – это Лемма 1, которую Тьюринг теперь повторяет дословно.

ЛЕММА 1. *Если в некоторой полной конфигурации $\mathcal{M}$ на ленте появляется S_1, то Un($\mathcal{M}$) доказуема.* [261]

Мы должны показать, как доказать Un($\mathcal{M}$).

Как и прежде, u – это то, что мы обычно называем нулем, u' – это 1, u'' – это 2, а $u^{(n)}$ обозначает u с n штрихами и представляет собой число n.

Кроме того, в следующем предложении статьи Тьюринг незаметно вводит несколько новых обозначений, состоящих из трех новых функций $r(n, m)$, $i(n)$ и $k(n)$. Это не логические функции, потому что они возвращают целые числа. Аргумент n – это полная конфигурация, а m – клетка ленты.

Функция $r(n, m)$ возвращает *индекс* символа, который появляется в полной конфигурации n в клетке m. Этот индекс равен 0 для пробела, 1 – для нуля, 2 – для 1 и т. д. Таким образом, $S_{r(n, m)}$ – это символ, который появляется в полной конфигурации n в клетке m. Тьюринг соединит эту функцию r на месте индекса S с логической функцией R:

$$R_{S_{r(n,m)}}(u,m).$$

Этот предикат всегда истинен, хотя Тьюринг вместо этого будет использовать в качестве аргументов R значения u с верхними индексами:

$$R_{S_{r(n,m)}}(u^{(n)},u^{(m)}).$$

Тьюринг позволяет себе использовать n и m в функции r, но требует $u^{(n)}$ и $u^{(m)}$ в предикате R.

Вторая представляемая Тьюрингом новая функция – $i(n)$, которая возвращает номер просматриваемой клетки в полной конфигурации n так, что предикат

$$I(u^{(n)}, u^{(i(n))})$$

всегда истинен, потому что он ссылается на полную конфигурацию n и просматриваемую клетку $i(n)$. Третья новая функция $k(n)$ возвращает *индекс* m-конфигурации в полной конфигурации n, так чтобы $q_{k(n)}$ была m-конфигурацией в полной конфигурации n. Предикат

$$K_{q_{k(n)}}(u^{(n)}).$$

всегда истинен, потому что означает, что $q_{k(n)}$ – m-конфигурация полной конфигурации n.

Давайте предположим, что в n-й полной конфигурации последовательность символов на ленте – $S_{r(n,0)}, S_{r(n,1)}, ..., S_{r(n,n)}$, за которой следуют только пробелы, что просматриваемый символ – $i(n)$-й и что $q_{k(n)}$ – m-конфигурация.

В полной конфигурации 0 вся лента – пустая. В полной конфигурации 1 на ленте возможен один непустой символ. В самом общем случае в полной конфигурации n на ленте возможно не более n непустых символов (а возможно, и меньше). Тьюринг представляет эту цепочку символов с началом в клетке 0, как $S_{r(n,0)}, S_{r(n,1)}, ..., S_{r(n,n)}$. В том, что он приводит здесь фактически $n + 1$ символ вместо n, нет проблемы, потому что некоторые из этих функций r несомненно возвращают 0 и, следовательно, ссылаются на пустые клетки.

Тогда мы можем построить высказывание

$$R_{S_{r(n,0)}}(u^{(n)},u)\,\&\,R_{S_{r(n,1)}}(u^{(n)},u')\,\&\ldots\&\,R_{S_{r(n,n)}}(u^{(n)},u^{(n)})$$
$$\&\,I(u^{(n)},u^{(i(n))})\,\&\,K_{q_{k(n)}}(u^{(n)})$$
$$\&\,(y)F((y,u')\mathrm{v}F(u,y)\mathrm{v}F(u',y)\mathrm{v}\ldots\mathrm{v}F(u^{(n-1)},y)\mathrm{v}R_{S_0}(u^{(n)},y)),$$

которое мы можем обозначить как CC_n.

То есть «полная конфигурация n» (*C*omplete *C*onfiguration n). Первая строка содержит конъюнкцию функций, соответствующих символам в первых $n + 1$ клетках. Вторая строка включает функции, относящиеся к просматриваемой клетке $i(n)$ и m-конфигурации $q_{k(n)}$.

В начале третьей строки F справа от квантора общности должна быть внутри больших круглых скобок. Если первая строка задает символы, которые появляются в клетках с номерами от 0 до n, то эта, последняя строка означает, что клетки с номерами, большими n, содержат пустые символы. Это – предикат R, который появляется в самом ее конце. Квантор общности y распространяется на все клетки. Или u' – преемник клетки y (то есть в y находится 0), или клетка y – преемник u (то есть в y находится 1), или клетка y – преемник u' (то есть в y находится 2), и так до тех пор, пока клетка y не оказывается преемником $n - 1$ (то есть в y находится n). Если y не относится ни к одному из этих случаев, клетка y содержит пустой символ.

Вот исправленная версия:

CC_n – сокращение для:

$$R_{S_{r(n,0)}}(u^{(n)},u)\,\&\,R_{S_{r(n,1)}}(u^{(n)},u')\,\&\,\ldots\,\&\,R_{S_{r(n,n)}}(u^{(n)},u^{(n)})$$
$$\&\,I(u^{(n)},u^{(i(n))})\,\&\,K_{q_{k(n)}}(u^{(n)})$$
$$\&\,(y)(F(y,u')\vee F(u,y)\vee F(u',y)\vee\ldots\vee F(u^{(n-1)},y)\vee R_{S_0}(u^{(n)},y)),$$

Если n равно нулю, первая строка исчезает, как и большая часть третьей строки. CC_0 – сокращение для

$$I(u,y)\,\&\,K_{q_1}(u)\,\&\,(y)R_{S_0}(u,y).$$

Как и раньше, $F(u, u')$ & $F(u', u'')$ & ... & $F(u^{(r-1)}, u^{(r)})$ обозначается как $F^{(r)}$.

Я покажу, что все формулы вида $A(\mathcal{M})$ & $F^{(n)} \to CC_n$ (сокращенно CF_n) доказуемы.

Вы помните, что формула $A(\mathcal{M})$ объединяет начальное условие машины (пустая лента, m-конфигурация q_1, просматриваемая клетка с номером 0) и описание машины Des($\mathcal{M}$). Выражение Des($\mathcal{M}$) объединяет множество выражений Inst. Каждое выражение Inst означает, как команда изменяет символ в просматриваемой клетке, m-конфигурацию и клетку, которая должна быть просмотрена.

Тьюринг, по существу, определяет формулу CF_n для каждой полной конфигурации n.

CF_n – сокращение для:

$A(\mathcal{M})$ & $F(n) \to CC_n$

Вот несколько первых из формул CF_n:

CF_0: $A(\mathcal{M}) \to CC_0$
CF_1: $A(\mathcal{M})$ & $F(u, u') \to CC_1$
CF_2: $A(\mathcal{M})$ & $F(u, u')$ & $F(u', u'') \to CC_2$
CF_3: $A(\mathcal{M})$ & $F(u, u')$ & $F(u', u'')$ & $F(u'', u''') \to CC_3$
...

Смысл CF_n – «n-я полная конфигурация $\mathcal{M}$ – это то-то и то-то», где «то-то и то-то» означает фактическую n-ю полную конфигурацию $\mathcal{M}$. Поэтому можно надеяться, что CF_n будет доказуемой.

Тьюринг показывает доказуемость формул CF_n методом индукции. Сначала он доказывает, что доказуема CF_0, а затем показывает, что если доказуема CF_n, то доказуема и CF_{n+1}.

CF_0, очевидно, доказуема, поскольку в полной конфигурации все символы – пустые, m-конфигурация – q_1, а просматриваемая клетка – u, то есть CC_0

$$(y)R_{S_0}(u,y) \,\&\, I(u,u) \,\&\, K_{q_1}(u).$$

Это просто перестроенная версия формулы CC_0, которую я показал ранее. Выражение для $A(\mathcal{M})$ таково:

$$Q \,\&\, (y)R_{S_0}(u,y) \,\&\, I(u,u) \,\&\, K_{q_1}(u) \,\&\, \mathrm{Des}(\mathcal{M})$$

$A(\mathcal{M})$ содержит те же, что и CC_0, предикаты R, I и K.

Тогда $A(\mathcal{M}) \to CC_0$ тривиальна.

Затем мы показываем, что $CF_n \to CF_{n+1}$ доказуема для каждого n.

Если вы посмотрите на выражение CF_n, то увидите, что

$$CF_n \to CF_{n+1}$$

– это лишь сокращение формулы

$$[A(\mathcal{M}) \,\&\, F^{(n)} \to CC_n] \to [A(\mathcal{M}) \,\&\, F^{(n+1)} \to CC_{n+1}].$$

Это более сложная часть индуктивного доказательства, но доказательство этой импликации позволит нам говорить, что $CF_0 \to CF_1$ доказуема, $CF_1 \to CF_2$ доказуема и т. д., так что все выражения CF_n доказуемы.

Рассмотрим три случая в зависимости от того, куда перемещается машина из n-й в $(n + 1)$-ю конфигурацию, – влево, вправо или остается на месте. Допустим, имеет место первый случай, *то есть* машина перемещается влево. Подобный аргумент применим и для других случаев.

В первой части следующего предложения Тьюринг определяет целые числа *a*, *b*, *c* и *d*, основанные на введенных ранее функциях *r*, *i* и *k*, но эти определения несколько запутанные:

> Если
>
> $r(n, i(n)) = a,\ r(n + 1, i(n + 1)) = c,\ k(i(n)) = b$ и $k(i(n + 1)) = d,$

В опубликованных Исправлениях этой статьи Тьюринг распутывает определения.

Все они относятся к полной конфигурации *n* и должны быть такими:

$a = k(n)$, индекс *m*-конфигурации;
$b = r(n, i(n))$, индекс символа в просматриваемой клетке $i(n)$;
$c = k(n + 1)$, индекс следующей *m*-конфигурации;
$d = r(n + 1, i(n))$, индекс нового символа в клетке $i(n)$.

a и *c* обозначают индексы для *q*; *b* и *d* обозначают индексы для *S*. Эти обозначения существуют исключительно для упрощения окончания этого предложения и нескольких следующих фрагментов:

> тогда $\mathrm{Des}(\mathcal{M})$ должен включать $\mathrm{Inst}\{q_a\ S_b\ S_d\ L\ q_c\}$ в качестве одного из членов, *m. e.*
>
> $$\mathrm{Des}(\mathcal{M}) \rightarrow \mathrm{Inst}\{q_a\ S_b\ S_d\ L\ q_c\}.$$

Поскольку Des состоит из конъюнкции всех членов Inst, то если Des истинна, тогда каждый отдельный член Inst тоже истинен, и импликация истинна. $A(\mathcal{M})$ – конъюнкция Des и прочих выражений, значит:

$$A(\mathcal{M}) \rightarrow \mathrm{Des}(\mathcal{M}).$$

Объединяя эти две импликации, мы имеем:

$$A(\mathcal{M}) \rightarrow \mathrm{Inst}\ \{q_a\ S_b\ S_d\ L\ q_c\}.$$

Выражение $F^{(n+1)}$ – сокращение для конъюнкции предикатов *F*, которые, как тоже предполагается, суть аксиомы. Поскольку $F^{(n+1)}$ истинно, мы можем присоединить его операцией конъюнкции к обеим частям этой формулы:

> Следовательно, $A(\mathcal{M})\ \&\ F^{(n+1)} \rightarrow \mathrm{Inst}\ \{q_a\ S_b\ S_d\ L\ q_c\}\ \&\ F^{(n+1)}.$
>
> Но доказуема, $\mathrm{Inst}\{q_a\ S_b\ S_d\ L\ q_c\}\ \&\ F^{(n+1)} \rightarrow (CC_n \rightarrow CC_{n+1})$

Другая установка требует: конъюнкция слева должна также включать $\mathcal{Q}$ для подтверждения единственности преемников:

$\text{Inst}\{q_a\, S_b\, S_d\, L\, q_c\}\ \&\ \mathcal{Q}\ \&\ F^{(n+1)} \to (CC_n \to CC_{n+1})$.

Если заменить индексы a, b, c и d в выражении Inst, то можно видеть, что она превращается в команду Inst, которая вызывает переход от полной конфигурации n к полной конфигурации $(n+1)$:

$\text{Inst}\ \{q_{k(n)}\, S_{r(n,\, i(n))}\, S_{r(n+1,\, i(n))}\, L\, q_{k(n+1)}\}\ \&\ \mathcal{Q}\ \&\ F^{(n+1)} \to (CC_n \to CC_{n+1})$.

Эта формула равносильна

$(CC_n\ \&\ \text{Inst}\ \{q_{k(n)}\, S_{r(n,\, i(n)}\, S_{r(n+1,\, i(n))}\, L\, q_{k(n+1)}\}\ \&\ \mathcal{Q}\ \&\ F^{(n+1)}) \to CC_{n+1}$

и интуитивно кажется довольно очевидной: CC_{n+1} следует из CC_n в конъюнкции с командой Inst, которая вызывает переход от CC_n к CC_{n+1}. Однако показать, что она доказуема, манипулируя входящими в нее логическими функциями, довольно замысловато из-за сложности обозначений Inst и CC_n.

Два утверждения, которые только что представил Тьюринг, доказуемы

и, следовательно, таким образом

$$A(\mathcal{M})\ \&\ F^{(n+1)} \to (CC_n \to CC_{n+1})$$

Это утверждение вида $\text{X} \to (\text{Y} \to \text{Z})$, и довольно легко показать, что оно равносильно $(\text{X} \to \text{Y}) \to (\text{X} \to \text{Z})$, поэтому:

$(A(\mathcal{M})\ \&\ F^{(n+1)} \to CC_n) \to (A(\mathcal{M})\ \&\ F^{(n+1)} \to CC_{n+1})$.

Вспомним, что F с верхним индексом – это сокращение для конъюнкции предикатов F, поэтому $F^{(n+1)} \to F^{(n)}$ и $(A(\mathcal{M})\ \&\ F^{(n+1)}) \to (A(\mathcal{M})\ \&\ F^{(n)})$

[262]

и $$(A(\mathcal{M})\ \&\ F^{(n)} \to CC_n) \to (A(\mathcal{M})\ \&\ F^{(n+1)} \to CC_{n+1}),$$

Оба выражения в скобках имеют вид сокращения CF_n,

т. е. $$CF_n \to CF_{n+1}.$$

CF_n доказуема для любого n.

Индуктивное доказательство, показывающее доказуемость CF_n, закончено, но мы ещё не покончили с леммой, потому что на самом деле должны доказать $\text{Un}(\mathcal{M})$.

Теперь согласно предположению этой леммы S_1 появляется в некоторой полной конфигурации где-то в последовательности символов, печатаемых $\mathcal{M}$; то есть для некоторых целых чисел N, K,

где N и K – соответственно номера полной конфигурации и клетки,

CC_N имеет $R_{S_1}(u^{(N)}, u^{(K)})$ одним из своих членов, и поэтому $CC_N \rightarrow R_{S_1}(u^{(N)}, u^{(K)})$ доказуема.

Это так, потому что любой член конъюнкции подразумевает этот член.

Тогда мы имеем

$$CC_N \rightarrow R_{S_1}(u^{(N)}, u^{(K)})$$

и

$$A(\mathcal{M}) \ \& \ F^{(N)} \rightarrow CC^N.$$

Верхний индекс при CC на самом деле должен быть нижним, но это – именно то определение CF_N, которое, как только что показал Тьюринг, доказуемо для всех N (хотя ранее он использовал строчную n, а не прописную N).

До сих пор Тьюринг имел дело с формулами, которые включали свободную переменную, названную u.

Мы также имеем

$$(\exists u)A(\mathcal{M}) \rightarrow (\exists u)(\exists u') \ldots (\exists u^{(N')}) \ (A(\mathcal{M}) \ \& \ F^{(N)}),$$

где $N' = \max(N, K)$.

На самом деле K (номер клетки, где появляется нуль) никогда не может быть больше N (полная конфигурация), поэтому N' всегда равен N. Выражение $A(\mathcal{M}) \ \& \ F^{(N)}$ справа, как только что показано, влечет CC_N, которое, как только что показано, влечет $R_{S1}(u^{(N)}, u^{(K)})$.

И таким образом

$$(\exists u)A(\mathcal{M}) \rightarrow (\exists u)(\exists u') \ldots (\exists u^{(N')}) \ R_{S_1}(u^{(N)}, u^{(K)}),$$

Функция R не требует всех целых чисел от u до $u^{(N')}$. Она требует только $u^{(N)}$ и $u^{(K)}$, таким образом, большая часть кванторов существования может быть удалена, и у нас остается:

$$(\exists u)A(\mathcal{M}) \rightarrow (\exists u^{(N)})(\exists u^{(K)})\ R_{S_1}(u^{(N)}, u^{(K)}),$$

Если мы заменим $u^{(N)}$ на s и $u^{(K)}$ на t, то получим:

$$(\exists u)\ A(\mathcal{M}) \rightarrow (\exists s)(\exists t)\ R_{S_1}(s, t),$$

Это и есть в точности определение $\mathrm{Un}(\mathcal{M})$, которое можно видеть в небольшом блоке на странице 301. Странно, однако, что это не то определение $\mathrm{Un}(\mathcal{M})$, которое подразумевалось Тьюрингом изначально. У него в выражении справа от знака импликации были $N(s)$ и $N(t)$, но эти предикаты означают лишь то, что s и t – натуральные числа, а это и так подразумевалось.

Начав с предположения, что «S_1 появляется на ленте в некоторой полной конфигурации $\mathcal{M}$», мы только что доказали формулу, которая была определена как $\mathrm{Un}(\mathcal{M})$,

то есть $\mathrm{Un}(\mathcal{M})$ доказуема.

Этим заканчивается доказательство Леммы 1.

Вторая лемма намного короче и требует лишь раскрытия смысла формулы, используя определенные ранее логические функции.

ЛЕММА 2. *Если* $\mathrm{Un}(\mathcal{M})$ *доказуема, то* S_1 *появляется на ленте в некоторой полной конфигурации* $\mathcal{M}$.

Если мы заменим функциональные переменные в доказуемой формуле какими-нибудь логическими функциями, мы получим истинное высказывание. В частности, если мы подставим в $\mathrm{Un}(\mathcal{M})$ значения, приведенные в таблице на с. 259–260, мы получим истинное высказывание со значением «S_1 появляется где-то на ленте в некоторой полной конфигурации $\mathcal{M}$».

Теперь Тьюринг установил, что $\mathrm{Un}(\mathcal{M})$ доказуема тогда и только тогда, когда S_1 появляется на ленте в некоторой полной конфигурации $\mathcal{M}$.

Теперь мы в состоянии показать, что Entscheidungsproblem не может быть разрешена. Давайте предположим обратное. Тогда существует общий (механический) процесс определения доказуемости $\mathrm{Un}(\mathcal{M})$. Согласно Леммам 1 и 2, это предполагает, что существует процесс, который устанавливает, что $\mathcal{M}$ когда-либо напечатает 0, но это невозможно, согласно §8. Следовательно, Entscheidungsproblem не может быть разрешена.

Оглянувшись назад, вы разве не согласитесь, что это была лишь часть пирога?

Вообще говоря, неудивительно, что Un($\mathcal{M}$) – довольно сложная формула. Если бы она была много проще, она имела бы вид, который можно было проанализировать разрешающей процедурой. Напротив, Un($\mathcal{M}$) включает $A(\mathcal{M})$ в качестве одного из своих членов, $A(\mathcal{M})$ содержит среди своих членов $\mathcal{Q}$ и Des($\mathcal{M}$), а Des($\mathcal{M}$) – конъюнкция всех членов Inst, которые образуют машину. У каждого члена Inst – пять кванторов общности, $\mathcal{Q}$ имеет три квантора общности и один квантор существования, $A(\mathcal{M})$ имеет еще один квантор общности, а Un($\mathcal{M}$) имеет три квантора существования.

Эта сложная вложенность кванторов не имела бы значения для разрешимости высказывания, если бы оказалось так, что оно содержит только одноместные предикаты, то есть предикаты лишь с одним аргументом. В 1915 году Леопольд Лёвенгейм (1878–1957) доказал, что высказывания, содержащие лишь одноместные предикаты, разрешимы[1].

Ко времени написания Тьюрингом своей статьи уже были достигнуты большие успехи в поиске разрешающих процедур для частных случаев формул. Вообще, перед применением разрешающей процедуры формула сначала преобразуется в *предваренную нормальную форму*, когда все кванторы (без отрицаний), переносятся в начало формулы и предшествуют выражению без кванторов, называемому «матрицей».

В те годы разные математики нашли разрешающие процедуры для классов формул в предваренной нормальной форме, которые начинаются с определенного шаблона кванторов. В литературе по проблемам разрешимости[2] эти шаблоны кванторов задаются с использованием символов $\forall$ и $\exists$ для кванторов общности и существования. Верхний числовой индекс указывает на конкретное количество кванторов, а звездочка – на произвольное их количество.

В 1928 году Пол Бернайс и Мозес Шёнфинкель (1889–1942) опубликовали разрешающую процедуру для утверждений, начинающих-

---

[1] Leopold Löwenheim, «Über Möglichkeiten im Relativekalkül», *Mathematische Annalen*, Vol. 76 (1915), 447–470. Английский перевод: «On Possibilities in the Calculus of Relatives» в кн. Jean van Heijenoort, ed., *FromFrege to G del: A Source Book in Mathematical Logic, 1879–1931* (Harvard University Press, 1967), 228–251.

[2] Лучше всего в Egon Börger, Erich Grädel, Yuri Gurevich, *The Classical Decision Problem* (Springer, 1997). В этой книге и в ее библиографии приводятся статьи, касающиеся Entscheidungsproblem и ее частных решений.

ся с $\exists^*\forall^*$ (произвольное число кванторов существования, сопровождаемых произвольным числом кванторов общности). В 1928 году Вильгельм Аккерман привел разрешающую процедуру для $\exists^*\forall\exists^*$ (произвольное число кванторов существования, сопровождаемых одним квантором общности, сопровождаемым произвольным числом кванторов существования). В 1932 году Гёдель привел разрешающую процедуру для двух кванторов общности между произвольным числом кванторов существования: $\exists^*\forall^2\exists^*$.

В связи с проблемой разрешимости исследовались также *редукционные классы*. Редукционный класс состоит из всех предложений, которые начинаются с особого шаблона кванторов. Было доказано, что предложения различных редукционных классов имеют разрешающую процедуру, только когда *все* предложения имеют разрешающую процедуру. В 1920 году Сколем доказал, что $\forall^*\exists^*$ определяет редукционный класс, а в 1933 году Гёдель сузил его до $\forall^3\exists^*$.

До доказательств Чёрча и Тьюринга не было известно, имеют ли данные редукционные классы разрешающие процедуры, за исключением лишь того, что если бы существовала разрешающая процедура для редукционного класса, то должна была существовать и *общая* разрешающая процедура. Следствием статей Чёрча и Тьюринга было то, что редукционные классы неразрешимы. В 1932 году Гёдель привел разрешающую процедуру для высказываний с префиксом $\exists^*\forall^2\exists^*$ и, шире, $\forall^2\exists^*$. После доказательства Тьюринга стало понятно, что высказывания с префиксом $\forall^3\exists^*$ неразрешимы. С добавлением всего одного квантора общности разрешимое высказывание вида $\forall^2\exists^*$ оборачивается неразрешимым высказыванием вида $\forall^3\exists^*$.

В следующем параграфе Тьюринг называет кванторы (quantifiers) словом *quantor*.

> Ввиду большого количества частных случаев решений Entscheidungsproblem для формул с ограниченными системами кванторов [263]
> интересно представить $\mathrm{Un}(\mathcal{M})$ в таком виде, когда все кванторы находятся в начале. Фактически $\mathrm{Un}(\mathcal{M})$ представляется в виде
>
> $$(u)(\exists x)(w)(\exists u_1)\ldots(\exists u_n)\mathfrak{B}, \qquad \text{(I)}$$
>
> где $\mathfrak{B}$ не содержит кванторов, а $n = 6$.

Тьюринг доказал что высказывания с префиксом $\forall\exists\forall\exists^6$ (используя общепринятую систему обозначений) образуют редукционный

класс. Не может быть разрешающего процесса для высказываний с таким префиксом.

Путем несложных преобразований мы можем получить формулу со всеми основными свойствами $\mathrm{Un}(\mathcal{M})$, которая имеет вид (I) с $n = 5$.

В Исправлениях к этой статье Тьюринг отмечает, что это последнее число должно быть равно 4, так что редукционный класс сужается до $\forall\exists\forall\exists$[4].

В 1962 году швейцарский логик Юлиус Рихард Бюхи (1924–1984) применил другой подход к Entscheidungsproblem с использованием машин Тьюринга и сумел несколько упростить доказательство[1]. Он показал, что высказывания вида $\exists\&\forall\exists\forall$ образуют редукционный класс. (Это высказывание – конъюнкция двух членов, каждому из которых предшествуют его собственный квантор или кванторы.) Статья Бюхи также заложила основы для доказательства того, что высказывания $\forall\exists\forall$ образуют редукционный класс, что означает, что не может быть общей разрешающей процедуры даже для высказываний с казалось бы простым префиксом $\forall\exists\forall$.

Разрабатывая методы решения задач, математики приносят миру огромную пользу, но столь же полезно, когда они доказывают, что нечто не имеет решения. Не существует способа разделить угол на три равные части с помощью линейки и циркуля, нет способа построить квадрат, равновеликий кругу, и нет способа доказать пятую аксиому Евклида на основании первых четырех. Не существует целочисленных решений уравнения $x^n + y^n = z^n$ для n, большего 2, нет способа установить в рамках арифметической системы ее непротиворечивость и не существует общей разрешающей процедуры для логики первого порядка.

Можно прекратить тратить впустую наше время, пытаясь их найти. Знание того, что невозможно, так же важно, как знание того, что возможно.

---

[1] J. Richard Büchi, «Turing-Machines and the Entscheidungsproblem», *Mathematische Annalen*, Vol. 148, No. 3 (June 1962), 201–213.

Глава 15

Лямбда-исчисление

В 1983 или 1984 годах, когда Алонзо Чёрчу было примерно 80 лет, его пригласили выступить в Центре изучения языка и информации Стэнфордского университета и еще пригласили на небольшую презентацию CSLI-компьютеров Одуванчик (Dandelion) компании Xerox. Эти компьютеры выполняли программы на языке Лисп, разработанном Джоном Маккарти (род. 1927). Чёрчу рассказали, что язык обработки списков Лисп был основан на лямбда-исчислении, которое Чёрч придумал около 50 лет назад.

Чёрч признался, что ничего не понимает в компьютерах, но однажды у него был студент, который разбирался в этом[1]. Тогда все, конечно, поняли, что это был Алан Тьюринг.

Лямбда-исчисление, разработанное Алонзо Чёрчем в начале 1930-х годов, позволило ему доказать, что не существует общей разрешающей процедуры для логики предикатов первого порядка. Алан Тьюринг до публикации своей статьи о вычислимых числах и Entscheidungsproblem изучал это доказательство и потому был обязан добавить к своей статье приложение, в котором он показал, что его подход и подход Чёрча в основном эквивалентны. Это приложение – тема настоящей главы.

Понятия лямбда-исчисления кажутся знакомыми оттого, что они оказали значительное влияние на развитие языков программирования. Довольно давно была замечена структурная связь между лямбда-исчислением и такими *процедурными*, или *императивными*, языками программирования, как ранний Алгол[2], из которого вышли Паскаль, C и множество его производных – C++, Java и C#. Написанные на процедурных языках программы построены на передаче данных про-

---

[1] María Manzano, «Alonzo Church: His Life, His Work and Some of His Miracles», *History and Philosophy of Logic*, Vol. 18 (1997), 212.

[2] P. J. Landin, «A Correspondence Between ALGOL 60 and Church's Lambda-Notation», *Communications of the ACM*, Vol. 8, No. 2 (Feb. 1965), 89–101; Vol. 8, No. 3 (Mar. 1965), 158–165.

цедурам (называемым также подпрограммами или методами), которые обрабатывают эти данные различными способами.

Еще более сильное влияние лямбда-исчисление оказало на языки *функционального* программирования, такие как Лисп, АПЛ, Haskell, Scheme и F#. На функциональном языке функции выстраиваются в цепочку, в которой каждая функция получает результат от предыдущей. Функциональные языки часто позволяют работать с функциями почти так же, как процедурные языки работают с данными. Хотя функциональные языки не достигли популярности процедурных, сейчас они переживают некоторое возрождение.

Алонзо Чёрч родился в 1903 году в Вашингтоне, округ Колумбия, и провел большую часть своей профессиональной жизни в университете Принстона. Он пришел в Принстон студентом и в 24 года стал доктором философии по математике. Провел два года в качестве научного сотрудника центра Национальных исследований, а затем вернулся в Принстон, где преподавал с 1929 года и до своей отставки в 1967 году. Затем вплоть до 1990 года Чёрч подрабатывал в UCLA.

Чёрч был трудолюбивым и дотошным человеком. Он говорил хорошо продуманными законченными фразами и работал до поздней ночи. Занятия Чёрч часто начинал с ритуала тщательного вытирания доски, нередко с использованием целого ведра воды. Работая над математической задачей, он использовал чернила разных цветов, а когда ему нужны были другие цвета, сам смешивал их в различных пропорциях. Когда он заканчивал страницу, которую нужно было сохранить надолго, он покрывал ее лаком Duco, который считал особенно подходящим для этих целей, потому что тот не коробил бумагу[1].

Чёрч был руководителем 31 докторской диссертации, в том числе Стивена Клини (1931), Джона Баркли Россера (1934), Леона Хенкина (1947), Мартина Дэвиса (1950), Хартли Роджерса (1952) и Рэймонда Смальяна (1959), а также Алана Тьюринга (1938)[2].

Нередко полагают, что Алонзо Чёрч основал Ассоциацию символической логики, поскольку он был первым редактором *Журнала символической логики* (*The Journal of Symbolic Logic*). На самом деле он ее не основывал, но действительно вел журнал на очень высоком

---

[1] Herbert B. Enderton, «Alonzo Church: Life and Work», введение к *Collected Works of Alonzo Church* (MIT Press, forthcoming). Предварительный текст доступен на http://www.math.ucla.edu/~hbe/church.pdf.

[2] Herbert B. Enderton, «In Memoriam: Alonzo Church, 1903–1995», *The Bulletin of Symbolic Logic*, Vol. 1, No. 4 (1995), 486–488.

уровне, придавая ему большую значимость ценными библиографиями по логике.

История лямбда-исчисления начинается с работы, которую Чёрч выполнил, будучи с 1927 по 1929 год научным сотрудником центра Национальных исследований. В то время математики хотели получить определение аморфного понятия *эффективной вычислимости*. Чтобы понять возможности и пределы числовых вычислений, нужно было определить функции формально и систематически, то есть в виде символов и строк с определенными правилами. Как это сделать наилучшим образом? Можно ли после этого показать, что эти функции полностью охватывают эффективную вычислимость?

Первая статья Чёрча на эту тему была получена *The Annals of Mathematics* 5 октября 1931 года и опубликована в апреле следующего года[1]. Именно в ней Чёрч ввел малую букву лямбда (λ) для представления функций.

Отчасти поводом для новой системы обозначений стала некоторая неоднозначность общепринятого представления функций. Рассмотрим следующее выражение:

$x^2 + 5x + 7$.

Это выражение само по себе синтаксически правильное, хотя мы не уверены в том, что с ним предполагается делать. Гораздо понятнее:

$f(x) = x^2 + 5x + 7$.

Это общепринятое обозначение функции, где x – связанная (или независимая) переменная. Мы можем менять эту связанную переменную функции на все, что угодно, пока не натолкнемся на противоречие с чем-то еще:

$f(y) = y^2 + 5y + 7$.

Теперь мы можем представить конкретное значение функции выражением вида $f(4)$. Мы знаем, что для вычисления значения функции следует заменить независимую переменную x на 4:

$f(4) = (4)^2 + 5 \cdot (4) + 7 = 43$.

Проделав это, вы, наверное, поразитесь тому, что не существует стандартного способа записи функционального выражения (то есть

[1] Alonzo Church, «A Set of Postulates for the Foundation of Logic», *The Annals of Mathematics*, 2nd Series, Vol. 33, No. 2 (Apr. 1932), 346–366.

выражения $y^2 + 5y + 7$) для конкретного значения y. Стоит только заменить y на 4, как независимая переменная исчезает. Если бы вы отвечали за исправление этого недостатка и разрабатывали систему обозначений для записи функции с заданным значением, вы, наверное, могли бы придумать что-то вроде

$[y^2 + 5y + 7]\ (4)$.

Вроде бы неплохо. Но что, если выражение имеет много независимых переменных? Тогда

$[y^2 + 5y + 18x - 2xy + 7]\ (4)$

довольно двусмысленно. Даже если допустить вот это:

$[y2 + 5y + 18x - 2xy + 7]\ (4, 5)$,

то придется предположить, что значения для x и y задаются в определенном порядке.

В *Principia Mathematica* Уайтхеда и Рассела классы, удовлетворяющие определенным функциям, обозначались дугой (или крышкой) – $\hat{y}$. Чёрч хотел поместить дугу перед переменной $\hat{}y$, но из типографских соображений символ вскоре был заменен греческой буквой лямбда – λy[1].

Система обозначений Чёрча с годами видоизменялась. В последующих обсуждениях я все же буду придерживаться системы обозначений Тьюринга, применяемой в приложении к его статье. Функция одной переменной представляется таким общим синтаксисом:

$\lambda x[M]$,

где M – выражение, содержащее связанную переменную x. Функция из самого первого примера может быть записана как

$\lambda x[x^2 + 5x + 7]$.

Функция с определенным значением связанной переменной записывается согласно общему синтаксису:

$\{F\}(A)$.

F – это функция, и если F имеет независимую переменную, то формула представляет собой функцию, в которой A может заменять эту

---

[1] J. Barkley Rosser, «Highlights of the History of Lambda-Calculus», *Annals of the History of Computing*, Vol. 6, No. 4 (Oct. 1984), 337–349.

независимую переменную. Например, если функция имеет независимую переменную x, то ее общее обозначение:

$\{\lambda x[M]\}(A)$.

Для функции из примера оно становится таким:

$\{\lambda x[x^2 + 5x + 7]\}(A)$.

Если значение x должно быть равно 4, то это может быть записано так:

$\{\lambda x[x^2 + 5x + 7]\}\ (4)$.

Итак, мы успешно обозначили функцию со значением для независимой переменной.

Функция с двумя независимыми переменными имеет общий вид

$\{\{F\}(A)\}(B)$,

но для удобства и читаемости его можно сократить до:

$\{F\}(A, B)$.

Если подставить вместо F конкретную функцию, получим следующее:

$\{\lambda x\lambda y[y^2 + 5y + 18x - 2xy + 7]\}(A, B)$.

Теперь ясно, что A подставляется вместо x, а B – вместо y, потому что в таком порядке следуют лямбды в начале формулы.

Допускаются и другие сокращения. Фигурные скобки могут опускаться, если это не приводит к путанице, так что

$\{F\}(A, B)$

становится

$F(A, B)$,

что похоже на запись обычной функции, за исключением того, что выражение F на самом деле содержит ещё и лямбды:

$\lambda x\lambda y[M](A, B)$.

Чёрч допускал также замену пары квадратных скобок одной точкой после ряда лямбд:

$\lambda x\lambda y.M(A, B)$.

Именно в таком виде будет записываться далее большинство лямбда-выражений.

Определив основную систему лямбда-обозначений, Чёрч ввел выражения для общих логических операций и правил подстановки для преобразования формул в равносильные формулы. Чёрч определил эти правила преобразования чисто формально, но их можно свести к следующим:

1. Можно изменить отдельную связанную переменную (например, x на y), если новая переменная не конфликтует в формуле с чем-то еще.
2. Если в формуле {λx.M}(N) в значении N не встречается имя x, то все вхождения x в M можно заменить на N, после чего формула остается прежней, но в ней вместо исходной переменной x используется N.
3. Обратное к II тоже справедливо.

Спустя полтора года после той первой статьи о лямбда-функциях Чёрч опубликовал вторую[1]. Чёрч пересмотрел список аксиом и подчеркнул «полностью формальный характер системы, которая позволяет абстрагироваться от смысла символов и рассматривать доказательство теорем (формальной логики) как игру с символами на бумаге по определенным произвольным правилам»[1]. Такой подход во многом близок к традициям формализма.

Чёрч также ввел сокращение *conv*, означающее «согласно преобразованию» («by conversion»), чтобы указать, что одна формула преобразована в равносильную формулу согласно правилам I, II или III. Например,

$$\lambda x[x^2 + 5x + 7](A) \text{ conv } A^2 + 5A + 7.$$

Чёрч заканчивает вторую статью коротким разделом о положительных целых числах.

Он применил лямбда-обозначения для определения символа 1, преемника, операций сложения и умножения, а также для пяти аксиом Пеано и заявил: «Наша программа – разработать теорию положительных целых чисел на том основании, которое мы только что описали, а затем с помощью известных методов или соответствую-

---

[1] Alonzo Church, «A Set of Postulates for the Foundation of Logic (Second Paper)», *The Annals of Mathematics*, 2nd Series, Vol. 34, No. 4 (Oct. 1933), 839–864.

[2] Там же, 842.

щих их модификаций перейти к теории рациональных чисел и теории вещественных чисел»[1].

Последующие шаги в этом направлении были сделаны в статьях ученика Чёрча Стивена Коула Клини (чья фамилия произносится как «клейни») с участием Джона Баркли Россера (1907–1989). В 1934 году Клини опубликовал основополагающую работу «Доказательство посредством доводов в формальной логике» («Proof by Cases in Formal Logic»)[2] и упростил систему обозначений для многократных лямбд. Вместо

$$\lambda x \lambda y M$$

можно записать

$$\lambda xy M.$$

Докторская диссертация Клини «Теория положительных целых чисел в формальной логике» была защищена и опубликована в 1935 году в двух частях[3]. Предпосылками к этой работе были две статьи Чёрча и более ранняя статья Клини, но вторая ее часть ссылается и на будущую статью Чёрча и Россера[4].

Хотя лямбда-исчисление, разработанное Чёрчем, Клини и Россером, весьма обширно и включает как логику, так и арифметику, я хочу уделить внимание элементарной арифметике, чтобы вы хотя бы почувствовали, как сложение и умножение могут быть реализованы исключительно за счет манипуляций с символами.

Определение натуральных чисел всегда нужно начинать либо с 0, либо с 1; Чёрч и Клини начинают с 1, и вот как она обознается[1]:

$$1 \to \lambda fx.f(x).$$

Я привожу определения так, как они даны на первых 10 страницах работы Клини «A Theory of Positive Integers in Formal Logic, Part I».

Смысл стрелки – «значит» или «сокращение для». Сама формула может показаться немного странной. На самом же деле она кажется

---

[1] Alonzo Church, «A Set of Postulates for the Foundation of Logic (Second Paper)», *The Annals of Mathematics*, 2nd Series, Vol. 34, No. 4 (Oct. 1933), 864.

[2] S. C. Kleene, «Proof by Cases in Formal Logic», *The Annals of Mathematics*, 2nd Series, Vol. 35, No. 3 (July 1934), 529–544.

[3] S. C. Kleene, «A Theory of Positive Integers in Formal Logic, Part I», *American Journal of Mathematics*, Vol. 57, No. 1 (Jan. 1935), 153–173; Vol. 57, No. 2 (Apr. 1935), 219–244.

[4] Alonzo Church and J. B. Rosser, «Some Properties of Conversion», *Transactions of the American Mathematical Society*, Vol. 39, No. 3 (May 1936), 472–482.

крайне странной, но это – лишь определение, а оно не должно сразу же иметь смысл. Или в расширенной системе обозначений:

$$1 \to \{\lambda fx[f(x)]\}.$$

Таким образом, 1 – это, по сути дела, функция с двумя связанными переменными f и x. Поначалу кажется, что этих двух переменных на две больше, чем нужно для определения простейшего числа.

Это – функция преемника:

$$S \to \lambda\rho fx.f(\rho(f, x)).$$

И снова, согласитесь, довольно странная. Хотя мы понимаем, что у функции преемника должна быть одна связанная переменная, вряд ли мы ждем, что их у нее окажется целых *три*.

Символ 2, к счастью, определяется именно так, как мы ожидаем:

$$2 \to S(1).$$

Если мы действительно хотим применить функцию преемника к 1, мы должны быть уверены, что все связанные переменные уникальны, поэтому давайте воспользуемся для 1 следующим равносильным выражением:

$$1 \to \lambda ab.a(b).$$

При работе с лямбда-выражениями функции и переменные часто меняются ролями. Ниже в пошаговых преобразованиях формул я иногда использую фигурные скобки для выделения функции со связанной переменной, заменяемой на этом шаге.

Функцию $S(1)$ можно также записать в виде $\{S\}(1)$ или:

$$\{\lambda\rho fx.f(\rho(f, x))\}(\lambda ab.a(b)).$$

Первая связанная переменная в функции преемника – ρ, поэтому она в этой функции заменяется выражением для 1, а сама ρ после λ исчезает:

$$\lambda fx.f(\lambda ab.a(b)(f, x)).$$

Теперь эта формула содержит другую функцию с двумя аргументами:

$$\lambda fx.f(\{\lambda ab.a(b)\}(f, x)).$$

Заменим a на f, а b на x:

$$\lambda fx.f(f(x))$$

и все.

Поскольку число 1 первоначально было определено как

$1 \rightarrow \lambda fx.f(x)$,

число 2 определяется как

$2 \rightarrow S(1)\ \text{conv}\ \lambda fx.f(f(x))$.

Сравнив преобразованное выражение для 2 и выражение для 1, вы увидите еще одну f и пару круглых скобок справа от точки. Теперь запишем 2 с разными переменными в виде $\lambda ab.a((b))$ и попытаемся определить следующего преемника $\{S\}(2)$:

$\{\lambda\rho\, fx.f(\rho(f, x))\}(\lambda ab.a((b)))$.

Снова подставляем 2 вместо ρ:

$\lambda fx.f(\{\lambda ab.a((b))\}(f, x))$.

Заменяем a на f и b на x:

$\lambda fx.f(f(f(x)))$.

Это – лямбда-выражение для 3. Подозреваю, что вы уже начали замечать закономерность. В абстрактном представлении положительных целых чисел нам более всего хотелось бы ощущать их преемственность. Эта нотация дает нам такое ощущение преемственности: каждое следующее целое число имеет еще одно вложение первой связанной переменной.

Клини определил операцию сложения следующим образом:

$+ \rightarrow \lambda\rho\sigma fx.\rho(f, \sigma(f, x))$.

Сомневаетесь? Давайте сложим 2 и 3. Для начала нам нужно сделать все связанные переменные различными. Я буду использовать $\lambda ab.a((b))$ для 2 и $\lambda cd.c(c(c(d)))$ для 3, так что $\{+\}(2, 3)$:

$\{\lambda\rho\sigma\, fx.\rho(f, \sigma(f, x))\}(\lambda ab.a((b)), \lambda cd.c(c(c(d))))$.

В функции + заменим ρ формулой для 2, а σ – формулой для 3:

$\lambda fx.\lambda ab.a((b))(f, \{\lambda cd.c(c(c(d))\}(f, x))$.

Подставленное число 3 – это теперь функция, где c заменена на f, а d – на x:

$\lambda fx.\{\lambda ab.a((b))\}(f, f(f(f(x))))$.

Теперь подставленное 2 – это функция, где a заменена на f, а b – *на* $f(f(f(x)))$:

$\lambda fx.f\ (f\ (f\ (f\ (f\ (x)))))$.

Вот и все. Ответ совпадает с $S(S(S(S(1))))$ или равен 5, как у нас принято считать.

Интересно, что функция умножения проще функции сложения:

$\times \rightarrow \lambda\rho\sigma x.\rho(\sigma(x))$.

Давайте попробуем умножить 2 на 3. Мы можем записать $\{\times\}(2, 3)$ как

$\{\lambda\rho\sigma x.\rho(\sigma(x))\}(\lambda ab.a((b)), \lambda cd.c(c(c(d))))$.

Заменим ρ формулой для 2, а σ – формулой для 3:

$\lambda x.\lambda ab.a((b))(\{\lambda cd.c(c(c(d)))\}(x))$.

Теперь 3 стало функцией, где x подставлена вместо c:

$\lambda x.\{\lambda ab.a((b))\}(\lambda d.x(x(x(d))))$.

Теперь и 2 стало функцией. Подставим вместо a выражение справа:

$\lambda x.\lambda b.\lambda d.x(x(x(d)))(\{\lambda d.x(x(x(d))\}(b))$.

В функции справа подставим b вместо d

$\lambda x.\lambda b.\{\lambda d.x(x(x(d)))\}(x(x(x(b))))$

и закончим последней подстановкой вместо d:

$\lambda xb.x(x(x(x(x(b))))))$.

Это – число 6, которое определенно равно произведению 2 и 3.

Функциональное определение чисел позволяет делать некоторые необычные вещи, например

$\{2\}(3)$

или:

$\{\lambda ab.a((b))\}(\lambda cd.c(c(c(d))))$.

Если выполнить сложные подстановки, то в итоге можно прийти к

$\lambda bd.b(b(b(b(b(b(b(b(b(d)))))))))$

или к числу 9, которое равно 3 в степени 2, и не случайно. Именно поэтому возведение m в степень n определяется просто как:

$\lambda mn.nm$.

В этой системе обозначений умножение оказывается проще, чем сложение, а возведение в степень – самым простым. В ходе экспериментов с лябда-исчислением Чёрч, Клини и Россер обнаружили: что бы они ни задумывали, все можно было выразить в лямбда-нотации – то, что позже назвали λ-определимостью. «Размышления Чёрча в конце концов привели его к предположению, что все λ-определимые функции – эффективно вычислимы»[1].

Курт Гёдель приехал в Институт передовых исследований в 1933 году и весной 1934 года прочитал в Принстоне несколько лекций по теореме о неполноте, а также по рекурсивным функциям, то есть функциям, построенным на базе элементарных функций[2]. Интерес Гёделя к рекурсивным функциям вызвало письмо, которое он получил в 1931 году от Жака Эрбрана (1908–1931), блестящего молодого французского математика, который погиб при восхождении на гору в Альпах.

Однако в то время Гёдель был уверен, что ни лямбда-функций, ни рекурсивных функций не достаточно, чтобы охватить все из того, что мы неформально считаем эффективной вычислимостью.

В 1936 году Чёрч опубликовал статью «Неразрешимая проблема элементарной теории чисел»[3], в которой фактически впервые появляется термин «λ-определимая». (До этого Клини использовал лишь термины «определимая» или «формально определимая» для выражения логических и арифметических операций в лямбда-обозначениях.) Чёрч ссылается на более ранние свои статьи и две статьи Клини, а также на две будущие статьи Клини, которые исследуют отношение между рекурсивными и λ-определимыми функциями[4]. Используя нумерацию Гёделя, Чёрчу удалось построить неразрешимую проблему, как Гёделю – неразрешимое высказывание.

---

[1] Stephen C. Kleene, «Origins of Recursive Function Theory», *Annals of the History of Computing*, Vol. 3, No. 1 (Jan. 1981), 59.

[2] Записки Клини и Россера основаны на лекциях Гёделя, ходивших в то время, но не издававшихся до 1965 года, когда они были включены в кн. Martin Davis, ed., *The Undecidable* (Raven Press, 1965), 41–71. Позже они были изданы в кн. Kurt Gödel, *Selected Works: Volume I, Publications 1929–1936* (Oxford University Press, 1986), 346–371.

[3] Alonzo Church, «An Unsolvable Problem of Elementary Number Theory», *American Journal of Mathematics*, Vol. 58, No. 2 (Apr. 1936), 345–363.

[4] S. C. Kleene, «General Recursive Functions of Natural Numbers», *Mathematische Annalen*, Vol. 112, No. 1 (Dec. 1936), 727–742; перепечатана в кн. Martin Davis, ed., *The Undecidable* (Raven Press, 1965), 237–252. S. C. Kleene, «λ-Definability and Recursiveness», *Duke Mathematical Journal*, Volume 2, Number 2 (1936), 340–353.

На этом основании Чёрч в самом первом выпуске *Журнала символической логики* (который он же и редактировал) опубликовал двухстраничную статью «Заметки по Entscheidungsproblem» с выводом «*В общем случае Entscheidungsproblem engere Functionenkalkül неразрешима*»[1]. Статья была получена *Журналом* 15 апреля 1936 года, то есть за шесть недель до подачи Тьюрингом 28 мая 1936 года своей статьи в *Лондонское математическое общество*.

Вероятно, Тьюринг провел большую часть лета 1936 года за чтением упомянутых здесь статей Алонзо Чёрча и Стивена Клини, изучая лямбда-исчисление и его связь с его вычислительными машинами. В трехстраничном приложении Тьюринга указана дата его получения Лондонским математическим обществом – 28 августа 1936 года; в конце него Тьюринг в предчувствии своего будущего места пребывания добавил «Высший колледж, Университет Принстона, Нью-Джерси, США». Он не покидал Англии ради Соединенных Штатов вплоть до 23 сентября, приехав в Нью-Йорк только 29-го[2].

Добавлено 28 *августа* 1936 года.

ПРИЛОЖЕНИЕ.

Вычислимость и эффективная вычислимость

Ниже в общих чертах доказывается теорема о том, что все эффективно вычислимые (λ-определимые) последовательности вычислимы, и наоборот.

«В общих чертах» означает, что в доказательстве будут некоторые пропуски.

Предполагается, что термины «правильная формула» (*well-formed formula,* W.F.F.) и «преобразование», как они используются Чёрчем и Клини, понятны. Во втором из этих доказательств принимается без доказательства существование нескольких формул; эти формулы могут быть непосредственно выведены, *например* из результатов Клини в «Теории положительных целых чисел в формальной логике», *Американский матем. журнал*, 57 (1935), 153–173, 219–244.

---

[1] Alonzo Church, «A Note on the Entscheidungsproblem», *The Journal of Symbolic Logic*, Vol. 1, No. 1 (Mar.1936), 40–41. См. также Alonzo Church, «Correction to a Note on the Entscheidungsproblem», *The Journal of Symbolic Logic*, Vol. 1, No. 3 (Sep. 1936), 101–102.

[2] Andrew Hodges, *Alan Turing: The Enigma* (Simon & Schuster, 1983), 116.

Под «вторым из этих доказательств» Тьюринг подразумевает обратное утверждение: что каждая вычислимая последовательность λ-определима.

W.F.F., представляющая целое число *n*, будет обозначаться как N_n.

Используя определения Клини для 1 и последующих чисел (но со связанными переменными, совместимыми с тем, что Тьюринг вскоре покажет), *N*1 – это $\lambda xy.x(y)$, *N*2 – $\lambda xy.x(x(y))$ и *Nn* – $\lambda xy.x(x(x(x...(y)...)))$.

Будем говорить, что последовательность γ, чей *n*-ый символ $\phi_\gamma(n)$ является λ-определимой или эффективно вычислимой, если $1 + \phi_\gamma(u)$ – λ-определимая функция *n*,

Во втором появлении ϕ_γ (как и в первом) аргументом должна быть *n*, а не *u*, а само выражение – $1+\phi_\gamma(n)$. *n*-ый символ вычислимой последовательности γ – или 0, или 1, но λ-исчисление по определению Чёрча и Клини включает в себя только *положительные* целые числа, исключая нуль. Функция $\phi_\gamma(n)$ не может быть λ-определимой, так как нуль не является λ-определимым. Поэтому и добавляется 1, чтобы получились числа 1 и 2.

т. е. если существует W.F.F. M_γ, такая что для всех целых чисел *n*,

$$\{M_\gamma\}(N_n) \text{ conv } N\phi_\gamma(n)+1,$$

т. е. $\{M_\gamma\}(N_n)$ преобразуется в $\lambda xy.x(x(y))$ или в $\lambda xy.x(y)$ в зависимости от значения 1 или 0 *n*-го символа λ.

λ в последней строке – ошибка; должно быть «*n*-го символа γ». Функция M_γ от значения N_n (обозначающего цифру последовательности) преобразуется либо в $\lambda xy.x(x(y))$ (или 2), либо в $\lambda xy.x(y)$ (или 1) для цифр 1 или 0 соответственно.

Например, если пятая цифра γ – это 1, то $\phi_\gamma(5)$ есть 1, и

$$\{M_\gamma\}(N_5) \text{ conv } N_{\phi\gamma(5)+1},$$

что означает

$$\{M_\gamma\}(N_5) \text{ conv } N_2.$$

Чтобы показать, что каждая λ-определимая последовательность γ вычислима, мы должны показать, как построить машину для вычисления γ. Применительно к машинам удобно выполнить тривиальную модификацию преобразования в исчислении. Это изменение заключается в использовании переменных x, x', x'', ... вместо a, b, c, ...

Тьюринг здесь не использует переменные a, b или c, но использует x и y. Все переменные ему нужны в стандартном виде, чтобы иметь возможность выполнять сравнения и сопоставления. Это напоминает требование раздела 8 (страница 221 этой книги) о том, что логика предикатов первого порядка перед обработкой ее машиной должна быть «изменена, чтобы быть систематической».

Теперь для формулы M_γ мы строим машину $\mathscr{L}$, которая записывает на ленте последовательность γ. Построение $\mathscr{L}$ чем-то напоминает построение машины $\mathscr{K}$, которая доказывает все доказуемые формулы функционального исчисления. Сначала мы строим машину выбора $\mathscr{L}_1$, которая, получив правильную формулу, скажем M, и обработав ее соответствующим образом, выдает некоторую формулу, в которую преобразуется M. Затем $\mathscr{L}_1$ можно изменить так, чтобы получилась автоматическая машина $\mathscr{L}_2$, которая последовательно выдает все [264]
формулы, в которые преобразуется M (ср. со сноской на с. 252).

Из пяти сносок на странице 252 своей статьи Тьюринг имеет в виду вторую (страница 251 этой книги), где он обсуждает машину, которая доказывает все доказуемые формулы логики первого порядка. Эта же машина подобна той и, наверное, даже чуть проще с учетом очень органичного метода преобразования λ-выражений.

Машина $\mathscr{L}$ включает $\mathscr{L}_2$ как составную часть. Движение машины $\mathscr{L}$, получившей формулу $M\gamma$, делится на участки, n-ый из которых предназначен для поиска n-го символа γ. Первый шаг на этом n-ом участке – формирование $\{M_\gamma\}(N_n)$. После чего эта формула передается машине $\mathscr{L}_2$, которая последовательно преобразует ее в другие разнообразные формулы. Каждая формула, в которую она может быть преобразована, обязательно появится, и каждая, как только она найдена, сравнивается с

$$\lambda x[\lambda x'[\{x\}(\{x\}(x'))]], \quad m.e.\ N_2,$$

и с

$$\lambda x[\lambda x'[\{x\}(x')]], \quad m.e.\ N_1.$$

Это абсолютно полные выражения для чисел 2 и 1. При построении машины для преобразования λ-выражений нам нужна абсолютная непротиворечивость в их записи, а это легче делать без использования синтаксических сокращений.

Если она совпадает с первой из них, то машина печатает символ 1 и n-ый участок заканчивается. Если она совпадает со второй, то печатается 0, и участок заканчивается. Если она отлична от обеих, то работа $\mathcal{L}_2$ возобновляется. Согласно гипотезе $\{M_\gamma\}(N_n)$, преобразуется в одну из формул N_2 или N_1; следовательно, n-ый участок обязательно завершится, *т. е.* n-ый символ γ будет обязательно записан.

Тьюринг пропускает строчку, прежде чем начать более сложное доказательство обратного утверждения: как построить λ-выражение, заключающее в себе действия определенной машины.

Чтобы доказать, что каждая вычислимая последовательность γ – λ-определимая, мы должны показать, как найти такую формулу M_γ, что для всех целых чисел n

$$\{M_\gamma\}(N_n) \text{ conv } N_{1+\phi\gamma(n)}.$$

Это – та же формула, что и раньше, но с видоизмененным индексом у последнего N. Теперь задача заключается не в описании машины, реализующей λ-функцию, а в определении λ-функции, имитирующей машину.

Пусть $\mathcal{M}$ – машина, вычисляющая γ, и давайте рассмотрим некоторое описание полных конфигураций $\mathcal{M}$, представленных своими номерами, например можно взять D.N полной конфигурации, как описано в §6.

В последующем обсуждении я собираюсь ссылаться на «номера конфигураций», представляющие собой просто последовательные целые числа 0, 1, 2, 3 и т. д., которые возрастают по ходу работы машины. Для любой определенной машины и для каждого номера конфигурации существует соответствующий описатель (Description Number или D.N – *прим. перев.*) полной конфигурации. Обычно это очень большие числа, которые включают коды символов, уже напечатанных на ленте, а также очередную m-конфигурацию.

Пусть $\xi(n)$ – D.N n-ой полной конфигурации $\mathcal{M}$.

У Тьюринга n – это то, что я называю номером конфигурации, тогда как $\xi(n)$ –описатель.

Таблица для машины $\mathcal{M}$ дает нам отношение между $\xi(n + 1)$ и $\xi(n)$ вида

$$\xi(n + 1) = \rho_\gamma(\xi(n)),$$

где ρ_γ – функция вполне определенного, хотя обычно не очень простого вида: она задается таблицей для $\mathcal{M}$.

Функция ρ_γ преобразует описатель в следующий за ним. Обычно ее аргумент – одно длинное число, а результат – другое длинное число. В сущности, эта функция должна найти внутри этой длинной последовательности шаблон чисел, соответствующий m-конфигурации и прочитанному символу, и построить следующую полную конфигурацию на основании таблицы для машины, возможно, включив в нее новый напечатанный символ и изменив m-конфигурацию и следующий прочитанный символ.

Тьюринг очень точно определяет эту функцию как «обычно не очень простого вида». В сущности, функция должна разбить описатель на отдельные цифры, чтобы проверить их. Поскольку описатель – большое десятичное число, функция может извлечь из него фрагмент любой длины, разделив его сначала на степень 10 и отбросив дробную часть, а затем разделив его на другую степень 10 и сохранив остаток. Хотя функция ρ_γ, безусловно, сложна, она вполне понятна.

ρ_γ – λ-определимая (доказательство этого я опускаю), *т. е.* существует правильная формула A_γ, такая что для всех целых чисел n

$$\{A_\gamma\}(N_{\xi(n)}) \text{ conv } N_{\xi(n+1)}.$$

По сути, A_γ – та же самая функция, что ρ_γ, за исключением того, что она выражена на языке λ-исчисления. Она преобразует Описатели в Описатели.

Пусть U означает

$$\lambda u[\{\{u\}(A_\gamma)\}(N_r)],$$

где $r = \xi(0)$;

U в начале предложения должна быть с индексом γ, потому что она связана с определенной вычислимой последовательностью. N_r – опи-

сатель полной конфигурации, с которой машина начинает работу, то есть 313. Это число соответствует описанию DAD, которое означает m-конфигурацию q_1 (DA) и просмотр пустой клетки (D). Переменная u – это номер конфигурации, то есть 0, 1, 2 и т. д. по ходу работы машины. Кажется, что u, заключенная в фигурные скобки и обозначающая функцию, не имеет смысла, но вы вскоре увидите, что она прекрасно работает.

тогда для всех целых чисел n

$$\{U_\gamma\}(N_n) \text{ conv } N_{\xi(n)}.$$

Аргумент функции U_γ – это номер конфигурации (0, 1, 2 и т. .). Тьюринг утверждает, что эта функция преобразуется в Номер Описания этой конфигурации. Давайте проверим это для конфигурации 4, которая предполагает преобразование выражения $\{U_\gamma\}(N_4)$ или:

$$\{\lambda u[\{\{u\}(A_\gamma)\}(N_{\xi(0)})]\}(\lambda xy.x(x(x(x(y))))).$$

Вместо N_r в функции U_γ я использовал $N_{\xi(0)}$ только потому, что мы помним, что индекс ссылается на описатель. Заменим u в функции выражением для 4:

$$\{\{\lambda xy.x(x(x(x(y))))\}(A_\gamma)\}(N_{\xi(0)}).$$

Теперь заменим x на A_γ:

$$\{\lambda y.A_\gamma(A_\gamma(A_\gamma(A_\gamma(y))))\}(N_{\xi(0)}).$$

Наконец, заменим y на $N_{\xi(0)}$:

$$A_\gamma(A_\gamma(A_\gamma(A_\gamma(N_{\xi(0)})))).$$

Первое применение A_γ к $N_{\xi(0)}$ дает в результате $N_{\xi(1)}$, следующее применение дает в результате $N_{\xi(2)}$ и т. д., а окончательный результат, как утверждал Тьюринг, – $N_{\xi(4)}$. Теперь видно, почему имело смысл ввести функцию u в определение U_γ: по сути дела, u объединяет вложенные функции A_γ.

[265]

Можно доказать, что существует формула V такая, что

$$\{\{V\}(N_{\xi(n+1)})\}(N_{\xi(n)}) \begin{cases} \text{conv } N_1, & \text{если при переходе от } n\text{-й к } (n+1)\text{-й полной конфигурации печатается символ 0.} \\ \text{conv } N_2, & \text{если печатается символ 1.} \\ \text{conv } N_3 & \text{в остальных случаях.} \end{cases}$$

В сущности, функция V анализирует Описатели двух последовательных полных конфигураций и определяет, были или не были напечатаны символы 0 или 1. Вот другая сложная, но осмысленная функция.

Пусть W_γ означает

$$\lambda u[\{V\}(\{A_\gamma\}(\{U_\gamma\}(u)))\}(\{U_\gamma\}(u))],$$

так, чтобы для любого целого числа n

$$\{\{V\}(N_{\xi(n+1)})\}(N_{\xi(n)}) \text{ conv } \{W_\gamma\}(N_n),$$

Слева от conv – формула, которая преобразуется в N_1, N_2 или N_3. Это очень просто показать, начав с конца, или:

$\{W_\gamma\}(N_n)$.

Заменим W_γ выражением, которое Тьюринг только что показал нам:

$\{\lambda u[\{\{V\}(\{A_\gamma\}(\{U_\gamma\}(u)))\}\{U_\gamma(u))]\}(N_n)$.

Заменим u на N_n:

$\{\{V\}(\{A_\gamma\}(\{U_\gamma\}(N_n)))\{U_\gamma\}(N_n))$.

Выражение $\{U_\gamma\}(N_n)$ преобразуется в $N_{\xi(n)}$, поэтому:

$\{\{V\}(\{A_\gamma(N_{\xi(n)}))\}(N_{\xi(n)})$.

Выражение $\{A_\gamma\}(N_{\xi(n)})$ преобразуется в $N_{\xi(n+1)}$, и вот что мы получаем в итоге:

$\{\{V\}(N_{\xi(n+1)})\}(N_{\xi(n)})$.

Благодаря этому небольшому доказательству теперь мы знаем, что $\{W_\gamma\}(N_n)$ преобразуется в N_1, N_2 или N_3 в зависимости от того, что будет напечатано при переходе от n-й к $(n+1)$-й полной конфигурации – 0, 1 или что-то другое.

и пусть Q – формула, такая что

$$\{\{Q\}(W_\gamma)\}(N_s) \text{ conv } N_{r(z)},$$

где $r(s)$ – s-е целое q, для которого $\{W_\gamma\}(N_q)$ преобразуется либо в N_1, либо в N_2.

Очевидно, что в формуле индекс у последнего N – $r(s)$, а не $r(z)$. Лишь некоторые из полных конфигураций печатают 0 или 1. Функция $r(s)$ раскрывает, какие именно. Например, если 0 или 1 были напечатаны в полных конфигурациях 1, 4, 6, 7 и т. д., то $r(1)$ возвращает 1, $r(2)$ – 4, $r(3)$ – 6, $r(4)$ – 7 и т. д.

Тогда, если M_γ означает

$$\lambda w[\{W_\gamma\}(\{\{Q\}(W_\gamma)\}(w))],$$

она будет обладать необходимым свойством†.

---

† В полном доказательстве λ-определимости вычислимых последовательностей было бы лучше модифицировать этот подход путем замены числового описания полных конфигураций описанием, которое можно легче обработать нашими инструментами. Давайте выберем определенные целые числа для представления символов и m-конфигураций машины. Допустим, что в определенной полной конфигурации числа, представляющие последовательность символов на ленте – $s_1\ s_2\ ...\ s_n$, что просматривается m-й символ и что m-конфигурация имеет номер t; тогда мы можем представить эту полную конфигурацию формулой

$$[[N_{s1}, N_{s2}, ..., N_{sm-1}], [N_t, N_{sm}], [N_{sm+1}, ..., N_{sn}]],$$

где

$$[a, b] \text{ значит } \lambda u[\{\{u\}(a)\}(b)],$$
$$[a, b, c] \text{ значит } \lambda u[\{\{\{u\}(a)\}(b)\}(c)]$$

и т. д.

Во второй половине доказательства Тьюринг сам ставит себе задачу поиска формулы M_γ, такой что для всех n

$\{M_\gamma\}(N_n)$ conv $N_{1+\phi\gamma(n)}$.

Формула говорит о том, является ли n-я цифра последовательности цифрами 0 или 1. Давайте начнем с

$\{M_\gamma\}(N_n)$.

Подставим сюда формулу, которую Тьюринг только что получил для M_γ:

$\{\lambda w\ [\{W_\gamma\}(\{\{Q\}(W_\gamma)\})\}(w)]\}(N_n)$.

Заменим w на N_n:

$\{W_\gamma\}(\{\{Q\}(W_\gamma)\})\}(N_n))$.

Выражение в круглых скобках, как было показано, преобразуется в $N_{r(n)}$, таким образом:

$\{W_\gamma\}(N_{r(n)})$.

Как было показано, эта формула преобразуется в N_1, N_2 или N_3 в зависимости от того, что было напечатано в полной конфигурации $r(n)$ – 0, 1 или что-то иное. Но $r(n)$ определена как функция, возвращающая только те полные конфигурации, которые приводят к печати 0 или 1.

В сноске приведена полная конфигурация, разделенная на участки до следующего просматриваемого символа на ленте и после него. λ-выражения, предлагаемые Тьюрингом для представления этих участков ленты, могут быть довольно длинными и увеличиваться в размере с очередной полной конфигурацией.

На этом статья заканчивается.

Высший колледж,
Университет Принстона,
Нью-Джерси, США.

В более строгом доказательстве обратного утверждения Тьюринг не рассматривал подход, который он обрисовал здесь. Статья «Вычислимость и λ-определимость» была получена *Журналом символической логики* 11 сентября 1937 года, спустя менее года после его появления в Принстоне[1]. Статья начинается так:

> Для выражения точного смысла интуитивного понимания «эффективной вычислимости» в применении, скажем, к функциям положительных целых чисел было дано несколько определений. Цель настоящей статьи – показать, что введенные автором вычислимые функции тождественны λ-определимым функциям Чёрча и общерекурсивным функциям, введенным Эрбраном и Гёделем и усовершенствованным Клини. Показывается [в этой статье], что каждая λ-определимая функция вычислима и что каждая вычислимая функция – общерекурсивная.

Сначала Тьюринг показывает, что λ-определимые функции вычислимы, приведя машину – видимо, более сложную, чем универсальная машина Тьюринга, – которая может анализировать и преобразовывать λ-функции.

Во второй половине доказательства говорится, что вычислимые функции рекурсивны. У Тьюринга не было необходимости показы-

[1] Alan Turing, «Computability and λ-Definability», *The Journal of Symbolic Logic*, Vol. 2, No. 4 (Dec. 1937), pp. 153–163.

вать, что вычислимые функции λ-определимы, потому что Стивен Клини уже показал (в статье «λ-определимость и рекурсивность»), что рекурсивные функции λ-определимы. Таким образом, все три определения эффективной вычислимости оказались равносильными.

Позже Тьюринг нередко упоминал те удивительные воображаемые машины, что он изобрел, лежа летом 1935 года на Грантчестерских лугах, но ни разу так и не привел реальных таблиц машины ни в одной из опубликованных статей. Когда он под руководством Чёрча написал свою докторскую диссертацию[1], в ней были только рекурсивные функции и λ-функции.

[1] Alan Turing, «Systems of Logic Based on Ordinals», *Proceedings of the London Mathematical Society*, 2nd Series, Vol. 45, No. 1 (1939), 161–228.

Глава 16

Постижение континуума

Действительность часто намного запутаннее и гораздо сложнее, чем история, которая пытается разложить ее по полочкам в форме предложений и параграфов. Историкам нужно сгладить острые углы, пренебречь малозначимыми персонажами и избежать уводящих в сторону отступлений. Такое упрощение иногда искажает историю в такой же степени, в какой пытается ее осветить. Получающаяся в итоге череда событий может показаться фатально неизбежной, как будто бы не могло случиться ничего, что бы заставило их идти иначе, и даже предполагается, что эти события привели к лучшему из возможных исходов. Иногда результат приводит к тому, что британский историк Герберт Баттерфилд (1900–1979) назвал «вольным истолкованием истории» в отношении тех авторов XIX века, которые изображали историю Британской империи как неуклонное движение вперед к современной парламентской демократии.

История науки, математики и техники особенно подвержена вольным истолкованиям. Унаследовав «правильные» научные теории и «точные» технологии, мы можем вытянуть из недр истории цепочку причинно-следственных связей, которые привели к этому непреложному результату. Допущенные неверные шаги скрадываются, а если обсуждаются исторические разногласия или вражда, то они всенепременно заканчиваются победой над теми, кто пытался препятствовать движению вперед к прекрасному мгновению, которое все мы теперь должны восхвалять.

Так, например, предысторию машины Тьюринга интересно представить чередой связанных интеллектуальных успехов – от Кантора и Фреге через Рассела и Гильберта к Гёделю, Чёрчу и Тьюрингу – достигающих своей высшей точки в одной математической статье, опубликованной в 1936 году. Сохранить такую книгу разумно короткой и содержательной – это именно то, что я сделал.

В процессе ее написания я пренебрег некоторыми расхождениями во взглядах. История математики, как любой другой интеллектуальной деятельности, густо усеяна острыми спорами и разногласиями[1]. В конце XIX и в течение XX века эти споры нередко затрагивали философию математики и довольно часто природу бесконечности.

Философия математики – обширная и сложная область, но, наверное, самый основной ее вопрос – и простой, и тревожащий:

Насколько существование математических объектов независимо от людей, их изучающих?

Математики просто открывают математические модели, которые уже существуют внутри самой материи вселенной, почти так же, как астрономы открывают звезды и другие астрономические тела? Или же математики *выдумывают* математику, как инженер конструирует новый пылесос или композитор сочиняет оперу? Более образно сказал об этом замечательный популяризатор истории математики Моррис Клайн (1908–1992):

> Математика – это россыпи алмазов, скрытых в глубинах Вселенной и постепенно извлекаемых из ее недр, или это коллекция искусственных камней, созданных человеком, и тем не менее столь блестящих, что они ослепляют своим блеском тех математиков, которые уже отчасти ослеплены гордостью за свои творения?[2]

По одну сторону этих споров – *реалисты*, или *платоники* (последователи учения Платона – *прим. перев.*), которые верят в слова Роджера Пенроуза об «объективности математической истины. Платоническое бытие, как я его вижу, определено наличием объективных внешних законов, которые не зависят ни от нашего частного мнения, ни от особенностей нашей культуры»[3].

По другую сторону этих споров – *конструктивисты*, которые видят математику как исключительно человеческое изобретение. Для конструктивистов кажущаяся перманентность и трансцендентность математики – это просто иллюзия, усиленная способностью человека к образному мышлению – способностью, развившейся в нашем мозгу на протяжении миллионов лет эволюции.

---

[1] Для очень интересующихся этим см. Hal Hellman, *Great Feuds in Mathematics: Ten of the Liveliest Disputes Ever* (Wiley, 2006).

[2] Morris Kline, *Mathematics: The Loss of Certainty* (Oxford University Press, 1980), 323. (Рус. перевод: Морис Клайн. *Математика: Утрата определенности*. – М.: Мир, 1984.)

[3] Roger Penrose, The Road to Reality: A Complete Guide to the Laws of the Universe (Alfred A. Knopf, 2005), 13.

Между этими двумя крайностями лежит множество других течений, каждое со своим собственным выразительным названием и защитниками, некоторые из которых, вероятно, уже негодуют по поводу моей грубой классификации диапазона между этими двумя крайностями.

Большинство действующих математиков, наверное, поместили бы себя в платонову область этого ландшафта. Платоново понимание математики доминирует в нашей культуре и обращается к нашим инстинктам. Когда мы восклицаем «Эврика!», мы говорим «я нашел», а не «я сделал» это. Спустя более 100 лет после обращения Давида Гильберта ко II Международному математическому конгрессу мы все еще испытываем трепет от его слов:

> Хоть эти задачи могут казаться нам неприступными и хоть мы беспомощно стоим перед ними, мы тем не менее твердо убеждены, что их решение должно подчиняться конечному числу чисто логических действий... Эта убежденность в разрешимости любой математической проблемы – мощный стимул в работе. Мы слышим в себе вечный зов: вот проблема, ищи ее решение. Вы можете найти его одним только разумом, ибо в математике нет *ignorabimus* (мы не будем знать – *прим. перев.*)[1].

Это слова платоника: решения где-то есть, нам нужно лишь найти их. Даже после того как надежды Гильберта на доказательство полноты, непротиворечивости и алгоритмической разрешимости рухнули, платонические инстинкты все еще были живы. Среди платоников заметной фигурой был Курт Гёдель.

Расхождения в математических воззрениях – это вопрос не только идеологии, но еще и уместности. В основе всех математических доказательств лежат определенные базовые допущения. Однако некоторые из этих допущений принимались в мире конечных объектов и становятся проблематичными, когда применяются к бесконечным множествам.

«Помыслить о бесконечности не так просто», заметил Аристотель (384–322 до н. э.), и мы можем представить себе его учеников, согласно кивающих с серьезным видом. «Существует большое число необъяснимых следствий, допускаете ли вы или нет, что есть такая вещь, как бесконечность»[2].

Чтобы продвигаться по этой зыбкой почве, в Книге III своей *Физики* Аристотель не без пользы поделил бесконечность на *актуаль-*

---

[1] Цит. по кн. Ben H. Yandell, *The Honors Class: Hilbert's Problems and Their Solvers* (A. K. Peters, 2002), 395.

[2] Aristotle, *Physics*, translated Robin Waterfield (Oxford World's Classics, 1996), кн. III, гл. 4, стр. 65. (Рус. перевод: Аристотель. Физика. – М., 1936.)

ную, или *завершенную* (*полную, совершенную*), и *потенциальную*. Потенциальная бесконечность – это бесконечность натуральных чисел: за одним числом следует другое. Бесконечное деление чего-нибудь на все более мелкие части – тоже потенциальная бесконечность. Это процессы, которые происходят бесконечно долго и никогда не заканчиваются. «Вообще говоря, бесконечное – в наличии чего-то еще сверх уже имеющегося. То, что уже имеется, само по себе всегда конечно, но всегда прирастает еще одной частью, которая отличается от первой»[1].

Однако в космологии Аристотеля *нет* актуальной бесконечности, и он приводит несколько аргументов, почему ее не может быть. Он очень мудро замечает, что «Бесконечность оборачивается противоположностью тому, что о ней говорят. Бесконечное – это не то, что вне себя не имеет ничего, а то, что вне себя всегда имеет еще что-то»[2].

Аристотель даже не позволяет бесконечности быть плодом ума:

> Глупо доверять тому, что может вообразить человеческий разум, ведь тогда все это находится лишь в уме, а не в реальности, где есть всякие излишества и изъяны. О любом из нас можно подумать, что он много больше, чем есть, и сделать его бесконечно великим, но человек не становится великим только от того, что кто-то о нем так думает; если же он на самом деле такой и кто-то о нем так думает, то это – чистая случайность[3].

Аристотель не был платоником.

Не все признавали отторжение Аристотелем бесконечности. Философ Стефан Кёрнер (1913–2000) заметил, что концепции Аристотеля

> никогда не принимались единодушно. Философы школы Платона, включая августинских богословов (Августин Блаженный (354–430) – *прим. перев.*), всегда признавали идею бесконечного во всем сущем, вечно (continua) оно или нет. Их не беспокоила неприменимость этого понятия к ощущениям, так как математика для них была не абстракцией – еще меньше описанием – ощущений, а описанием действительности; и действительность воспринималась не чувствами, а разумом[4].

Математики часто интересовались завершенной бесконечностью и пытались работать с бесконечными процессами как с целым. Именно признание разницы между завершенной бесконечностью и потенциальной бесконечностью заставляет нас записывать

---

[1] Aristotle, *Physics*, translated Robin Waterfield (Oxford World's Classics, 1996), кн. III, гл. 6, стр. 72.

[2] *Там же*, кн. III, гл. 6, стр. 73.

[3] *Там же*, кн. III, гл. 8, стр. 76–77.

[4] Stephan Körner, «Continuity», в кн. Paul Edwards, ed., *The Encyclopedia of Philosophy* (Macmillan, 1967), Vol. 2, 205.

$$\lim_{n\to\infty}\left(1+\frac{1}{n}\right)$$

вместо

$$\left(1+\frac{1}{\infty}\right)^{\infty}.$$

Первая формула выражает предел. К чему стремится это выражение, когда n становится очень, очень и очень большим? Оно приводит к числу, которое известно нам как постоянная Эйлера, или e, и равная примерно 2,71828...

Вторая формула использует символ ∞ завершенной бесконечности и в итоге совершенно непонятна.

Строгое определение математического предела было введено немецким математиком Карлом Вейерштрассом (1815–1897), хотя более ранние математики были близки к нему. Это понятие стало надежным математическим основанием дифференциального и интегрального исчисления. До введения понятия предела исчисление опиралось на «бесконечно малое», число, которое еще не нуль (потому что с ним все еще можно было обращаться как с конечной величиной), но достаточно близкое к нулю, чтобы в конце концов пренебречь им. В исчислениях до сих пор сохраняется пережиток этих «бесконечно малых» в виде обозначения dx.

Другой немецкий математик XIX века Леопольд Кронекер (1823–1891) имел очень строгое представление об использовании завершенных бесконечностей в математике. Кронекер был очень известен своим афоризмом «Бог создал целые числа; все остальное – работа человека»[1]. Может *показаться,* что упоминание о высшем существе – или, точнее, существование математических объектов помимо людей – ставит Кронекера в ряды платоников, но вот «все остальное» разоблачает его как строгого конструктивиста. Кронекер хотел построить всю математику на конечных конструкциях, использую-

[1] Такого высказывания в работах Кронекера не обнаружено. Впервые в печати оно появилось в 1893 году в виде «Die ganzen Zahlen hat der liebe Gott gemacht, alles andere ist Menschenwerk». См. William Ewald, ed., *From Kant to Hilbert: A Source Book in the Foundations of Mathematics* (Oxford University Press, 1996), Vol. II, 942. В своем докладе 1922 года «The New Grounding of Mathematics. First Report» Гильберт процитировал его в связи с исключительностью целых чисел: «Die ganze Zahl schuf der liebe Gott, alles andere ist Menschenwerk». См. *From Kant to Hilbert*, Vol. II, 1120.

щих конечные целые числа. У него были проблемы даже с понятием предела и определением иррациональных чисел.

Один из первых учеников Кронекера начал заниматься математическими исследованиями, которые были совершенно возмутительными – не только определяя множества бесконечных объектов, но и уверенно *считая* эти бесконечные объекты, а затем и выполняя арифметические операции над этими величинами. Кронекер возражал против таких методов и временами даже задерживал их публикацию, и в итоге он теперь больше известен своими злонамеренными и маниакальными преследованиями этого первого ученика – Георга Кантора.

На взгляд Кронекера, причины этих преследований объяснялись скорее параноидальными взглядами на мир Кантора, нежели реальными намерениями Кронекера[1]. Однако история пишется победителями. Оказалось, что теория множеств Кантора и его деление множеств на счетные и несчетные были чрезвычайно полезными, тогда как Кронекер с его, по большому счету, бесполезными теоремами для истории математики закончился.

Понятие трансфинитных чисел у Кантора, пусть даже чуть «хромающее», – в высшей степени платоновское. Вот письмо Кантора 1883 года:

> Мы можем говорить о реальности или существовании целых чисел, как конечных, так и бесконечных, в *двух* смыслах... Во-первых, мы можем считать целые числа реальными, поскольку по определению они занимают вполне определенное место в нашем сознании... Но тогда и реальность может приписываться числам в том смысле, что они должны восприниматься как отражение или копия событий и связей во внешнем мире, который противостоит интеллекту.
> ...На основании моей вполне реалистичной, но в то же время не менее идеалистичной точки зрения, я не сомневаюсь, что эти два вида реальности всегда идут рядом в том смысле, что понятие, определенное в первом случае как всегда существующее, также обладает в определенных, даже в бесконечно многих, случаях преходящей реальностью... У этого соединения обоих фактов есть своя истинная основа в *единстве всего, к чему мы сами принадлежим*[2].

В предыдущих главах я обсуждал логицизм Бертрана Рассела (идущий от Фреге и Пеано) и формализм Давида Гильберта. В на-

---

[1] Harold Edwards, «Kronecker's Place in History», в кн. William Aspray and Philip Kitcher, eds., *History and Philosophy of Modern Mathematics* (University of Minnesota Press, 1988), 139–144.

[2] Georg Cantor, «Foundations of a General Theory of Manifolds: A Mathematico-Philosophical Investigation into the Theory of the Infinite», в кн. *From Kant to Hilbert*, Vol. II, pgs. 895–6.

чале XX столетия этим течениям противостояло другое направление (или философия). Оно было названо *интуиционизмом* и пошло от голландского математика Лейтцена Эгберта Яна Брауэра (1881–1966).

Мрачный и пессимистичный, склонный к мистицизму Брауэр грозно появляется на заре XX века как суровый учитель, пораженный хаосом, царящим вокруг него. Ученик Брауэра Вальтер П. ван Стигт описывает взгляд Брауэра на жизнь как «смесь романтического пессимизма и радикального индивидуализма». В раннем трактате, названном *Жизнь, искусство и мистицизм* (1905), Брауэр «выступает против промышленного загрязнения и господства человека над природой посредством своего интеллекта и установленных им общественных институтов и призывает к возврату к “Природе” и к мистическому и уединенному созерцанию»[1].

Брауэр посещал, а затем обучался в университете Амстердама. Хотя его диссертация была посвящена основаниям математики (предвещая его более поздние интересы), большая часть его ранних работ была из области топологии.

Брауэр ввел термин «интуиционизм» для обозначения своей идеи о том, как представляются математические сущности в голове. Все они – предметы размышлений, а их символическое представление на бумаге – неизбежное зло для передачи этих мыслей от одного человека другому. Формализм, напротив, фокусируется скорее на манипуляциях с символами, которые имеют место лишь на бумаге, а это – не более чем игра по бессмысленным правилам.

С началом оформления планов Рассела и Гильберта стало ясно, что труды Кантора должны стать общепризнанными. На взгляд же Брауэра (как и Анри Пуанкаре), широкое применение теории множеств Кантора и трансфинитных чисел должно было привести лишь к математическим катастрофам.

Брауэр не был всецело против понятия бесконечности. Он соглашался с идеей бесконечных множеств, но только если эти множества конструируемы и счетны – то есть могут находиться во взаимнооднозначном отношении с целыми числами. Уже в 1913 году Брауэр подчеркивал, что «интуиционист признает только существование счетных множеств... алеф-нуль – единственная бесконечная мощ-

---

[1] Walter P. van Stigt, «Brower’s Intuitionist Program», в кн. Paolo Mancosu, ed., *From Brouwer to Hilbert: The Debate on the Foundations of Mathematics in the 1920s* (Oxford University Press, 1998), 5.

ность, существование которой признают интуиционисты»[1]. Множество вещественных чисел должно быть запрещено именно потому, что его элементы – несчетные. Единственный способ определить множество вещественных чисел – заявить, что множество содержит все вещественные числа. Нельзя показать несколько первых чисел с многоточием или определить некоторое правило их принадлежности множеству. Нет такого правила; не существует последовательности; множество нельзя сконструировать; следовательно, такого множества быть не может. Поэтому большая часть теории трансфинитных чисел Кантора просто «бессмысленна для интуициониста»[2].

С 1918 по 1928 год Брауэр публиковал статьи с интуиционистской критикой формализма, а также статьи с попытками предложить новые основания математики, свободной от проблем и парадоксов. В частности, Брауэр нашел ошибочным закон исключенного третьего, согласно которому нечто либо обладает определенным свойством, либо нет. Поскольку этот закон определенно относится к конечным множествам, Брауэр посчитал неразумным применять его к бесконечным множествам.

В одном известном примере[3] Брауэр взялся за общепринятое мнение, что предел сходящейся последовательности всегда меньше нуля, равен нулю или больше нуля. (Это связано с законом исключенного третьего в том смысле, что предел либо меньше нуля, либо не меньше нуля.)

Вот определение последовательности:

$$c_n = \left(-\frac{1}{2}\right)^n \quad \text{для } n < k,$$

$$c_n = \left(-\frac{1}{2}\right)^k \quad \text{для } n \geq k.$$

Несколько первых значений последовательности:

$$-\frac{1}{2},\ \frac{1}{4},\ -\frac{1}{8},\ \frac{1}{16},\ -\frac{1}{32},$$

---

[1] L. E. J. Brouwer, «Intuitionism and Formalism», *Bulletin of the American Mathematical Society*, Vol. 20 (1913), 81–96.

[2] Там же.

[3] L. E. J. Brouwer, «On the Significance of the Principle of Excluded Middle in Mathematics, Especially in Function Theory» (1923), в кн. Jean van Heijenoort, ed., *From Frege to Gödel: A Source Book in MathematicalLogic, 1879–1931* (Harvard University Press, 1967), 337.

поэтому последовательность явно сходится к нулю, когда n меньше k. Но если n больше k, то все остальные значения cn равны

$$\left(-\frac{1}{2}\right)^k,$$

и это то значение, к которому сходится последовательность.

Здесь есть ловушка: значение k – это номер позиции первой цепочки цифр 0123456789, которая появляется в десятичном представлении числа π.

Так к какому же значению сходится cn – меньшему нуля, большему нуля или к самому нулю? Все зависит от k – либо оно четно, либо нечетно, либо ни то, ни другое. Платоник утверждал бы, что предел последовательности cn – реально существующее число, даже если мы не знаем, каково оно. Конструктивист возразил бы на это: раз этот предел невозможно построить, его не существует. Он не определен. Он проваливается в трещину закона исключенного третьего.

Однажды Брауэр читал лекцию об этой неопределенной последовательности, и кто-то заметил, что хотя *мы* не можем знать, как сходится cn, Бог, конечно же, знает. «У меня нет прямой связи с Богом», – ответил Брауэр[1].

Теперь мы знаем, к чему фактически сходится последовательность Брауэра, хотя это стало известно лишь спустя около трех десятилетий после его смерти[2]. Цепочка цифр 0123456789 появляется на 17 387 594 880-ой цифре десятичного представления числа π, поэтому cn сходится к $2^{-17\,387\,594\,880}$. Теперь, когда первоначальная конструкция Брауэра развалилась, нетрудно придумать другой критерий для k. Давайте переопределим k как позицию цепочки из миллиона идущих подряд семерок в десятичном представлении числа π. Поскольку цифры числа π появляются случайным образом и распределены равномерно, то миллион семерок, возможно, когда-нибудь да появятся. (А может быть, нет. Хотя многие математики допускают, что в десятичном представлении числа π может появляться любая цепочка цифр, этого никогда не было доказано. Некоторые предполагаемые последовательности цифр в числе π просто не могут быть найдены без привлечения ресурсов, превышающих всю вселенную.)

---

[1] Constance Reid, *Hilbert* (Springer, 1970, 1996), 184.

[2] Jonathan Borwein, The Brouwer-Heyting Sequence, http://www.cecm.sfu.ca/~jborwein/brouwer.html.

Как следствие неприятия закона исключенного третьего для бесконечных множеств Брауэр также отвергал правомерность некоторых доказательств *методом от противного* (*сведения к абсурду*) и даже утверждение Гильберта, что каждая математическая проблема разрешима!

Это коснулось и логики тоже. Закон исключенного третьего выражается в пропозициональной логике как

$$-X \vee X.$$

В классической логике Уайтхеда и Рассела, а также Гильберта эта формула равносильна

$$X \rightarrow X,$$

а обе они равносильны:

$$-(X \& -X).$$

Импликация $X \rightarrow X$ – символическое представление древнего философского принципа тождества («все есть то, что оно есть»), тогда как последняя из этих трех формул означает принцип противоречия: нечто не может в одно и то же время, обладать неким свойством и не обладать им. В логике Аристотеля все они были отдельными и разными понятиями, но под «грубым инструментом» пропозициональной логики они сжались до синонимов[1].

Для Давида Гильберта ограничения, которыми Брауэр хотел сковать математическую мысль, были слишком тесными. Гильберт вовсе не собирался отказываться от своих идей, даже когда признавал необходимость некоторой предосторожности. «Мы будем тщательно исследовать те способы формирования понятий и те методы вывода, которые дают хорошие плоды; мы будем пестовать их, поддерживать их и делать их пригодными к употреблению всякий раз, когда есть малейшая надежда на успех. Никто не в состоянии увести нас от рая, который сотворил для нас Кантор»[2]. Тем не менее Гильберт пытался применить к доказательствам некоторые более строгие критерии, чтобы не употреблять в них бесконечность.

Перепалка между Гильбертом и Брауэром достигла своего предела. И в 1928 году Гильберт уволил Брауэра из редколлегии журнала

---

[1] Floy E. Andrews, «The Principle of Excluded Middle Then and Now: Aristotle and *Principia Mathematica*», *Animus*, Vol. 1 (1996), http://www2.swgc.mun.ca/animus/1996vol1/andrews.pdf.

[2] David Hilbert, «On the Infinite» (1925) in *From Frege to Gödel*, 375–376.

Mathematische Annalen. Альберт Эйнштейн, один из трех главных редакторов журнала, был против этого. Событие это стало для Брауэра горьким разочарованием, и в течение десятилетий он едва ли что-то публиковал. Погиб он в возрасте 85 лет, когда, выйдя из своего дома, был сбит автомобилем.

Неизвестно, был ли знаком Тьюринг с идеями интуиционистов до написания своей статьи о вычислимых числах. Макс Ньюмэн – профессор Кембриджа, чьи лекции по основаниям математики вдохновляли Тьюринга и кто подталкивал его к публикации статьи, – несомненно, знал Брауэра по их совместной работе в области топологии. Макс Ньюмэн был соавтором официального некролога памяти Брауэра от Королевского общества[1]. (Это был необычный некролог: он состоял из 30 страниц и включал 5 страниц библиографии трудов Брауэра.) Однако Ньюмэн написал только часть некролога о работах Брауэра в области топологии, к тому же это случилось около трех десятилетий спустя после выхода статьи Тьюринга.

Статья Тьюринга занимает необычный обособленный островок между формализмом и конструктивизмом. Конечно, его машины сводят алгоритмы к серии предопределенных манипуляций с символами на ленте, хотя его разделение вещественных чисел и вычислимых чисел – и его определение вычислимых чисел как подмножества вещественных чисел, которые действительно могут быть вычислены, – определенно имеет привкус конструктивизма. Определение Тьюрингом вычислимых чисел позже привело к появлению математической теории «вычислимого анализа» («теории вычислимости» – *прим. перев.*), параллельного классическому «действительному анализу»[2].

Очень созвучна взглядам Брауэра идея о том, что вычисление числа – это процесс, который происходит в течение длительного времени. Цифр не существует, пока машина их не вычислит, а машины Тьюринга не могут быть успешно проанализированы общим конечным процессом для точного определения того, что они могли бы сделать когда-то в будущем. Нет алгоритма, который позволит определить по описателю машины, напечатает ли она когда-нибудь 0 или 1, или напечатает ли она только конечное число нулей и единиц, или напечатает ли она когда-нибудь последовательность цифр 0123456789. Если бы такой алгоритм существовал, мы смогли бы воплотить его в ма-

---

[1] G. Kreisel and M. H. A. Newman, «Luitzen Egbertus Jan Brouwer. 1881–1966», *Biographical Memoirs of Fellows of the Royal Society*, Vol. 15 (Nov. 1969), 39–68.

[2] См., например, Oliver Aberth, *Computable Analysis* (McGraw-Hill, 1980).

шине, которая вычисляет бесконечный ряд цифр числа π. Мы смогли бы определить, напечатает ли машина когда-нибудь последовательность цифр 0123456789 (или миллион семерок подряд), и узнали ли бы, в конце концов, сходится ли к нулю последовательность Брауэра.

Само существование такого алгоритма фактически предполагало бы (платоново) наличие бесконечной последовательности цифр у числа π и у любого иррационального числа. Эти бесконечные ряды цифр существовали бы, не будучи фактически вычисленными. Однако подобного алгоритма нет; мы вынуждены вымучивать каждую цифру, чтобы узнать, какова же она.

Как вы видели, иногда Тьюрингу нравится определять число, цифры которого требуют рассмотрения других машин; выходит, что эти числа – невычислимые. Брауэр делает нечто подобное в своей статье 1921 года «Каждое ли вещественное число имеет десятичное представление?»[1], где он определяет вещественное число, цифры которого (снова) обусловлены появлением определенных цепочек в бесконечном ряду цифр числа π.

Несмотря на это любопытное сходство, я не вижу каких-либо подтверждений связи с интуиционизмом Брауэра статьи, которую Тьюринг представил Лондонскому математическому обществу в 1936 году. Работа Тьюринга и его выводы настолько необычны, что, я подозреваю, он работал, не находясь под влиянием *чьих-то* более ранних философских воззрений на математику.

Однако в конце 1936 года Тьюринг прибыл в Принстон, чтобы учиться у Алонзо Чёрча, и, наверное, впоследствии несколько расширил свой кругозор в области математических возможностей и идей, включая, видимо, и интуиционизм Брауэра.

Чёрч определенно был знаком с интуиционизмом. Получив в 1927 году степень доктора философии в Принстоне, он был два года научным сотрудником Национального центра исследований.

> Я провел год в Гарварде и год в Европе, полгода в Гёттингене, так как в то время там был Гильберт, и полгода в Амстердаме, так как я интересовался работами Брауэра, который был одним из моих консультантов… Я думаю, что он не преподавал тогда. Он был довольно стар. Обычно я приезжал поездом к его месту обитания, неподалеку от города[2].

---

[1] Перепечатана в *From Brouwer to Hilbert*, 28–35.

[2] William Aspray, The Princeton Mathematics Community in the 1930s: An Oral-History Project. Интервью с Алонзо Чёрчем в Калифорнийском университете 17 мая 1984 года, http://www.princeton.edu/~mudd/finding aids/mathoral/pmc05.htm.

Характеристика «довольно стар» – несколько натянутая: когда Чёрч давал это интервью, ему было 80 лет, а Брауэру в 1929 году – только 48. Возможно, былые сражения Брауэра действительно взяли свое.

Впоследствии оказалось, что у Чёрча был определенный интерес (но не приверженность) к идеям интуиционистов. Первая статья Чёрча о лямбда-исчислении начинается со слов: «В этой статье мы представляем ряд аксиом по основаниям формальной логики, в которых избегаем использования свободной, или действительной, переменной и в которых налагаем некоторые ограничения на закон исключенного третьего, в качестве средства избежать парадоксов, связанных с математикой трансфинитности»[1].

Интерес к интуиционизму также обнаруживается в работах ученика Чёрча Стивена Клини, который включил раздел об интуиционизме в свою книгу *Введение в метаматематику* (1952) и позже (как соавтор) в книгу *Основы интуиционистской математики* (1965). В 1953 году фотография Брауэра – найденная в Мадисоне, штат Висконсин, где Клини преподавал в это время, – появляется в статье Клини по истории теории рекурсивных функций[2].

Возможно, Тьюринг находился также под влиянием Германа Вейля, который в то же самое время работал в Институте передовых исследований. Вейль получил докторскую степень в Гёттингене при Гильберте, преподавал в университете Цюриха и в 1930 году вслед за Гильбертом вернулся в Гёттинген, но вынужден был покинуть Германию в 1933 году, поскольку его жена была еврейкой. Примерно с 1919 по 1928 год Вейль исследовал математику с точки зрения интуиционизма и никогда не терял интереса к нему.

Беглый взгляд на идеи интуиционистов Тьюринг бросает в кратком продолжении статьи, написанном им в Принстоне и содержащем некоторые исправления его статьи о вычислимых числах. Как я уже писал (на странице 63), основная статья появилась в *Трудах Лондонского математического общества*, том 42, часть 3 (от 20 ноября 1936 года) и часть 4 (от 23 декабря 1936 года). Части, вышедшие с октября 1936 года по апрель 1937 года, все вместе были опубликованы во втором выпуске тома 42.

---

[1] Alonzo Church, «A Set of Postulates for the Foundation of Logic», *The Annals of Mathematics*, 2nd Series, Vol. 33, No. 2 (Apr. 1932), 346.

[2] Stephen C. Kleene, «Origins of Recursive Function Theory», *Annals of the History of Computing*, Vol. 3. No. 1 (Jan. 1981), 62.

Следующая статья появилась в *Трудах Лондонского математического общества*, том 43, часть 7 (от 30 декабря 1937 года). Позже она была включена во второй выпуск тома 43, который содержал части, выпущенные с мая по декабрь 1937 года.

[544]

О ВЫЧИСЛИМЫХ ЧИСЛАХ ПРИМЕНИТЕЛЬНО К ENTSCHEIDUNGSPROBLEM. ИСПРАВЛЕНИЕ А. М. ТЬЮРИНГА.

В статье «О вычислимых числах применительно к Entscheidungsproblem»* автор привел доказательство неразрешимости Entscheidungsproblem для «engere Funktionenkalkül». Это доказательство содержало некоторые формальные ошибки†, которые будут исправлены здесь: в этой статье есть и некоторые другие высказывания, которые нужно исправить, хотя на самом деле в них нет ложных утверждений.

---

* *Тр. Лондонского математического общ.* (2), 42 (1936-7), 230–265.

† Автор признателен П. Бернайсу за указанные ошибки.

Эта трехстраничная статья четко делится на две части. Первая часть включает исправления формул и утверждений, которые появляются в разделе 11 статьи при доказательстве неразрешимости Entscheidungsproblem. Эти исправления уже вошли в мои комментарии в главе 14. Но ради полноты я приведу и эту часть статьи тоже и прерву ее лишь дважды.

Выражение для Inst $\{q_i S_j S_k L_{ql}\}$ на с. 260 статьи должно читаться как

$$(x,y,x',y')\Big\{(R_{S_j}(x,y)\,\&\,I(x,y)\,\&\,K_{q_i}(x)\,\&\,F(x,x')\,\&\,F(y',y))$$
$$\rightarrow (I(x',y')\,\&\,R_{S_k}(x',y)\,\&\,K_{q_l}(x')\,\&\,F(y',z)\vee[(R_{S_0}(x,z)\rightarrow R_{S_0}(x',z))$$
$$\&\,(R_{S_1}(x,z)\rightarrow R_{S_1}(x',z)\,\&\ldots\&\,(R_{S_M}(x,z)\rightarrow R_{S_M}(x',z))])\Big\},$$

$S_0, S_1, \ldots, S_M$ – символы, которые может печатать $\mathcal{M}$.

Это исправление тоже не совсем правильное. В нем пропущен квантор общности для z, который должен быть справа от члена $F(y', z)$ во второй строке и применяться к остальной части формулы. На странице 299 этой книги приведен правильный вариант.

Предложение на с. 261 в строке 33, а именно

$$\text{«Inst}\,\{q_a\,S_b\,S_d\,L\,q_c\}\ \&\ F^{(n+1)} \to (CC_n \to CC_{n+1})$$

доказуемо», ложно (даже с новым выражением для Inst $\{q_a\ S_b\ S_d\ L\ q_c\}$): мы не можем вывести, например, $F^{(n+1)} \to (-F(u, u''))$ и поэтому никогда не может использовать член

$$F(y', z) \vee [(R_{S_0}(x, z) \to R_{S_0}(x', z))\ \&\ \ldots\ \&\ (R_{S_M}(x, z) \to R_{S_M}(x', z))] \qquad [545]$$

в Inst $\{q_a\ S_b\ S_d\ L\ q_c\}$.

Здесь Тьюринг признает, что его формулировка натуральных чисел некорректна.

Для исправления этого введем новую функциональную переменную G [$G(x, y)$ должна иметь значение «x предшествует y»]. Тогда если Q – сокращение для

$$(x)(\exists w)(y, z)\{F(x, w)\ \&\ (F(x, y) \to G(x, y))\ \&\ (F(x, z)\ \&\ G(z, y) \to G(x, y))$$
$$\&\ [G(z, x) \vee (G(x, y)\ \&\ F(y, z) \vee (F(x, y)\ \&\ F(z, y)) \to (-F(x, z))]\},$$

исправленная формула $\mathrm{Un}(\mathcal{M})$ должна быть

$$(\exists u)A(\mathcal{M}) \to (\exists s)(\exists t)R_{S1}(s, t),$$

где $A(\mathcal{M})$ – сокращение для

$$Q\ \&\ (y)R_{S0}(u, y)\ \&\ \mathrm{I}(u, u)\ \&K_{q1}(u)\ \&\ \mathrm{Des}(\mathcal{M}).$$

Тогда предложение на странице 261 (строка 33) должно читаться как

$$\mathrm{Inst}\{q_a\ S_b\ S_d\ L\ q_c\}\ \&\ Q\ \&\ F^{(n+1)} \to (CC_n \to CC_{n+1}),$$

а строка 29 должна читаться как

$$r(n, i(n)) = b,\quad r(n + 1, i(n)) = d,\quad k(n) = a,\quad k(n + 1) = c.$$

Вместо слов «логическая сумма» на с. 260 в строке 15 следует читать «конъюнкция». С этими модификациями доказательство верно. $\mathrm{Un}(\mathcal{M})$ можно вставить в формулу (I) (с. 263) с $n = 4$.

Следом идет пустая строка, завершающая первую из двух частей исправления. Вторая часть касается совсем другой темы и относится к более раннему разделу основной статьи.

Некоторые трудности возникают из-за специфического способа определения «вычислимого числа» (с. 233).

Соответствующий раздел находится на странице 98 этой книги и гласит: «Говорят, что последовательность вычислима, если она может быть вычислена машиной без циклов. Число вычислимо, если оно отличается на целое число от числа, вычисленного машиной без циклов».

Употребление Тьюрингом в следующем предложении слова «интуитивный» должно пониматься в общепринятом смысле; если бы Тьюринг затронул что-то, связанное с Брауэром, он использовал бы слово «интуиционистский». Более того, из этого предложения понятно, что данная гипотеза *не* удовлетворяет требованиям интуиционизма, даже если удовлетворяет интуитивным.

Если вычислимые числа должны удовлетворять интуитивным требованиям, то мы должны иметь:

Если мы можем задать правило, которое ставит в соответствие каждому положительному целому числу n два рациональных числа a_n, b_n, *удовлетворяющих условиям* $a_n \le a_{n+1} < b_{n+1} \le b_n$, $b_n - a_n < 2^{-n}$, *то существует* вычислимое число α, *для которого* $a_n \le \alpha \le b_n$ *для каждого n.* (A)

Для интуициониста «правило» – это конструкция. Для n, начиная с нуля, 2^{-n} равняется двоичным числам 1; 0,1; 0,01; 0,001 и т. д. Таким образом, это правило связывает последовательные пары рациональных чисел, которые становятся все ближе и ближе друг к другу – одна двоичная цифра ближе по величине для каждого нового значения n. По обыкновению мы назвали бы такой процесс сходящимся рядом.

Этому может быть дано доказательство, отвечающее обычным математическим стандартам, но использующее принцип исключенного третьего.

Проблема состоит в том, что такое правило может заключать в себе нечто, что так или иначе не может быть доказано, например появление определенных, заранее неизвестных цепочек цифр в бесконечной последовательности цифр десятичного представления числа π. Вскоре Тьюринг приведет более пример в своем «фирменном» тьюринговском стиле.

С другой стороны, следующее неверно:

Существует правило, посредством которого с учетом правила формирования последовательностей a_n, b_n *в* (A) *мы можем получить* D.N. *машины, вычисляющей* α. (B)

Обратите внимание, что он подает это утверждение как ложное.

(B) ложно, по крайней мере, если мы согласны с тем, что десятичные числа вида $m/2^n$ будут всегда заканчиваться нулями, как можно видеть в этом случае.

Числа вида $m/2^n$ – подмножество рациональных чисел. Они представляют исключительный интерес в связи с машинами Тьюринга, потому что двоичное представление такого числа имеет только конечное число единиц. Например, рациональное число 12345/65536 в двоичном представлении:

0,0011 0000 0011 1001 0000...

Поскольку 65 536 – это 2^{16}, то не более 16 (в зависимости от числителя) первых двоичных цифр после запятой отличны от нуля, а остальные – нули. Любое число вида $m/2^n$, где m меньше 2^n, начинается максимум с n ненулевых двоичных цифр, за которыми следуют одни лишь нули.

Тьюринг собирается привести правило формирования a_n и b_n, но это правило основывается на последовательности, которую печатает другая машина по имени $\mathcal{N}$.

Пусть $\mathcal{N}$ – некая машина, а c_n определяется следующим образом: $c_n = \frac{1}{2}$, если $\mathcal{N}$ не печатает символ 0, пока не достигнута n-я полная конфигурация $c_n = \frac{1}{2} - 2^{-m-3}$, если первый 0 был напечатан в m-й полной конфигурации ($m \le n$). Пусть [546]

$a_n = c_n - 2^{-n-2}$, $b_n = c_n + 2^{-n-2}$.

Давайте рассмотрим пример. Предположим, что $\mathcal{N}$ – машина, которая печатает последовательность 1111101... . Вот вычисление c_n, a_n и b_n:

n	**Последовательность**	c_n	a_n	b_n
0	1	$\frac{1}{2}$	$\frac{1}{2} - \frac{1}{4} = \frac{64}{256}$	$\frac{1}{2} + \frac{1}{4} = \frac{192}{256}$
1	1	$\frac{1}{2}$	$\frac{1}{2} - \frac{1}{8} = \frac{96}{256}$	$\frac{1}{2} + \frac{1}{8} = \frac{160}{256}$
2	1	$\frac{1}{2}$	$\frac{1}{2} - \frac{1}{16} = \frac{112}{256}$	$\frac{1}{2} + \frac{1}{16} = \frac{144}{256}$
3	1	$\frac{1}{2}$	$\frac{1}{2} - \frac{1}{32} = \frac{120}{256}$	$\frac{1}{2} + \frac{1}{32} = \frac{136}{256}$
4	1	$\frac{1}{2}$	$\frac{1}{2} - \frac{1}{64} = \frac{124}{256}$	$\frac{1}{2} + \frac{1}{64} = \frac{132}{256}$

n	**Последовательность**	c_n	a_n	b_n
5	0	$\frac{127}{256}$	$\frac{127}{256} - \frac{1}{128} = \frac{125}{256}$	$\frac{127}{256} + \frac{1}{128} = \frac{129}{256}$
6	1	$\frac{127}{256}$	$\frac{127}{256} - \frac{1}{256} = \frac{126}{256}$	$\frac{127}{256} + \frac{1}{256} = \frac{128}{256}$
...				

Значение c_n всегда равно $\frac{1}{2}$ до первого нуля в последовательности. Если первый в последовательности нуль – в позиции m (в примере это 5), то c_n становится $(2^{(m+2)} - 1)/2^{(m+3)}$ для n, большей или равной m.

Тогда неравенства (A) выполняются,

Значение a_n всегда возрастает; значение b_n убывает. Абсолютное значение разности между a_n и b_n *всегда* меньше 2^{-n}. (На самом деле разность всегда равна 2^{-n-1}.)

и первый символ α есть 0, если $\mathcal{N}$ когда-либо печатает 0, и 1 – в противном случае.

Очевидно, что в этом примере предел равен 127/256, поэтому вычисленная машиной последовательность α – двоичное представление 011111110000... этого рационального числа. Если первый 0 в последовательности окажется в позиции $n = 4$, то пределом последовательности α будет 63/128, а сама последовательность – 011111100000... . Если же 0 никогда не появится в последовательности, то ее пределом будет $\frac{1}{2}$ или 100000000000....

Итак, у нас есть правило для формирования a_n и b_n, но это правило опирается на последовательность, которую печатает некоторая другая машина $\mathcal{N}$; это правило требует процедуры, чтобы определить, напечатает ли когда-нибудь подобная машина цифру 0.

Первый символ в α – это 1, если $\mathcal{N}$ никогда не печатает нуль, и 0 в противном случае, поэтому

Если (B) было истинно, то у нас должен быть способ найти первый символ α данного D.N. машины $\mathcal{N}$: *то есть* мы должны быть в состоянии определить, напечатает ли $\mathcal{N}$ когда-нибудь 0, вопреки результатам §8 указанной статьи.

«Указанная статья» – это первоначальная статья Тьюринга. Следующее утверждение шокирует и может поначалу показаться вам (как и мне) неправильным:

Таким образом, хотя из (A) следует, что должны существовать машины, которые вычисляют, скажем, постоянную Эйлера, мы не можем в настоящее время описать подобную машину, поскольку еще не знаем, имеет ли постоянная Эйлера вид $m/2^n$.

Упоминаемая здесь постоянная Эйлера – это *не* знаменитое число e, служащее основанием неперовых (натуральных) логарифмов, а чуть менее известная постоянная γ (гамма), которая вычисляется как

$$(2i-1)n+\sum_{r=1}^{\infty}(2c_r-1)\left(\frac{2}{3}\right)^r.\ \gamma=\lim_{n\to\infty}\left(\sum_{i=1}^{n}\frac{1}{i}-\int_1^n\frac{1}{x}dx\right)$$

или, более понятно, как разность между суммой и интегралом функции $1/x$

$$\gamma=\lim_{n\to\infty}\left(\sum_{i=1}^{n}\frac{1}{i}-\int_1^n\frac{1}{x}dx\right)$$

и равняется примерно 0,57721566490153286...

Эта постоянная также известна как постоянная Эйлера–Маскерони. В 1734 году Эйлер привел формулу этой постоянной, а в 1781 году вычислил 16 ее цифр. Лоренцо Маскерони (1750–1800) добился присоединения своего имени к имени постоянной, вычислив в 1790 году 32 ее цифры (хотя только первые 19 из них оказались правильными) и первым употребив для ее обозначения букву γ[1].

Тьюринг выбрал эту постоянную не случайно. Интересная особенность этой постоянной состоит в том, что никому не известно, рациональное это число или иррациональное, и если иррациональное, то алгебраическое оно или трансцендентное. Этого не знали в 1937 году и не знают до сих пор.

Приведенная ранее цитата Давида Гильберта, относящаяся к «неприступным» проблемам, на самом деле предваряется словами: «Возьмите любую явно не решаемую проблему, такую как вопрос об иррациональности постоянной Эйлера–Маскерони C или о существовании бесконечного количества простых чисел вида $2^n + 1$. Хотя эти задачи могут казаться нам...» Статус постоянной Эйлера–Маскерони считался *настолько* неприступным, что Гильберт даже не включил его в свой список 23 проблем нового столетия.

[1] Julian Havil, *Gamma: Exploring Euler's Constant* (Princeton University Press, 2003), 89.

Если постоянная Эйлера рациональна, то мог бы иметь место тот факт, что ее знаменатель есть степень двух. Если это так, то ее двоичное представление завершается бесконечной строкой нулей.

Если бы у нас была машина Тьюринга для вычисления γ, то, конечно, не существовало бы общего процесса, чтобы определить, имеет ли γ вид $m/2^n$, потому что это было бы равносильно определению того, печатает ли машина только конечное число единиц, а это невозможно. Это более чем понятно.

Тьюринг утверждает нечто гораздо более серьезное: что для определения машины Тьюринга, вычисляющей постоянную Эйлера, необходимо знать, рационально оно или иррационально, – что сама машина должна «знать», завершается γ бесконечной цепочкой нулей или нет. Это могло бы привести в удивление любого, кто действительно программировал компьютерные алгоритмы для вычисления тысяч и миллионов цифр числа γ.

И все же Тьюринг прав, и связано это с неспособностью его машин стирать цифры, после того как они напечатаны: если число имеет вид $m/2^n$, где m меньше 2^n, то мы ждем, что машина вслед за первыми n цифрами будет печатать одни нули. Машина же, вычисляющая постоянную Эйлера, *не* поведет себя так, потому что алгоритм аппроксимирует постоянную Эйлера все меньшими (но конечными) членами. Если бы постоянная Эйлера и вправду имела вид $m/2^n$, машине действительно нужно было «знать», что она вычисляет точное значение. В противном случае машина будет бесконечно приближаться к числу, чтобы «добить» его до самого конца. Любая отличная от нуля цифра вслед за первыми n цифрами – это просто ошибка (и очень проблематичная, потому что у Тьюринга она не может быть стерта), но эти ненулевые цифры также неизбежны.

Как бы высоко вы не оценили интересный анализ Тьюрингом проблемы с постоянной Эйлера, вы, вероятно, обнаружите, что его решение хуже, чем сама проблема.

Этой неприятности можно избежать, изменив способ сопоставления вычислимых чисел вычислимым последовательностям, при этом общее количество вычислимых чисел остается неизменным. Это можно сделать разными способами*, например так.

* Такое применение перекрывающихся интервалов для определения вещественных чисел изначально принадлежит Брауэру.

Если нависавший над этим разделом статьи дух интуиционизма до сих пор еще не был вам очевиден, то подтверждение тому – сноска, где упоминается определение Брауэром вещественных чисел.

Континуум вещественных чисел всегда был проблематичным понятием из-за способа объединения дискретных и непрерывных свойств. Каждое вещественное число должно представлять определенную точку континуума, хотя мы чувствуем себя весьма неудобно, говоря, что континуум – это объединение всех таких дискретных (отдельных) точек, особенно после того, как Кантор сообщил нам, что эти дискретные (отдельные) точки даже не могут быть пронумерованы.

Брауэр, уйдя от завершенной бесконечности, попытался определить вещественные числа так, чтобы сохранить и непрерывные, и дискретные свойства континуума.

Придуманный для этого Брауэром метод известен как «последовательность выбора». Последовательности выбора бывают разными, но в применении к построению вещественного числа это – потенциально бесконечные последовательности пар рациональных чисел. Каждая пара последовательности задает интервал, вложенный в предыдущий интервал. Вот, например, возможная последовательность выбора пар вложенных рациональных чисел:

[3, 4]
[3.1, 3.2]
[3.14, 3.15]
[3.141, 3.142]
. . .

В классическом смысле эта последовательность выбора сходится, и мы с восторгом замечаем, что она, похоже, сходится к числу π. Однако, говоря о последовательностях выбора Брауэра, важно избегать понятия «сходимости», потому что она подразумевает завершенную бесконечность. Каждый из элементов последовательности определяет непрерывный отрезок между двумя его конечными точками. Таким образом, последовательность выбора сохраняет идею непрерывности. Последовательность не имеет в качестве *предела* число π. Вместо него она поддерживает тип *гало* (*halo*, от греч. *γλως* – круг)[1] вокруг числа π.

---

[1] Термин пошел от Эдмунда Гуссерля (Edmund Husserl, 1859–1938). См. Mark van Atten, Dirk van Dalen, and Richard Tieszen, «Brouwer and Weyl: The Phenomenology and Mathematics of the Intuitive Continuum», *Philosophia Mathematica*, Vol. 10, No. 2 (2002), 207.

Это гало действительно становится со временем все меньше и меньше, поскольку последовательность становится длиннее, а интервалы – короче, но гало никогда не сжимается до точного неизмеримого иррационального числа.

Один ученик Брауэра следующим образом объясняет различие между последовательностями выбора и обычными пределами:

> Следует подчеркнуть, что с точки зрения интуициониста сама последовательность выбора, растущая со временем, и есть вещественное число... На конструкции Брауэра каждый очень хорошо представляет, что такое вещественное число, поскольку оно – это сама растущая последовательность. Последовательность выбора – это не метод аппроксимации вещественного числа, которое лежит в трансцендентной действительности, ожидающей достижения своей цели[1].

В интуиционистском континууме вещественные числа всегда незавершенные –незаконченные и никогда не заканчивающиеся – и имеют ненулевой размер.

Приведенная выше последовательность выбора, вероятно, генерируется по некоторому алгоритму (правилу). Следовательно, она называется «правомерной» последовательностью выбора. Существуют также «неправомерные» или «незаконные» последовательности выбора, в которых каждый элемент выбирается неким лицом (вроде математика), решающим, как дальше развивается последовательность. Математик может даже подбрасывать монету для определения следующих элементов последовательности выбора.

> Мы можем считать неправомерную последовательность выбора своего рода интуицией, даже несмотря на то, что она в некотором отношении весьма отличается от правомерной интуиции. Тем не менее она – последовательность, выполняемая во времени субъектом (или его высшим «я»), только часть которой фактически завершена. Фактически пройденный нами законченный первоначальный ее отрезок должен восприниматься как расширение горизонта намерений, ведущих к вещественному числу, и мы должны считать ее «средой свободного становления». Нужно отметить, что мы говорим здесь о последовательности выбора как о процессе, как о развивающейся во времени последовательности действий[2].

Последовательности выбора могут быть также комбинациями правомерных и неправомерных последовательностей, например при выполнении определенных арифметических операций над несколькими незаконными последовательностями.

---

[1] Mark van Atten, *On Brouwer* (Wadsworth, 2004), 31.

[2] Van Atten, van Dalen, and Tieszen, «Brouwer and Weyl: The Phenomenology and Mathematics of the Intuitive Continuum», 212.

Можно считать, что две последовательности выбора равны, если они одинаково начинаются и остаются одинаковыми некоторое время, как следующие две последовательности:

[3, 4]	[3, 4]
[3.1, 3.2]	[3.1, 3.2]
[3.14, 3.15]	[3.14, 3.15]
[3.141, 3.142]	[3.141, 3.142]

Затем интервал может сузиться, что сделает последовательности не равными, но перекрывающимися:

[3.141, 3.14175] [3.14125, 3.142]

Однако, возможно, потом последовательности снова станут равными:

[3.14125, 3.14175] [3.14125, 3.14175]

О том, что будет дальше, можно только предполагать.

Кажется, будто правомерная последовательность полностью определяется порождающим ее алгоритмом и, стало быть, представляет фактическую дискретную (отдельную) точку в континууме.

> Даже так ясно, что для правильного понимания последовательности выбора как представления точки в интуитивном континууме ее нужно рассматривать как последовательность в развитии, правомерна она или нет. В случае неправомерных последовательностей это можно легко понять, но то же самое справедливо и для правомерных последовательностей. Поскольку если, наоборот, правомерную последовательность считать законченным объектом, мы можем снова впасть в соблазн представлять точку классическим атомистическим способом. Но тогда континуум был бы разорван. Иными словами, условие, что точка не «существует», а только «становится» ею, сохраняет континуум[1].

На самом деле такой подход должен быть в некоторой степени знаком читателям статьи Тьюринга, потому что у нас есть опыты с машинами Тьюринга. Мы представляем вещественное число машиной Тьюринга (или ее описателем), но машина генерирует цифры непрерывно. Она никогда не останавливается, и мы никогда не можем точно установить по описателю машины, каким на самом деле станет это число.

В двоичном виде цифры числа $\pi/4$ таковы:

0,1100100100001111110110101010001000...

---

[1] Van Atten, van Dalen, and Tieszen, «Brouwer and Weyl: The Phenomenology and Mathematics of the Intuitive Continuum», 212–213.

(Я беру $\pi/4$, а не π, чтобы число лежало между 0 и 1.) Машина Тьюринга, вычисляющая цифры $\pi/4$, на самом деле может считаться вычислителем последовательностей выбора. Когда машина вычисляет первую цифру 1, эта цифра фактически представляет не число 0,1, а диапазон от 0,10000000... до 0,11111111..., потому что это диапазон возможных вещественных чисел, которые начинаются с 0,1. Зная, что последовательность 0,11111111... равна 1, порождаемые этой машиной вложенные последовательности выбора таковы:

[0,1; 1,0]
[0,11; 1,00]
[0,110; 0.111]
[0,1100; 0,1101]
[0,11001; 0,11010]
[0,110010; 0,110011]
[0,1100100; 0,1100110]
. . .

В этом смысле обычная машина Тьюринга, которая следует соглашениям Тьюринга (то есть никогда не стирает напечатанную цифру), генерирует последовательность выбора Брауэра, представляющую вычислимое вещественное число. Процесс длится бесконечно долго и никогда заканчивается.

Если Тьюринг, как я, видит эту изящную связь, то он не признается в этом. Очевидно, у него не было такой, как у меня, возможности прочитать трудные статьи Брауэра, расшифрованные учеными XXI века. Если Тьюринг и читал что-то из Брауэра (или получил некоторые из его идей из вторых рук), то это могло быть нечто вроде статьи Брауэра 1928 года «*Die Struktur des Kontinuums*» (с нем. *Структура континуума*)[1], где последовательности выбора описываются как древовидные структуры, которые разветвляются и покрывают отрезок континуума перекрывающимися интервалами. Это могло навести Тьюринга на мысль перевести вычислимые последовательности в вычислимые числа. Помимо необходимости поставить задачу, связанную с постоянной Эйлера, Тьюринга, возможно, беспокоило и то, что его машины всегда вычисляли числа от 0 до 1, и он чувствовал, что должен выйти за эти рамки.

---

[1] Англ. перевод: L. E. J. Brouwer, «The Structure of the Continuum» в кн. *From Kant to Hilbert*, Vol. II, 1186–1197.

По первоначальному замыслу Тьюринга вычислимая последовательность становится вещественным числом простым приписыванием к нему двоичной запятой. Исправленная версия гораздо сложнее:

> Допустим, что первый символ вычислимой последовательности γ есть i и что за ним следует 1, повторяющаяся n раз, затем 0 и, наконец, последовательность, r-й символ которой – c_r; тогда последовательность γ должна соответствовать вещественному числу
>
> $$(2i-1)n+\sum_{r=1}^{\infty}(2c_r-1)\left(\frac{2}{3}\right)^r.$$

Символ i представляет собой знак числа: 0 – для отрицательного и 1 – для положительного. Ряд повторяющихся n раз символов 1 – это целая часть числа. Нуль необходим для завершения этого ряда; его роль – что-то вроде запятой в двоичном числе. Последующие символы образуют дробную часть числа, которая всегда имеет вид

$$\pm\frac{2}{3}\pm\frac{4}{9}\pm\frac{8}{27}\pm\frac{16}{81}\pm\frac{32}{243}\pm\ldots,$$

где символы c_r определяют знак каждого из слагаемых – «плюс» (для 1) или «минус» (для 0).

Предположим, например, что машина печатает следующую последовательность:

1 111110 11011...

Несколько пробелов вставлено мною только для того, чтобы облегчить преобразование в число. Первая цифра – 1, обозначающая положительное число. Следующие пять цифр 1 завершаются нулем, поэтому целая часть есть 5. Дробная часть есть:

$$\frac{2}{3}+\frac{4}{9}-\frac{8}{27}+\frac{16}{81}+\frac{32}{243}.$$

На этом этапе процесса число равно $5\frac{278}{243}$ или $6\frac{35}{243}$. Заметьте, что целая часть вычисленного числа фактически равна 6, а не 5. Если все цифры дробной части числа суть 1, то дробная часть

$$\frac{2}{3}+\frac{4}{9}+\frac{8}{27}+\frac{16}{81}+\frac{32}{243}+\ldots$$

сходится к 2. Аналогично если все символы дробной части суть 0, то дробная часть сходится к –2. Последовательность с уже преобразованной целой частью N может на самом деле «гулять» между $N-2$ и $N+2$, создавая перекрывающиеся интервалы.

Если считать, что машина, которая вычисляет γ, вычисляет и это действительное число, то (B) выполняется.

То есть если существует правило формирования последовательностей a_n и b_n, которые приближаются к числу, мы можем получить описатель машины. Сама машина вычисляет a_n и b_n, а затем выбирает, какой символ печатать – 0 или 1, в зависимости от того, что нужно сделать – вычесть или добавить $(2/3)^n$, чтобы внести вычисленное число в пределы этого нового диапазона.

Теперь мы придерживаемся этого единственного метода вычисления чисел, основанного на сходящихся границах. Нельзя больше описывать простые машины, которые вычисляют рациональные числа вроде $\frac{1}{2}$ или $\frac{1}{4}$. Теперь эти числа должны аппроксимироваться, как все остальные. Например, последовательность для $\frac{1}{2}$ теперь такова:

0 0 1010 1101 0100 1101...

Первые две цифры представляют (положительный) знак и запятую в двоичном числе. Последующие цифры задают слагаемые 2/3, –4/9, 8/27, –16/81 и т. д. Часть приведенной здесь последовательности вычисляет $^1/_2$ с точностью, чуть большей, чем 3 десятичные цифры или 10 двоичных цифр.

Преимущество состоит в том, что машина не должна «знать», что это действительно вычисление рационального числа вида $m/2^n$. Она не должна «знать», когда прекратить вычисление и погрузиться в бесконечный ряд нулей. Даже последовательность, которая заканчивается бесконечным рядом нулей или единиц, связывается с числом, созданным по бесконечному ряду уменьшающихся, но конечных членов.

Есть другие способы вычислить $\frac{1}{2}$. Вот один из вариантов:

0 10 0101 0010 1011 0010...

Первая цифра – (положительный) знак, а следующие две цифры представляют число 1. Последующие цифры задают слагаемые –2/3, 4/9, –8/27, 16/81 и т. д., противоположные цифрам первой последовательности.

Уникальность представления вещественных чисел последовательностями символов теперь теряется, но теоретически это не так важно, так как в общем случае D.N. не уникальны.

Высший колледж,
Принстон, Нью-Джерси, США.

На этом краткая статья исправлений Тьюринга заканчивается, причем без капли сожаления о том, что этот внезапный сдвиг парадигмы чуточку сбивает нас с толку. Мы могли бы испытывать чувство некоторого удовлетворения от того, что у нас теперь есть определение вычислимых чисел, математически менее затруднительное, но едва ли оно компенсирует тошноту от качки на философски изменчивых волнах континуума.

Часть IV

И далее

Глава 17

Весь мир – машина Тьюринга?

Независимо от того, насколько хорошо вы понимаете идею и принцип действия машины Тьюринга, это на самом деле не поможет вам построить вычислительную машину. Цифровые вычислительные машины создаются из транзисторов или иных переключающих элементов вроде реле или электроламп. Эти транзисторы собираются в логические схемы, выполняющие простые логические функции, из которых затем собирают высокоуровневые компоненты, такие как регистры и сумматоры[1].

Машина Тьюринга строится из того, о чем Тьюринг ничего нам не говорит. У Тьюринга не было намерений предложить свою машину в качестве проекта настоящего компьютера. Его машина – упрощенная абстрактная модель вычислений, выполняемых человеком или машиной. Первоначальная и вполне определенная цель создания Тьюрингом своей машины – доказать, что не существует общей разрешающей процедуры для логики первого порядка. И только позже эти воображаемые устройства начали способствовать нашему пониманию теории вычислений. Это длилось около 20 лет, после чего машина Тьюринга стала предметом изучения дисциплины, известной нам теперь как информатика (computer science).

Чтобы приспособить машину Тьюринга для целей, отличных от первоначальных, нужно определить ее несколько иначе. Большинство машин Тьюринга тратит бесконечное время на вычисление цифр некоторого вещественного числа от 0 до 1. Тогда как обычно в математике, как и в программировании, вычисляют функции. Функция требует на входе одно или более чисел, называемых еще аргументами функции. На основании этого входа функция вычисляет результат, называемый значением функции.

---

[1] Эта иерархия описана в кн. Charles Petzold, *Code: The Hidden Language of Computer Hardware and Software* (Microsoft Press, 1999).

Существует один важный класс функций – теоретико-числовые функции, называемые так потому, что и вход, и выход представляют собой натуральные числа. В разделе 10 своей статьи (стр. 268 этой книги) Тьюринг разработал метод вычисления теоретико-числовых функций путем печати последовательностей из символов 1, разделяемых одним символом 0. Количество подряд идущих единиц первой последовательности – значение функции для аргумента 0; количество подряд идущих единиц следующей последовательности – значение функции для 1 и т. д.

Одним из математиков, пристально и критически взглянувшим на теоретико-числовые функции машин Тьюринга, был Стивен Коул Клини. Клини был студентом Алонзо Чёрча в Принстоне и, получив в 1934 году степень доктора философии, начал преподавать в университете Висконсина в Мэдисоне.

Позже Клини писал: «Принимая в целом идею Тьюринга о возможностях его машин, я усомнился в том, что его способ их применения для вычисления теоретико-числовых функций – простейший. Во всяком случае, таким способом могла быть вычислена только полная функция $\phi(x)$»[1]. Метод Тьюринга не работает для частичных функций, то есть функций, справедливых лишь на собственном подмножестве натуральных чисел.

С весны 1941 года Клини начал искать другие подходы в рамках семинара по основаниям математики, который он вел в университете Висконсина. Позже заново определенные Клини машины Тьюринга вошли в XIII главу его – теперь уже классической – книги 1952 года *Введение в метаматематику*.

У Клини машина Тьюринга тоже читает и пишет символы и перемещается по ленте влево и вправо. Но она ограничивается только одним символом – простой вертикальной черточкой, называемой *галочкой* или счетной *палочкой*. Машина отличает этот символ от пустой клетки. Натуральные числа представляются рядами палочек в смежных клетках и отделяются друг от друга пустыми клетками. Поскольку у Клини натуральные числа начинаются с 0, одна палочка представляет 0, две палочки представляют 1 и т. д. Клини – пожалуй, первый из авторов, кто в своих текстах изобразил на рисунке ленту классической машины Тьюринга[2].

---

[1] Stephen C. Kleene, «Origins of Recursive Function Theory», *Annals of the History of Computing*, Vol. 3, No. 1 (Jan. 1981), 61.

[2] Stephen C. Kleene, *Introduction to Metamathematics* (D. Van Nostrand, 1952), 358–360. (Рус. перевод: Клини С. К. Введение в метаматематику: пер. с англ. – М.: ИЛ, 1957.)

Обычно машины Тьюринга начинаются с пустой ленты. Усовершенствованные Клини машины начинаются с ленты, на которую уже записан вход (аргументы) функции в виде одного или более разделенных пробелами рядов палочек. После чего машины Клини вычисляют значение функции и записывают это число на ленту. В качестве первого примера Клини приводит функцию преемника, которая вычисляет следующее за записанным на ленте число; она совершает этот ошеломительный подвиг простым приписыванием еще одной палочки к уже существующему ряду палочек.

Машины Клини, вычисляющие функции, требуют лишь конечного времени для вычисления, поэтому, закончив, машина может остановиться. Специальной конфигурации «стоп» или «останов» у нее нет, но есть то, что Клини называет «пассивным положением», когда у машины нет возможности продолжать дальше. Если по команде машины она попадает в несуществующую конфигурацию, «то говорят, что машина *останавливается*. Положение, в котором она остановилась, мы называем *конечным положением* или *результатом*»[1].

В понимании Тьюринга хорошая машина, которую Тьюринг называет ациклической машиной или удовлетворительной машиной, никогда не останавливается. В новой формулировке Клини хорошая машина заканчивает вычисление функции и затем останавливается. Машина Клини, которая попадает в бесконечный цикл и никогда не останавливается, – плохая машина. В этом отношении машины Клини, очевидно, гораздо ближе по смыслу к обычной математической и компьютерной функции, вычисляющей выход по входу за конечное число шагов.

Как сказано в главе 15, к 1936 году существовали три различные формулировки интуитивного понятия эффективной вычислимости:

- машины Тьюринга;
- рекурсивные функции в определении Курта Гёделя, данном в 1934 году на основе предположения Жака Эрбрана, а затем исследованном Клини; и
- λ-определимые функции, разработанные Чёрчем и его студентами, а более других – Клини.

Равносильность этих трех формулировок была установлена Тьюрингом частично в приложении к его статье о вычислимых числах и более строго в его статье 1937 года «Вычислимость и λ-определимость»,

---

[1] Stephen C. Kleene, *Introduction to Metamathematics* (D. Van Nostrand, 1952), 358.

а также Стивеном Клини в его статье 1936 года «λ-определимость и рекурсивность». С тех пор термины «рекурсивная функция» и «вычислимая функция» стали почти синонимами.

Стивен Клини был первым, кто придумал название для описания того, как эти формализации соотносятся с интуитивным понятием вычислимости. Именно во *Введении в метаматематику* Клини впервые формулирует то, что явно называет *тезисом Чёрча*: «*Каждая эффективно вычислимая функция* (*эффективно разрешимый предикат*) *является общерекурсивной*». Двумя главами позже Клини говорит: «тезис Тьюринга о том, что любая функция, которая считается интуитивно вычислимой, вычислима согласно его определению, то есть посредством одной из его машин, равносилен тезису Чёрча...»[1].

В книге, вышедшей в 1967 году, Клини объединил эти два тезиса в один:

> Тезисы Тьюринга и Чёрча равносильны. Мы будем, как правило, называть их либо *тезисом Чёрча*, либо – в связи с тем, что один из трех его вариантов имеет дело с «машинами Тьюринга», – *тезисом Чёрча–Тьюринга*[2].

С тех пор предпочтение отдавалось термину «тезис Чёрча–Тьюринга».

Конечно, *Введение в метаматематику* – книга для математиков. Шесть лет спустя еще одна, теперь уже классическая книга пересекла границу между математикой и информатикой.

Мартин Дэвис родился в 1928 году в Нью-Йорке. В 1950 году в Принстонском университете он получил степень доктора философии за диссертацию *К теории рекурсивной неразрешимости*. Консультантом диссертации Дэвиса был Алонзо Чёрч, который был также консультантом диссертаций Клини в 1934 году и Тьюринга в 1938 году.

Возможно, уже в начале 1952 года в курсе, который Дэвис вел в университете Иллинойса, он начал говорить о проблеме завершения вычислений машиной Тьюринга как о «проблеме останова»[3]. Этот термин стал более известным после выхода в 1958 году книги Дэвиса *Вычислимость и неразрешимость*. В предисловии к книге Дэвис шутливо замечает: «Несмотря на то, что в этой книге мало

---

[1] Kleene, *Introduction to Metamathematics*, 300, 376.

[2] Stephen Cole Kleene, *Mathematical Logic* (John Wiley & Sons, 1967; Dover, 2002), 232. (Рус. перевод: Клини С. К. Математическая логика: пер. с англ. – М.: Мир, 1973.)

[3] См. B. Jack Copeland, *The Essential Turing* (Oxford University Press, 2004), 40, сноска 6.

существенно нового, опытный читатель, наверное, обнаружит в ней некоторую новизну в размещении и истолковании некоторых тем», а затем поясняет: «в частности, машина Тьюринга занимает в ней центральное место»[1].

Если у Клини во *Введении в метаматематику* машина Тьюринга не упоминается вплоть до страницы 321 и почти не участвует в обсуждении вплоть до главы 13, в *Вычислимости и неразрешимости* Дэвиса машина Тьюринга выносится в самое начало – на первую страницу первой главы.

Дэвис, подобно Клини, представляет натуральные числа рядами палочек и использует машины для вычисления функций. Примеры машин, выполняющих сложение, вычитание и умножение, начинаются уже с 12-й страницы.

Хотя *Вычислимость и неразрешимость* – как будто бы учебник математики, Дэвис понимал, что книга «затрагивает некоторые философские вопросы и теорию цифровых машин и потому [представляет] потенциальный интерес для нематематиков»[2].

Отметим еще одно отличие: книга *Вычислимость и неразрешимость* вышла в новой серии издательства McGraw-Hill «Обработка информации и вычислительные машины». И даже для этой серии книга была уникальной. Прочие книги серии уделяли внимание «практическим» вопросам вычислительной техники и программирования. Среди наименований этой серии в 1958–1959 годах – *Аналоговое моделирование: Решение полевых задач*; *Высокоскоростная обработка данных*; *Основы цифровых машин*; *Цифровые вычислительные системы* и *Основы программирования для компьютеров*.

Вычислимость и неразрешимость Мартина Дэвиса, можно точно сказать, положила начало изучению вычислимости как предмета, который позже вошел в стандарт учебной программы по специальности «Информатика» («Computer Science»).

Именно в *Вычислимости и неразрешимости* Дэвис на странице 70 вводит термин, часто используемый в связи с машинами Тьюринга:

> Теперь пусть *Z* – простая машина Тьюринга. Мы можем связать с *Z* следующую проблему разрешимости:
> *Определить для данного моментального описания α, действительно ли существует вычисление Z, которое начинается с α.*

---

[1] Martin Davis, *Computability and Unsolvability* (McGraw-Hill, 1958: Dover, 1982), vii–viii.

[2] Davis, Computability and Unsolvability, vii.

> То есть мы хотим определить, действительно ли *Z*, начав с заданного состояния, в конце концов остановится. Мы называем эту проблему *проблемой останова* для *Z*[1].

Ниже на той же странице Дэвис формулирует теорему: «*Существует машина Тьюринга, проблема останова которой рекурсивно неразрешима*».

Влияние книги Мартина Дэвиса распространилось настолько, что проблема останова теперь всегда связывается с машинами Тьюринга, несмотря на то что оригинальные машины Тьюринга никогда не останавливаются!

За исключением быстродействия, объема памяти и интерфейсов на любой вкус пользователя, все современные компьютеры, в сущности, одинаковы. Каждый компьютер, способный эмулировать машину Тьюринга (а это очень просто), – это универсальный компьютер. Более того, любой универсальный компьютер может эмулировать любой другой универсальный компьютер.

Некоторые из самых первых компьютеров *не* были столь мощными, как машина Тьюринга. Возможно, первой вычислительной машиной, которая была – по крайней мере, *потенциально* – универсальной, была машина, созданная между 1938 и 1941 годом Конрадом Цузе и названная Z3[2]. Даже построенная в 1830-х годах Аналитическая машина Чарльза Бэббиджа должна считаться универсальной машиной, несмотря на то что она работала на шестеренках, а не на реле. Фактически все компьютеры, созданные после 1944 года, были универсальными машинами.

Ключевое свойство универсальной машины – *программируемость*. В компьютере должен существовать некоторый способ ввода и исполнения потока команд. В современных компьютерах команды – это байты в памяти, называемые машинным кодом. В машине Цузе команды кодировались отверстиями, пробиваемыми на 35-миллиметровой кинопленке. Машина Бэббиджа должна была использовать перфокарты, подобные тем, что управляли шелкоткацкими станками Жаккарда.

Некоторые ранние компьютеры могли выполнять лишь линейный поток команд. Универсальная машина должна иметь возможность обходить некоторые команды потока на основании результатов преды-

---

[1] Davis, Computability and Unsolvability, 70.

[2] Raúl Rojas, «How to Make Zuse's Z3 a Universal Computer», *IEEE Annals of the History of Computing*, Vol. 20, No. 3 (1998), 51–54.

дущих вычислений. Такая возможность известна как *условный переход*, и она важна для реализации вычислительных циклов.

Языки программирования компьютеров часто называют «полными по Тьюрингу», если синтаксис языка позволяет им имитировать машину Тьюринга.

Элементарный язык разметки гипертекста HTML, широко используемый во Всемирной паутине, не предназначен для вычислений и, безусловно, неполный по Тьюрингу. JavaScript, часто используемый в HTML, – конечно, полный по Тьюрингу, как, в сущности, все используемые сегодня языки программирования. Любой полный по Тьюрингу язык программирования может эмулировать любой другой полный по Тьюрингу язык.

Машина Тьюринга не только определила основные требования к эффективной вычислимости, но и установила ее пределы: ни один из известных сегодня компьютеров или языков программирования не мощнее машины Тьюринга; ни один компьютер или язык программирования не может решить проблему останова; ни один компьютер или язык программирования не может предсказать судьбу другой компьютерной программы. За эти пределы не поможет вырваться ни «наилучший» язык программирования, ни какая-то другая машина. В лучшем случае можно лишь быстрее выполнять работу. Мы можем собрать тысячи процессоров, работающих параллельно, можем приложить усилия для создания квантовых компьютеров, которые осуществляют сугубо параллельные вычисления, но так и не сможем хоть чуточку приблизить этот безнадежно конечный мир, в котором живем, к бесконечности.

Несмотря на ограничения машин Тьюринга, некоторые математики предпринимали путешествие в мир *гипервычислений* и описывали машины, которые преодолевали ограничения Тьюринга. Отчасти такие исследования спровоцировал сам Алан Тьюринг, кратко описав волшебного «оракула» в своей докторской диссертации – сложной работе 1939 года «Логические системы, основанные на ординальных числах»:

> Давайте представим, что мы обладаем некими неведомыми средствами решения [неразрешимых] теоретико-числовых задач; своего рода оракулом, как бывало прежде. Не вдаваясь в природу этого оракула, скажем только, что он не может быть машиной. С помощью оракула мы могли бы создать новый вид машины (назовем их *o*-машинами), одной из основных операций которой является решение данной теоретико-числовой задачи[1].

[1] Alan Turing, «Systems of Logic Based on Ordinals», *Proceedings of the London Mathematical Society*, Series 2, Volume 45 (1939), 172–173.

Наверно, всем нам хотелось бы иметь в жизни маленького оракула, помогающего нам в решении по-настоящему трудных вопросов. Исследователи, изучающие гипервычисления, опираясь на идею оракула, добавляли в машины Тьюринга новые возможности так, чтобы они не замыкались в рамках действительности. Будучи интересными математическими построениями, гиперкомпьютеры совершенно неосуществимы на практике, потому что нарушают основные законы физики, например ускоряя время так, что каждый следующий шаг вычислений выполняется вдвое быстрее предыдущего. Мартин Дэвис дошел даже до того, что стал считать гипервычисления «мифом» и сравнивал сторонников гипервычислений с фанатами трисекции угла и изобретателями вечного двигателя[1].

По моему мнению, исследования гипервычислений больше полезны тем, что поднимают вопросы о вычислительной универсальности. Алан Тьюринг проектировал свою воображаемую машину для моделирования основных операций человеческого «компьютера» посредством механически действующего точного алгоритма. Он обнаружил, что у этих машин есть некоторые врожденные ограничения. Спустя десятилетия мы создали компьютеры, которые по вычислительным возможностям подобны машинам Тьюринга и, следовательно, имеют те же самые ограничения. Мы не видим практического пути выхода за пределы этих ограничений.

По этой причине универсальность вычисления – и по возможностям, и по ограничениям – видимо, лежит в основе любого типа деятельности, связанной с обработкой данных. Эти ограничения, скорее всего, присущи материальному миру природы так же, как законы термодинамики.

Если у машины Тьюринга есть врожденные ограничения, которые, похоже, не могут быть преодолены без нарушения законов физики, то какие именно из них присущи *естественным* механизмам, выполняющим вычислительные или логические операции? Этот вопрос становится самым главным (и, наверное, даже чуть тревожным), когда мы рассматриваем два самых важных «естественных механизма» в нашей жизни, которые нам хотелось бы исследовать таким способом, – человеческий разум и сама вселенная.

Строго говоря, тезис Тьюринга говорит только о равносильности машины Тьюринга и механических алгоритмов. Это вовсе не предпо-

---

[1] Martin Davis, «The Myth of Hypercomputation», в кн. Christof Teuscher, ed., *Alan Turing: Life and Legacy of a Great Thinker* (Springer, 2004), 195–211.

лагает, что не может быть вычислительной машины, которая способна превзойти по быстродействию машину Тьюринга, или что такие машины не нарушают некоторого известного всеобщего закона[1].

Возможно, мы что-то упустили. Возможно, есть некий таинственный физический механизм, способный выполнять какую-то сложнейшую вычислительную операцию, которая просто не может эмулироваться машиной Тьюринга. Действительно ли модель машины Тьюринга помогает нам понять человеческий разум и Вселенную? Или по глупости мы низводим известные нам сложнейшие объекты до уровня убогой машины, которая не может даже складывать должным образом?

Жизнь машины Тьюринга вне математики и вычислений началась спустя несколько лет после публикации в 1936 году статьи Тьюринга со случайной встречи Уоррена Маккаллока (1898–1969) и Уолтера Питтса (1923–1969).

Блестящий Уолтер Питтс еще мальчиком в Детройте самостоятельно изучал латынь и греческий, философию и математику, из-за чего считался в своей семье ненормальным. Когда ему было 15 лет, он сбежал в Чикаго. Бездомный Питтс проводил большую часть времени в парке, где завел знакомство со стариком по имени Берт. У него с Бертом были схожие интересы в философии и математике, и Берт предложил ему прочитать книгу профессора Чикагского университета Рудольфа Карнапа (1891–1970), видимо, *Логический синтаксис языка*, вышедшую в 1937 году. Уолтер Питтс прочитал книгу, а затем отправился в офис Карнапа, чтобы обсудить кое-какие проблемы, которые он обнаружил. Стариком по имени Берт оказался Бертран Рассел[2].

Если вы не верите в эту историю, то более правдоподобной, возможно, будет такая: когда Бертран Рассел преподавал в университете

---

[1] Краткий разбор с хорошей библиографией см. C. Jack Copeland, «The Church-Turing Thesis», Stanford Encyclopedia of Philosophy, http://plato.stanford.edu/entries/church-turing.

[2] История приписывается бывшему студенту МакКаллока Мануэлю Блюму в кн. Pamela McCorduck, *Machines Who Think: A Personal Inquiry into the History and Prospects of Artificial Intelligence*, 25th anniversary edition (A. K. Peters, 2004), 89. Блюм также рассказал эту историю в Manual Blum, «Notes on McCulloch-Pitts' *A Logical Calculus of the Ideas Immanent in Nervous Activity*» в кн. Rook McCulloch, ed., *Collected Works of Warren S. McCulloch* (Intersystems Publications, 1989), Vol. I, 31. Ее варианты is recounted в интервью с Майклом Арбибом (Michael A. Arbib) в кн. James A. Anderson and Edward Rosenfeld, eds., *Talking Nets: An Oral History of Neural Networks* (MIT Press, 1998), 218.

Чикаго, он обычно прогуливался по парку Джексона и однажды заметил молодого человека, читающего книгу Карнапа. Они разговорились, и Рассел пригласил Уолтера Питтса на кафедру Карнапа[1].

Или вот еще: когда Питтсу было 12 лет, он, преследуемый в Детройте какими-то хулиганами, укрылся в библиотеке. Когда библиотека закрылась, он оказался взаперти. Он начал читать *Principia Mathematica* Рассела и Уайтхеда, оставаясь в библиотеке в течение трех дней, а по прочтении отправил Расселу письмо, указывающее на некоторые ошибки. Когда Рассел написал ему ответ с приглашением в Кембридж, Питтс решил стать математиком[2].

Доподлинно известно лишь то, что Уолтер Питтс действительно посещал в 1938 году в Чикаго лекции Бертрана Рассела и тогда же бывал на кафедре Рудольфа Карнапа. Молодой человек произвел впечатление на Карнапа, который, желая поддержать его, давал ему учебные задания. Но он не знал даже имени Питтса и не имел возможности найти его[3].

Питтс был «застенчивым, замкнутым в себе очкариком с плохими зубами и привычкой крутить волосы, несколько нервозный со склонностью наталкиваться на разные вещи»[4]. (Позже, во время Второй мировой войны, Питтс получил группу 4F отборочной комиссии и был отнесен к «предпсихопатической» категории, но в то же время был зачислен в Манхэттенский проект и получил допуск к сверхсекретным работам[5].) После того как Карнап напал на след Питтса, почти год спустя после посещения тем его кафедры, Питтс начал изучать логику с Карнапом и посещать занятия в университете Чикаго, включая семинары, проводимые рыжебородым украинцем Николаем Рашевским (1899–1972).

Рашевский получил докторскую степень по теоретической физике в Киевском университете и в 1924 году уехал в США. Он интересовался применением математических моделей к биологическим процессам, дисциплиной, которая, опираясь на сторонние эмпири-

---

[1] Интервью с Джеком Коуэном (Jack D. Cowan) в кн. *Talking Nets*, 104.

[2] Интервью с Джеромом Леттвином (Jerome Y. Lettvin) в *Talking Nets*, 2. В кн. Jerome Y. Lettvin, «Warren and Walter», *Collected Works of Warren S. McCulloch*, Vol. II, 514–529, Питтс провел неделю в библиотеке, читая *Principia Mathematica*, но только когда она была открыта.

[3] Neil R. Smalheiser, «Walter Pitts», *Perspectives in Biology and Medicine*, Vol. 43, No. 2 (Winter 2000), 218.

[4] Smalheiser, «Walter Pitts», 22.

[5] Там же.

ческие исследования, проводила собственные лабораторные опыты. К 1934 году Рашевский даже нашел название тому, чем он занимался: *математическая биофизика*. В 1935 году он стал первым ассистентом профессора математической биофизики в университете Чикаго. Вслед за этим в 1938 году вышла его книга под названием *Математическая биофизика*, а затем, в 1939 году, – журнал *Бюллетень математической биофизики* с публикациями статей Рашевского и его последователей[1].

В 1942 и 1943 годах Питтс опубликовал три работы в *Бюллетене математической биофизики* и примерно в это же время был представлен Уоррену Маккаллоку.

Уоррен Маккаллок рос в Нью-Джерси и сначала посещал Квакерский колледж Хейверфорд (Haverford) в Пеннсильвании. Вскоре после его поступления в колледж в 1917 году учитель и философ Руфус Джонс (1863–1948) – который в это время способствовал открытию Комитета помощи друзьям Америки – спросил Маккаллока: «Кем ты собираешься стать?.. И что ты собираешься делать?» Тот ответил, что не имеет понятия, «но есть один вопрос, на который я хотел бы найти ответ. Что такое число, которое способен понять человек, и что такое человек, способный понять число?» На что Руфус Джонс смог лишь ответить: «Друг мой, ты будешь заниматься этим столько, сколько проживешь»[2].

Маккаллок посещал Йельский университет, чтобы изучить философию и психологию, в 1927 году получил степень доктора медицины в Колледже врачей и хирургов в Нью-Йорке, лечил людей с серьезными поражениями головного мозга в Лечебном центре Бельвью (Bellevue) в Нью-Йорке и работал с психически больными в Государственной больнице Роклэнд (Rockland)[3]. Вернувшись в 1934 году в Йель, Маккаллок работал с Дюссэ де Баренном (1885–1940), который первым применил метод отображения функциональных участков мозга путем введения стрихнина в открытые части мозга кошки и

---

[1] Tara H. Abraham, «Nicholas Rashevsky's Mathematical Biophysics», *Journal of the History of Biology*, Vol. 37, No. 2 (Summer 2004), 333–385.

[2] Warren S. McCulloch, «What is a Number, That a Man May Know it, and a Man, That he May Know a Number?», Collected Words of Warren S. McCulloch, Vol. IV, 1226.

[3] Большая часть биографической информации о МакКаллоке исходит из Michael A. Arbib, «Warren McCulloch's Search for the Logic of the Nervous System», *Perspectives in Biology and Medicine*, Vol. 43, No. 2 (Winter 2000), 193–216.

наблюдения за ними. В 1941 году Маккаллока переводят в Психоневрологический институт университета Иллинойса.

Маккаллок был потрясающей личностью, «был похож на Моисея; имел длинную бороду, густые брови [и] странное свечение глаз. Долгое время он и вправду напоминал сумасшедшего. У него были серые глаза, и когда в них действительно появлялись яркость и блеск, он производил неизгладимое впечатление»[1]. Маккаллок был общителен, «откупоренная на ночь бутылка [шотландского виски] была прекрасным средством его преображения»[2], и он становился замечательным рассказчиком. (Все без исключения истории об Уолтере Питтсе, повстречавшем Бертрана Рассела в Джексон-парке, пошли от Маккаллока). Маккаллок писал стихи, рассуждал о философии и щеголял своей эрудицией при каждом удобном случае.

Не совсем ясно, как Уоррен Маккаллок и Уолтер Питтс познакомились, но они быстро нашли общий язык, даже вплоть до вхождения Питтса в семейство Маккалока. Маккаллок намеревался изложить теорию работы мозга, и знание Питтсом математической логики было именно тем, в чем он нуждался. Они трудились над статьей на кухонном столе Маккалока вместе с его дочерью Тэффи, рисовавшей иллюстрации к статье[3]. Результатом первого сотрудничества Маккаллока и Питтса стала историческая статья «Логическое исчисление идей, присущих нервной деятельности», опубликованная в *Бюллетене математической биофизики* Рашевского в 1943 году.

Из исследований второй половины XIX века ученые знали, что нервная система состоит из клеток, называемых нейронами, и что эти нейроны, скорее всего, связаны в сеть. Новые исследования в XX веке показали, что эти нейроны во многом подобны переключателям, которые срабатывают, когда раздражение достигает определенного порога[4].

Эти нейроны напомнили Маккаллоку и Питтсу логические функции, поэтому они смоделировали их на базе пропозициональной логики, используя для этого систему обозначений Рудольфа Карнапа. Одним из ключевых элементов вне рамок традиционной логики была временная задержка между входом и выходом; она позволяла нейро-

---

[1] Интервью с Джеком Коуэном (Jack D. Cowan) в *Talking Nets*, 102.
[2] Arbib, «Warren McCulloch's Search for the Logic of the Nervous System», 202.
[3] Arbib, «Warren McCulloch's Search for the Logic of the Nervous System», 199.
[4] Tara H. Abraham, «(Physio)logical Circuits: The Intellectual Origins of the McCulloch-Pitts Neural Networks», Journal of the History of the Behavioral Sciences, Vol. 38, No. 1 (Winter 2002), 19.

нам собираться в цепочки так, чтобы сигналы, проходящие по сети, могли оставаться активными. Статья Маккаллока–Питтса определяет аксиомы для этой упрощенной модели, а затем переходит к доказательству нескольких теорем.

У статьи Маккаллока–Питтса было немного предшественников. Библиография включает только *Логический синтаксис языка* Карнапа, *Grundzüge der Theoretischen Logik* (читается как «*Grundüge*») Гильберта и Аккермана и *Principia Mathematica* Уайтхеда и Рассела. На 15-й странице этой 19-страничной статьи Маккалока и Питтса обнаруживается небольшое расширение этого списка, когда они делают заключение:

> во-первых, каждая такая сеть, снабженная лентой, сканерами, подключенными к афферентам, и подходящими эфферентами для выполнения необходимых моторных действий, может вычислять только те числа, которые может вычислить машина Тьюринга; во-вторых, каждое из этих последних чисел может быть вычислено такой сетью... Это представляет интерес в плане психологического обоснования определения вычислимости по Тьюрингу и его эквивалентов – λ-определимости Чёрча и примитивной рекурсивности Клини: если какое-то число может быть вычислено организмом, оно вычислимо согласно этим определениям, и наоборот[1].

(Здесь афференты – это нервные волокна, передающие возбуждения в направлении от тканей и органов к центральной нервной системе; эфференты – то же, но в обратном направлении. – *прим. перев.*)

Несколько лет спустя, в 1948 году, Маккаллок сделал связь с машиной Тьюринга более явной. Он объяснял, что нащупывал пути к разработке теории нейрофизиологии,

> и только когда увидел статью Тьюринга, я начал понимать, что иду в правильном направлении, и с помощью Питтса сформулировал необходимое логическое исчисление. Мы сделали то, что замышляли (и думаю, довольно успешно), – считать мозг машиной Тьюринга; то есть устройством, которое могло бы выполнять те функции, которые должен выполнять мозг, если он только не сбивается с истинного пути и не впадает в психоз... Восхищало то, что простейшего набора необходимых допущений было достаточно, чтобы показать, что нервная система может вычислить любое вычислимое число. И этот тип устройства, если хотите, – машина Тьюринга[2].

---

[1] W. S. McCulloch and W. Pitts, «A Logical Calculus in the Ideas Immanent in Nervous Activity», *Bulletin of Mathematical Biophysics*, Vol. 5 (1943), 129. Также в *Collected Works of Warren S. McCulloch*, Vol. I, 357. (Рус. перевод: Мак-Каллок У. С., Питтс В. Логическое исчисление идей, относящихся к нервной активности //Автоматы: пер. под ред. А. А. Ляпунова. – М.: ИЛ, 1956.)

[2] Lloyd A. Jeffress, ed., *Cerebral Mechanisms in Behavior: The Hixon Symposium* (John Wiley & Sons, 1951), 32–33.

Еще позднее (в 1955 году) Маккаллок сказал более прямо: «Питтс и я показали, что мозг – это машина Тьюринга и что любая машина Тьюринга могла бы состоять из нейронов»[1], хотя имеющихся знаний было недостаточно, чтобы применить эту эквивалентность на практике:

> На теоретический вопрос, можете ли вы создать машину, действующую подобно мозгу, ответ таков: если вы определите окончательно и однозначно, что, по-вашему, на самом деле мозг делает с информацией, то вы можете создать такую машину. Питтс и я обосновали подобное построение. Но можете ли вы сказать, что, по-вашему, делает мозг?[2]

Статья Маккаллока и Питтса, возможно, так и томилась бы во мраке математической биофизики, не привлеки она внимание двух ключевых фигур XX века в области вычислений – Норберта Винера и Джона фон Неймана.

Норберт Винер был продуктом эксперимента домашнего обучения, печально известного с тех пор, как его выдержал Джон Стюарт Милль. Позже оба этих человека написали воспоминания об опыте формирования чудо-ребенка властными предками; по версии Винера, нанесенные ему раны долгое время оставались глубокими и незаживающими. В течение многих лет он боролся с неизвестным биполярным расстройством, когда периоды блестящих исследований сменялись периодами необъяснимого гнева и убийственного отчаяния.

Винер поступил в университет Тафтса, когда ему было 11 лет, в 14 лет стал бакалавром искусств в математике, а в 18 лет стал самым молодым доктором философии Гарварда. Однако отец Винера высказался в печати, что его сын был «не столь уж одаренным» и, более того, «ленивым»[3]. Кроме того, родители Норберта Винера долго скрывали от него, что он еврей; он узнал об этом только в 15 лет.

Покинув Гарвард, Винер изучал математическую логику у Бертрана Рассела и теорию чисел у Г. Г. Харди в Кембридже, дифференциальные уравнения у Давида Гильберта в Гёттингене накануне Первой мировой войны и философию у Джона Дьюи в Колумбийском университете. В 1919 году он был принят на кафедру математики Массачусетского технологического института.

---

[1] Warren S. McCulloch, «Mysterium Iniquitatis of Sinful Man Aspiring into the Place of God», *The Scientific Monthly*, Vol. 80, No. 1 (Jan. 1955), 36. Also in *Collected Works of Warren S. McCulloch*, Vol. III, 985.

[2] Там же, 38.

[3] Flo Conway and Jim Siegelman, *Dark Hero of the Information Age: In Search of Norbert Wiener, the Father of Cybernetics* (Basic Books, 2005), 21.

В период между войнами Винер стал первопроходцем в новых областях техники связи и аналоговых вычислений. Он участвовал в проекте аналоговых вычислений Ванневара Буша в МТИ и, видимо, был увлечен работами Клода Элвуда Шеннона по теории связи. Во время Второй мировой войны Винер работал над системами противовоздушной обороны. Эти системы использовали более сложные, чем ранее, методы предсказания курса следования самолета с целью уйти от выпущенного по нему снаряда. Особый интерес у Винера вызывала идея обратной связи – возвращение информации с выхода на вход для плавной коррекции хода процесса.

Норберт Винер *не был* на первой исторической встрече физиологов, психологов и антропологов, собравшихся в отеле Бекман (Beekman) 13 мая 1942 года при поддержке Фонда Джосайи Мэйси-младшего для исследования некоторых междисциплинарных связей. Там был Уоррен Маккаллок, а также антропологи муж и жена Грегори Бэйтсон и Маргарет Мид. Винер присутствовал на первой послевоенной конференции Мэйси под названием «Механизмы обратной связи и циклические причинные системы в биологии и общественных науках»[1], где были Уолтер Питтс, Джон фон Нейман и все, кто интересовался еще какими-то работами и размышлял над тем, как их можно объединить.

В 1947 году Норберт Винер написал книгу, сводящую воедино некоторые исследования, которые обсуждались на этих конференциях. Ему понадобилось новое слово для обозначения исследований, которые охватывают связь и управление в машинах, живых существах и социальных структурах. Для этого он выбрал греческое слово *кибернетика*, означающее искусство *штурмана*, или *рулевого*, или *пилота*. Главное в управлении судном – учесть обратную связь, чтобы погасить и исправить любое отклонение от курса. Вышедшей в 1948 году книгой Винера стала его *Кибернетика, или Управление и связь в животном и машине*.

Журнал *Time* возвещал: «Только однажды наступает великий час, когда выходит научная книга, заставляющая неистово звонить во все колокола дюжину других наук. Эта книга – *Кибернетика*»[2]. По современным меркам *Кибернетика* – необычайно маленькая книжка, соединившая будоражащую воображение прозу с солидными математическими выкладками. Во введении Винер отдает дань уважения

---

[1] Conway and Siegelman, *Dark Hero*, 155.

[2] Вышел 27 декабря 1948 года, цит. по Conway and Siegelman, *Dark Hero*, 182.

многим людям, чьими работами он проникся. Это и Уоррен Маккаллок, «интересовавшийся изучением организации коры головного мозга». Это и Алан Тьюринг, «который был, наверное, первым среди тех, кто изучал логические возможности машины ради интеллектуального эксперимента». Это и Уолтер Питтс, который «был учеником Карнапа в Чикаго, а также был связан с профессором Рашевским и его школой биофизиков». Это и пионеры вычислительной техники «д-р Эйкен из Гарварда, д-р фон Нейман из Института перспективных исследований и д-р Голдстайн из университета Пеннсильвании со своими машинами ENIAC и EDVAC»[1].

Глава 5 *Кибернетики* называется «Вычислительные машины и нервная система». Винер сравнивает переключательные механизмы вычислительных машин с моделью мозга Маккаллока и Питтса:

> Примечательно, что нервные системы человека и животного, способные, как известно, работать как вычислительная система, содержат элементы, идеально приспособленные для работы в качестве реле. Эти элементы – так называемые *нейроны*, или нервные клетки. Под воздействием электрических токов они проявляют довольно сложные свойства, тогда как обычное их физиологическое поведение очень близко к принципу «все или ничего», то есть они либо спокойны, либо, когда «раздражены», проходят через ряд изменений, почти не зависящих от природы и мощности раздражителя[2].

Двумя главами позже Винер отмечает, что «осознание того, что мозг и вычислительная машина имеют много общего, может дать новые и здравые подходы к психопатологии и даже к психиатрии»[3]. Однако Винер не был слепым технократом. Он был глубоко озабочен тем, какое воздействие эта новая наука и технология могла бы оказать на людей, и в продолжение своей книги 1948 года написал книгу *Человеческое использование человеческих существ: Кибернетика и общество* (Houghton Mifflin, 1950).

Кибернетика была объектом всестороннего исследования вплоть до конца 1951 года, когда Винер внезапно и без объяснения причин порвал все связи с Уорреном Маккаллоком и с образовавшейся вокруг его притягательной личности группой кибернетиков, включая

---

[1] Norbert Wiener, *Cybernetics: or Control and Communication in the Animal and the Machine* (John Wiley & Sons, 1948; second edition, MIT Press, 1961), 12, 13, 14, 15. Номера страниц даются по 2-му изданию. (Рус. перевод: Винер Н. Кибернетика, или Управление и связь в животном и машине. – 2-е изд. – М.: Сов. радио, 1968.)

[2] Wiener, *Cybernetics*, 120.

[3] Wiener, *Cybernetics*, 144.

Питтса, писавшего свою докторскую диссертацию под руководством Винера. Этому разрыву давалось несколько объяснений. Одно состоит в том, что эмоционально уязвимый Винер не был способен оценить оттенки индивидуальности Маккаллока. В его напыщенных речах Винер иногда не мог отличить фактов от диких домыслов[1]. Другое объяснение состоит в том, что жена Винера, ревниво оберегая репутацию мужа, солгала ему, что «мальчики» Маккаллока обольстили их дочь[2].

Без связки Винер–Маккаллок кибернетика как цельная дисциплина очень пострадала. Среди тех, кто воспринял разрыв тяжелее всех, был, пожалуй, Уолтер Питтс. Он был эмоционально опустошен, прервал свои исследования и докторскую диссертацию и постепенно начал опускаться. «Он не просто пил, что свойственно талантливым людям, он синтезировал в лаборатории новые аналоги барбитуратов и опиатов и экспериментировал на себе, принимая сложные спирты»[3]. Уолтер Питтс умер в 1969 году в возрасте 46 лет от желудочного кровотечения, часто сопутствующего хроническому алкоголизму.

Но если бы даже разрыв между Винером и Маккаллоком не произошел, небесспорно, что кибернетика смогла бы выжить. Концепция широкого междисциплинарного зонтика совершенно не вписывается в американскую академическую науку, для которой ключом к успеху всегда была специализация. Несмотря на многочисленные попытки оживить идеи кибернетики, она прижилась разве что в языке массовой культуры, в словах вроде киборг (сокращение «кибернетического организма») и в популярном префиксе «кибер» в таких словах, как киберпространство, киберкафе, киберпанк и нелепом киберсексе. И даже эти киберслова в последние годы употребляются все реже, уступая место более популярной приставке «*e*-» (от *e*lectronic – *прим. перев.*).

Статья Маккаллока и Питтса о математической модели нейронной сети послужила катализатором и для Джона фон Неймана, принявшего участие в разработке нескольких оригинальных компьютерных проектов, включая EDVAC (Electronic Discrete Variable Automatic Computer, Электронная автоматическая вычислительная машина с дискретными переменными). В *Первом предварительном сообщении об EDVAC* (First Draft of a Report on the EDVAC) от 30 июня 1945 года фон Нейман описал переключательный механизм компьютера: «Каж-

---

[1] Arbib, «Warren McCulloch's Search for the Logic of the Nervous System», 201–202.
[2] Conway and Siegelman, *Dark Hero*, 222–229.
[3] Smalheiser, «Walter Pitts», 223.

дое цифровое вычислительное устройство содержит в качестве своего *базового элемента* реле с дискретной устойчивостью. Такой элемент имеет два или более различных состояний, в которых он может находиться бесконечно долго»[1]. Цитируя статью Маккаллока и Питтса, фон Нейман писал: «Стоит напомнить, что нейроны высших живых организмов – безусловно, базовые элементы в вышеупомянутом смысле»[2].

В течение следующего года фон Нейман изучал связь между живыми существами и машинами, используя греческое слово *автомат* для механизма, проявляющего признаки живого организма[3]. В письме Норберту Винеру фон Нейман очень удивлялся тому, как они неосознанно пришли к изучению человеческого мозга – пожалуй, самого сложного из всех естественных или искусственных автоматов:

> Наши мысли – полагаю, Ваши, Питтса и мои тоже – до сих пор были сосредоточены в основном на теме неврологии, а точнее на нервной системе человека и в первую очередь на центральной нервной системе. Таким образом, пытаясь понять поведение автомата и общие принципы управления им, мы немедленно выбрали без преувеличения самый сложный под этим солнцем объект... Наши – или, по крайней мере, мои – представления об автоматах в целом были бы еще более путаными, чем есть, если бы не предпринятые нами чрезвычайно смелые попытки, которые я хотел бы поставить в один ряд с совсем не неврологическим тезисом Р. [так у автора] Тьюринга[4].

По Нейману, автомат состоял из входа и выхода с некоторой обработкой между ними. В сентябре 1948 года на симпозиуме «Механизмы головного мозга в поведении» в Калифорнийском технологическом институте он сделал предварительный доклад. Его сообщение под названием «Общая и логическая теория автоматов» содержало множество сопоставлений человеческого мозга с компьютерами того времени в терминах емкости, скорости, переключателей и рассеяния энергии. Он подчеркивал необходимость развивать новый вид логики и размышлял о том, что станет одним из главных интересов фон Неймана, – о самовоспроизводящихся автоматах[5].

---

[1] John von Neumann, *First Draft of a Report on the EDVAC* (Moore School of Electrical Engineering, 1945), §4.1.

[2] Там же, §4.2.

[3] William Aspray, «The Scientific Conceptualization of Information: A Survey», *Annals of the History of Computing*, Vol. 7, No. 2 (April, 1985), 133.

[4] Письмо от 29 ноября 1946 года, из Miklós Rédei, ed., *John von Neumann: Selected Letters* (American Mathematical Society, 2005), 278.

[5] John von Neumann, «The General and Logical Theory of Automata» в кн. Lloyd A. Jeffress, *Cerebral Mechanisms in Behavior: The Hixon Symposium* (John Wiley & Sons, 1951), 1–41.

Винер шутил по поводу взглядов фон Неймана: «То, что Вы говорите о репродуктивных способностях будущего, меня очень заинтересовало... Это удачная находка для нового доклада Кинси»[1]. (Альфред К. Кинси – американский сексолог – *прим. пер.*). Для фон Неймана самовоспроизводящиеся автоматы были не шуточным вопросом. Он задавался вопросом о существовании некоторого неизвестного закона, препятствующего машине копировать себя. Таким образом, не воспроизводятся даже живые существа (но не сама ДНК), поэтому этот вопрос поднимал и некоторые интересные онтологические проблемы.

Многочисленные исследования по теории автоматов и машинам Тьюринга привели к появлению на свет в 1956 году основополагающей книги *Исследования по автоматам*. Редакторами вышедшего в издательстве Принстонского университета сборника были основатель теории связи Клод Элвуд Шеннон и пионер искусственного интеллекта и автор языка программирования ЛИСП Джон Маккарти. В сборник вошли статьи по автоматам Джона фон Неймана, Стивена Клини, пионера искусственного интеллекта Марвина Минского (р. 1927 г.), а также некоторые из первых статей по машинам Тьюринга Клода Э. Шеннона и Мартина Дэвиса.

В разгар холодной войны в начале 1950-х годов Норберт Винер и Джон фон Нейман оказались на противоположных политических полюсах. Винер был в ужасе от применения ядерного оружия против японских городов Хиросимы и Нагасаки во время Второй мировой войны. Он перестал получать правительственные деньги для исследований, а в своих статьях все чаще обращался к социальным проблемам, связанным с ростом использования новых технологий в военных и мирных целях. В Джоне фон Неймане холодная война, наоборот, разожгла антикоммунистические настроения, и он стал твердым сторонником ядерного вооружения. В 1955 году у фон Неймана был обнаружен рак кости. В 1956 году он был госпитализирован, а в следующем году умер в возрасте 53 лет. Вполне возможно, что рак был вызван воздействием радиации во время наблюдения за испытаниями атомной бомбы[2].

Джон фон Нейман оставил после себя незаконченный курс лекций, изданный в 1958 году отдельной книгой *Компьютер и мозг*. В це-

---

[1] Письмо от 10 августа 1949 года, цит. по Steve J. Heims, *John von Neumann and Norbert Wiener: From* Mathematics to the Technologies of Life and Death (MIT Press, 1980), 212.

[2] Heims, *John von Neumann and Norbert Wiener*, 369–371.

лом неудачная, книга содержит много привлекательных намеков на то, какой могла бы стать ее полная версия. Длинная незаконченная рукопись об автоматах была отредактирована и закончена Артуром В. Берксом (1915–2008) и издана в 1966 году под названием *Теория самовоспроизводящихся автоматов*.

В своих ранних исследованиях по самовоспроизводящимся автоматам фон Нейман описывал машину, которая могла бы жить в большом супе с плавающими вокруг нее запасными частями, и изучал, как такая машина могла бы собирать своих двойников из этих частей. Подобный тип автоматов стал известен как *кинематический* автомат и мог быть прообразом того, что мы обычно называем роботом.

После дискуссий со своим другом Станиславом Уламом, изучавшим рост кристаллов, Нейман решил исследовать вместо этой гораздо более простую модель автоматов, именуемых *клеточными*.

Клеточный автомат – это математическая конструкция, которая по строению напоминает клетку. Вообще говоря, клеточные автоматы могут быть многомерными, но большая часть исследований ограничивалась двумерной сеткой. Каждая клетка сетки подвержена влиянию своих соседей, как будто бы все они связаны в простую сеть. В процессе последовательных «перемещений», или «порождений», по определенным правилам клетки меняют свое состояние. Простые правила для клеточных автоматов могут зачастую приводить к сложному поведению. Джон фон Нейман работал с клетками, которые имеют 29 состояний, и доказал, что их достаточно для создания Универсальной машины Тьюринга[1].

Клеточные автоматы вырвались из академических рамок в 1970 году, когда британский математик Джон Хортон Конвей (род. 1937 г.) спроектировал простой тип клеточных автоматов, которые он назвал Игрой в Жизнь (не путать с настольной игрой с тем же названием). Автомат для Игры в Жизнь имеет простые правила: в двумерной сетке, напоминающей миллиметровку, клетка либо жива (заполнена), либо мертва (пуста). В очередном поколении клетка может менять свое состояние в зависимости от состояний восьми ее ближайших соседей. Если живая клетка окружена двумя или тремя живыми клетками, она остается живой. Если она окружена одной и менее живыми клетками, она умирает от одиночества; окруженная четырьмя или более живыми клетками, она умирает от тесноты. Мертвая клет-

[1] William Aspray, *John von Neumann and the Origins of Modern Computing* (MIT Press, 1990), 203–204.

ка, окруженная ровно тремя живыми клетками, оживает в результате непонятной формы воспроизводства.

Игра в Жизнь Конвея была представлена в нескольких колонках «Математические забавы» Мартина Гарднера в журнале *Scientific American*[1], а в 1974 году журнал *Time* уже сетовал на то, что «растущей ордой фанатиков игры потрачено впустую драгоценного компьютерного времени, возможно, уже на миллионы долларов»[2]. Конечно, в 1974 году были не *персональные* компьютеры, а коллективные универсальные ЭВМ. Сегодня в Игру в Жизнь играют главным образом на персональных компьютерах, что, вероятно, вызывало бы меньше беспокойства у журнала *Time*.

Несмотря на простые правила, автомат Игры в Жизнь порождает довольно сложные организмы. Например, можно создать организм, постоянно производящий потомство. Хоть это кажется едва ли возможным, но из таких клеточных автоматов могут быть созданы машины Тьюринга. Игра в Жизнь – это абсолютный Тьюринг[3].

Клеточными автоматами интересовался и немецкий инженер Конрад Цузе. Он родился всего на два года и один день раньше Алана Тьюринга, и в то время как Тьюринг писал свою статью о вычислимых числах, Цузе создавал вычислительную машину в квартире своих родителей в Берлине.

В 1969 году Цузе издал короткую, 74-страничную книгу под названием *Rechnen der Raum* (с нем. – *Вычислительное пространство*), которая открывает область «цифровой физики» – объяснение поведения и законов Вселенной с позиций вычислимости.

Исторически считается, что законам физики свойственна непрерывность. Казалось бы, расстояние, скорость, масса и энергия лучше всего выражаются вещественными числами и связаны дифференциальными уравнениями. С другой стороны, квантовая теория допускает, что в основе строения вселенной может лежать дискретная и цифровая природа, а непрерывная природа реального мира – лишь иллюзия. Цузе спрашивает: «*Является ли природа цифровой, аналоговой или гибридной?*». «И есть ли на самом деле какие-то основания

---

[1] Эти колонки об Игре в Жизнь позднее были собраны в книге Martin Gardner, *Wheels, Life, and Other* Mathematical Amusements (W. H. Freeman, 1983). См. также Гарднер М. Крестики-нолики: пер. с англ. – М.: Мир, 1988.

[2] Вышел 21 января 1974 года, цит. по William Poundstone, *The Recursive Universe: Cosmic Complexity and* the Limits of Scientific Knowledge (William Morrow, 1985), 24.

[3] Paul Rendell, «A Turing Machine in Conway's Game of Life», March 8, 2001.

для такого вопроса?»[1] Для исследования законов физики в цифровой форме Цузе создал «цифровые частицы», которыми он управлял по правилам клеточных автоматов. *Rechnender Raum* – очень предварительное исследование возможностей цифровой физики для моделирования вселенной, но тем не менее это – смелый шаг.

На первый взгляд трудно представить вселенную огромной вычислительной машиной. Если пренебречь относительно ничтожными формами жизни, населяющими едва ли одно небесное тело во вселенной, то большой вычислительной активности в ней не наблюдается. Неужели вселенная – всего лишь скопление летающих вокруг нас камней?

Тут полезно рассмотреть более широкую временну́ю картину. Современная космологическая модель предполагает, что вселенная – это результат Большого взрыва, случившегося 13,7 млрд лет назад. Земля образовалась около 4,5 млрд лет назад, жизнь на Земле появилась около 3,7 млрд лет назад, первые приматы – возможно, около 10 млн лет назад, а современные люди – около 200 тыс. лет назад. Конечно, происходило еще что-то, приводившее к возрастанию сложности. Сразу после Большого взрыва вселенная была совершенно однородной – воплощением простоты, – а затем начали рождаться более сложные частицы и, в конце концов, атомы и молекулы. Это движение от простого к сложному – возможно, на основе довольно простых законов вселенной – очень напоминает клеточные автоматы.

Вычислительные модели вселенной во многом обязаны своим появлением как теории связи Клода Элвуда Шеннона и Норберта Винера, так и Тьюрингу. Использование понятия энтропии для измерения информации накрепко соединило связь и термодинамику, которая за последние несколько лет была предметом нескольких интересных популярных книг[2]. Например, Демон Максвелла – этот воображаемый чертенок, изобретенный Джеймсом Клерком Максвеллом (1831–1879), который может управлять дверцей, чтобы разделить молекулы газа на быстрые и медленные и таким образом уменьшить

---

[1] Konrad Zuse, *Calculating Space*, translation of *Rechnender Raum* (MIT Technical Translation, 1970), 22 (page 16 in the German publication).

[2] Tom Siegried, *The Bit and the Pendulum: From Quantum Computing to M Theory — The New Physics of Information* (John Wiley & Sons, 2000). Hans Christian von Baeyer, *Information: The New Language of Science* (Harvard University Press, 2003). Charles Seife, *Decoding the Universe: How the New Science of* Information is Explaining Everything in the Cosmos from Our Brains to Black Holes (Viking, 2006).

энтропию, – оказывается невозможен, потому что демон поглощает *информацию* из системы.

Американский физик Джон Арчибальд Уилер (1911–2008) связал существование вселенной с наблюдениями за ней человеком. В процессе наблюдения мы задаемся вопросами типа «да-нет», получая в ответ информацию. Уилер характеризует этот процесс легко запоминающейся фразой из трех слов «It from bit» («Это от бита»):

> It from bit означает, что каждый элемент физического мира имеет в своей основе – чаще всего в самой глубине – нематериальный источник и объяснение; то, что мы называем реальностью, возникает в результате тщательного анализа задаваемых вопросов типа «да-нет» и регистрации получаемых приборами ответов; короче, все физические тела – в основе своей теоретико-информационные, а это – **сопричастная (participatory) вселенная**[1].

Предполагая, что мир сделан из информации, Уилер отвергает идею о вселенной как некоего рода машине, потому что это «должно явно или неявно предполагать супермашину, схему, устройство, чудо, которые окажутся вселенными в их бесконечном разнообразии и бесконечном числе»[2].

Совсем иной подход у Дэвида Дейча (род. 1953 г.), одного из пионеров квантовой вычислительной техники. Дейч – твердый защитник идеи «многих миров» в квантовой физике, выдвинутой американским физиком Хью Эвереттом (1930–1982). То, что мы воспринимаем как парадоксы корпускулярно-волновой двойственности квантовой физики, есть, в сущности, влияние многих миров, которые искажают квантовые явления. Известная нам вселенная – это лишь одна из возможных вселенных в абсолютной *мультивселенной* (*multiversity*).

В своей книге 1997 года *Структура реальности* Дейч пытается объяснить природу вселенной, опираясь на четыре учения:

- эпистемологию в изложении философа науки из Вены Карла Поппера (1902–1994);
- квантовую физику с позиции «многих миров» Хью Эверетта;
- эволюцию в изложении английского натуралиста Чарльза Дарвина (1809–1882) и британского биолога-эволюциониста Ричарда Докинса (р. 1941);
- вычисление, впервые введенное Аланом Тьюрингом.

---

[1] John Archibald Wheeler, «Information, Physics, Quantum: The Search for Links» (1989) в кн. Anthony J. G. Hey, ed., *Feynman and Computation: Exploring the Limits of Computers* (Perseus Books, 1999), 311.

[2] Там же, 314, но здесь Уилер цитируется по другой статье, написанной в 1988 году.

Рассуждениями о генераторах виртуальной реальности Дейч развивает то, что он называет *принципом Тьюринга*. На первый взгляд кажется, что принцип Тьюринга относится к механизмам, выполняющим вычисления: «*Существует абстрактный универсальный компьютер, в который заложены всевозможные вычисления, способные осуществить практически любой физический объект*». На самом деле Дейч наделяет такой компьютер способностью моделировать любой физический процесс. Вскоре он показывает, что эти вычисления равносильны созданию мира виртуальной реальности. Принцип Тьюринга постепенно развивается и становится еще сильнее: «*Можно построить генератор виртуальной реальности, в который заложена любая физическая среда*»[1], включая мир, в котором мы живем.

Независимый «квантовый механик» и профессор машиностроения МТИ Сет Ллойд (р. 1960) предпочитает истолковывать квантовую физику не с точки зрения многих миров, а с точки зрения «сверхъестественности», но описывает мир и его части с точки зрения вычисления и информации – «*Большой взрыв был и информационным взрывом*». Несмотря на это, Ллойд отвергает идею моделирования вселенной машиной Тьюринга. «Вселенная в основе своей – квантово-механическая, и обычному компьютеру нелегко смоделировать квантово-механическую систему»[1]. Это одна из причин, почему он считает квантовый компьютер более отвечающим задаче.

> Вселенная – это физическая система. Таким образом, она может быть эффективно смоделирована квантовым компьютером – тех же размеров, что и сама вселенная. Поскольку вселенная поддерживает квантовые вычисления и может быть эффективно смоделирована квантовым компьютером, вселенная в вычислительном отношении не более и не менее мощнее, чем универсальный квантовый компьютер... Теперь мы можем дать точный ответ на вопрос, является ли вселенная квантовым компьютером в техническом смысле. Ответ – да. Вселенная – это квантовый компьютер[3].

Одно из свойств, которым квантовые компьютеры дополняют обычную машину Тьюринга, – способность генерировать настоящие случайные числа как результат квантовых процессов.

Клеточные автоматы вновь обрели жизнь в качестве модели физических законов вселенной в работах британского физика, математика, создателя компьютерной программы Mathematica Стивена Вольфра-

---

[1] David Deutsch, *The Fabric of Reality* (Penguin Books, 1997), 132–135.
[2] Seth Lloyd, *Programming the Universe: A Quantum Computer Scientist Takes on the Cosmos* (Alfred A. Knopf, 2006), 46, 53.
[3] Там же, 54–55.

ма (Stephen Wolfram) (р. 1959), достигнув высшей точки в амбициозной, объемной и богато иллюстрированной книге 2002 года *A New Kind of Science* (*Новый вид науки*). К этому Вольфрама подтолкнули наблюдения за клеточными автоматами, которые, опираясь на простые правила, показывали высокую сложность. Он тесно связывает свои автоматы с универсальностью машин Тьюринга и описывает, как они могут моделировать физические процессы. Вольфрам не использует в своей системе квантовую механику и считает, что ему этого и не нужно, поскольку «у меня есть большое подозрение, что такого рода программы, о которых я говорил... действительно, как выяснится в итоге, проявят многие, если не все, ключевые признаки квантовой теории»[1].

В книге *Новый вид науки* Вольфрам обнаруживает вычислительную универсальность в таком множестве проявлений, что определяет Принцип вычислительной эквивалентности, который

> представляет собой новый закон природы в том смысле, что ни одна система никогда не сможет выполнить точные расчеты, более сложные, чем те, что выполняются клеточными автоматами и машинами Тьюринга... Но как же быть с вычислениями, которые мы выполняем абстрактно на компьютерах или в наших головах? Возможно ли, чтобы они были более сложными? По-видимому, нет, особенно тогда, когда нам нужны настоящие результаты, а не одни только неопределенности. Но раз вычисление должно быть точным, оно должно быть в конечном счете реализовано в виде физического процесса, и потому ему должны быть присущи те же ограничения, что и любому другому подобному процессу[2].

Как только мы убеждаемся в том, что в мире нет ничего невычислимого (на обычных или на квантовых компьютерах), то ни одна из его составляющих не может быть исключением из этого правила. Например, жизнь – это часть вычислимой вселенной, как и одно из самых известных и таинственных ее проявлений – человеческий разум.

Веками философы, биологи, психологи и даже обыкновенные люди бились над природой человеческого разума. Зачастую открыто признавая, что большинство наших физиологических функций – результаты механических, физических и химических процессов в различных органах, мы не вполне готовы свести умственную деятельность к подобными процессам. Мы чувствуем, что разум – это что-то особенное. Понятно, что мозг *что-то* делает, когда мы думаем, но следует признать, что мы не можем объяснить *всего*, что в нем происходит.

---

[1] Stephen Wolfram, *A New Kind of Science* (Wolfram Media, 2002), 538.
[2] Там же, 720, 721.

В западной культуре это положение известно как «дуализм ум–тело» и чаще всего приписывается Рене Декарту (1596–1650) и особенно его *Meditationes de Prima Philosophia* (*Размышления о первой философии*) 1641 года. Декарт полагал, что большая часть тела человека (и все так называемые низшие животные) – это в основном машина, но разум работает совсем по-другому.

В 1940-х годах дуализм начал сдавать свои позиции. Согласно неврологу и информатику Майклу А. Арбибу, в 1943 году Маккаллок и Питтс в своей статье о нейроне решили этот вопрос. Мозг имеет подходящую структуру для выполнения вычислений; следовательно, Маккаллок и Питтс «показали, что “нечто вполне законченное”, что можно логически продумать, может быть выполнено нейронной сетью. Они уничтожили дуализм»[1].

Спустя несколько лет, в 1949 году, философ Гилберт Райл (1900–1976) пошатнул позиции дуализма в своей книге *The Concept of Mind* (*Понятие сознания*), выдвинув веские доводы без ссылок на статью Маккаллока и Питтса. В наши дни дуализм – явно в забвении. Большинство исследователей мышления (философов или нейробиологов) считают, что разум – это чисто физиологическое проявление человеческого организма, главным образом нервной системы и мозга.

Неудивительно, наверное, что отторжение дуализма совпало по времени с растущим пониманием вычислений и алгоритмов. Машина Тьюринга была задумана как модель человека-вычислителя, решающего вполне определенную алгоритмическую задачу, поэтому эта связь между машиной и разумом существовала с самого зарождения автоматических вычислений. Неудивительно, наверное, и то, что одним из первых, кто изучал искусственный интеллект, был сам Алан Тьюринг, и лучше всего – в своей Умной статье 1950 года «Вычислительные машины и интеллект», в которой он придумал то, что теперь называется Тестом Тьюринга.

Стоит только отказаться от дуализма, и разум должен рассматриваться как естественное проявление мозга (наряду с остальной частью тела), а не как сверхъестественное «что-то еще». Вопреки нашему сопротивлению трудно пройти мимо таких выводов: во-первых, что разум в основе своей – машина Тьюринга с ее возможностями и ограничениями; и во-вторых, что теоретически можно создать искусственный разум.

---

[1] Arbib, «Warren McCulloch’s Search for the Logic of the Nervous System», 213.

Как выразился американский философ Дэниел Деннетт (р. 1942), «Алан Тьюринг сделал основной шаг, благодаря которому мы могли бы заменить философский вопрос Канта – как стало возможным мышление – вопросом техническим: давайте думать, как можно реализовать мышление»[1].

То, что, кажется, беспокоит нас больше всего в Тесте Тьюринга (и более, в любом предположении о том, что мозг – это компьютер), – внутренний голос нашего рассудка, который мы называем «сознанием». Сознание придает силы нашему чувству личной независимости и свободы.

Тем не менее сознание неуловимо и призрачно. Большинству из нас хотелось бы, наверное, иметь внутренний монолог сознания, продолжающийся непрерывно на протяжении всех наших дней, но в силу самой своей природы сознание становится невидимым, когда оно не работает. Большинство из нас общаются с другими людьми, полагая, что те обладают сознанием, подобным нашему собственному, и все же у нас нет уверенности, что это так, и нам трудно убедить других в наличии сознания у нас самих.

Определение того, как в нашем мозгу формируется самосознание, – это то, что австралийский философ Дэвид Чалмерс (р. 1966) называет «трудной задачей» сознания, в отличие от сравнительно легкой задачи взаимодействия мозга с нашими органами чувств.

Тест Тьюринга, который призывает машины дурачить людей, чтобы те сомневались в своих умственных способностях, неявно становится на точку зрения бихевиоризма, для которого, чтобы считать существо «разумным», не важно, что происходит в его центральном процессоре. Мы работаем с чем-то, что считается «черным ящиком». В конце концов, именно так мы взаимодействуем с другими людьми, потому что не можем убедиться в том, что у кого-то еще есть сознание. При этом даже если мы не можем отличить машину от человека, нам самим, наверное, очень сильно хочется отличаться от машины.

Мы знаем, что все, что мы думаем о компьютерах, говорит нам о том, что они, в сущности, следуют лишь определенному набору правил. Они не знают, что действуют так же, как мы.

Этот вопрос был поднят известным мысленным опытом американского профессора философии Джона Сёрля (John Searle) (р. 1932),

---

[1] Daniel Dennett, интервью в кн. Susan Blackmore, *Conversations on Consciousness: What the Best Minds Think About the Brain, Free Will, and What it Means to be Human* (Oxford University Press, 2006), 81.

названным «китайской комнатой». Человек, не знающий китайского языка, имеет книгу, которая позволяет ему, тем не менее, давать осмысленные ответы на вопросы по-китайски. Этот человек мог бы пройти Тест Тьюринга на китайском языке, совершенно не понимая ни вопросов, ни ответов[1].

Большая проблема состоит в том, что компьютеры имеют дело только с синтаксисом, тогда как люди способны понимать и семантику. У Сёрля это означает, что компьютер – каким бы сложным он ни становился – никогда не поймет того, что он делает, так, как это может человек.

Английский математический физик Роджер Пенроуз (Roger Penrose) (р. 1931) тоже уверен, что разум больше, чем просто вычислительный орган. В своих книгах *The Emperor's New Mind* (*Новый ум императора*) (1989) and *Shadows of the Mind* (*Тени разума*) (1994) Пенроуз утверждает, что сознание – за пределами вычислений, и размышляет о том, что некоторый квантовый процесс в мозгу выполняет неалгоритмические операции, которые превосходят возможности машины Тьюринга.

Пенроуз считает теорему Гёделя о неполноте особенно показательной. Будучи людьми, мы понимаем истинность выводимого Гёделем недоказуемого высказывания, тогда как ни одно вычисление не может показать эту истинность, потому что не умеет делать выводов из аксиом. Это наблюдение не ново: в своей книге 1958 года *Доказательство Гёделя* (издательство Нью-Йоркского университета) Эрнест Нэйджел (Ernest Nagel) и Джеймс Р. Ньюмен (James R. Newman) обнаружили в теореме Гёделя похожее отторжение машинного интеллекта, как и философ Джон Рэндолф Лукас (John Randolph Lucas) (р. 1929) в своем известном эссе 1961 года «Разум, машины и Гёдель»[2]. Их доводы таковы: легко справляясь с аксиоматической математикой, машины не обладают способностью к метаматематике, требующей понимания за рамками аксиом.

Дэниел Деннетт – видимо, более других сочетающий в себе глубину мысли философа с опытом ученого для создания портрета разума в таких увлекательных книгах, как *Сознание объясненное* (1991) – думает

---

[1] John R. Searle, «Minds, Brains, and Programs» из *The Behavioral and Brain Sciences, Vol. 3* (Cambridge University Press, 1980). Перепечатано в Douglas R. Hofstandter and Daniel Dennett, eds., *The Mind's I: Fantasies and Reflections on Self and Soul* (Basic, Books, 1981), 353–373.

[2] J. R. Lucas, «Minds, Machines and Gödel», *Philosophy*, Vol. 36, No. 137 (Apr.–Jul. 1961), 112–127.

иначе. Деннетт свел воедино понятия вычислимости и связал их с прочным пониманием эволюции и обширными знаниями современных исследований в нейробиологии. В его видении мозг и разум оказываются неподходящей средой для машины Тьюринга: мозг – это часть нервной системы, которая сама – часть тела; они неразделимы. Небольшое возмущение течения мысли заставляет сердце биться чуть быстрее, чтобы дать мозгу больше кислорода. Разные препараты влияют на мозг поразному. Мозг получает сигналы от глаз, ушей и других органов и постоянно взаимодействует через тело с окружающим миром.

Мозг – не линейная обрабатывающая система. Это сугубо децентрализованная параллельная система даже без места, где все «сходятся» в одном центральном «картезианском театре» (как Деннетт насмешливо называет воззрения Декарта). Вместо этого Деннетт предлагает «многоэскизную» («multiple drafts») модель разума, в которой части и детали сенсорного ввода, визуальные данные и слова остаются незаконченными, отрывочными и неполными. Если мозг – компьютер, то он не похож на компьютер, который спроектировал бы любой разумный инженер! В нем действительно царит беспорядок.

Более того, то, что мы считаем сознанием, – более упорядоченная деятельность, выводящая на вершину этой распараллеленной структуры. Деннетт утверждает:

> предположение о том, что человеческое сознание (1) слишком молодо, чтобы быть встроенным во врожденный аппарат, (2) в большой степени является продуктом культурной эволюции, который передается в мозг на ранних стадиях обучения, и (3) его удачное устройство определяется бесчисленными микроустановками в глубинах мозга, означает, что его важные функциональные особенности, вполне вероятно, должны быть невидимы для нейроанатомического исследования, несмотря на чрезвычайную рельефность их проявлений[1].

Сознание, хотя бы в каком-то смысле, «разговаривает» с собой, а это требует культуры языка.

Конечно, при создании компьютера, моделирующего человеческий разум, есть один момент: он должен иметь непомерный объем причудливого ввода и не будет работать даже сносно без многолетнего обучения и накопления опыта. И все-таки возможен ли теоретически компьютер, который смог бы пройти неограниченный Тест Тьюринга (который, как считает Деннетт, должен быть очень трудным и справедливым) и обладал бы сознанием? Деннетт считает, что ответ на оба вопроса – да.

---

[1] Daniel Dennett, *Consciousness Explained* (Back Bay Books, 1991), 219.

Независимо от того, какое из механистических представлений мозга вы предпочтете, другое пугающее последствие заключается в том, что решения принимаются не нами, а машиной. Что же тогда происходит с нашей свободой?

Исчезновение свободы в механистическом мире, в сущности, уже давно было предопределено открытием того факта, что движением любой частицы управляют строгие детерминистские законы. В своей ранней книге 1814 года *Essai Philosophique sur les Probabilités* (*Философский очерк вероятности*) маркиз Пьер Симон де Лаплас (1749–1827) писал:

> Дарованный на мгновение разум, который смог бы постичь все силы, движущие природой, и взаимоотношения существ, ее образующих, – разум, совершенно безграничный, чтобы подвергнуть эти данные анализу, – охватил бы единой формулой движения гигантских тел во вселенной и мельчайшего атома; для него не было бы ничего неопределенного, и будущее, как и прошлое, предстало бы пред его взором[1].

Этот взгляд иногда известен как Демон Лапласа. Трудно не прийти к выводу о том, что движение каждого атома во вселенной – включая те атомы, из которых состоят клетки нашего мозга, – стало фиксированным во время Большого взрыва, и с тех пор эти атомы движутся вокруг предопределенным образом.

Конечно, Демон Лапласа не может существовать в действительности. Чтобы отслеживать всю информацию о каждой частице во вселенной, потребовался бы компьютер, больший, чем сама вселенная. Принцип неопределенности Гейзенберга утверждает, что мы не можем знать и положение, и импульс элементарных частиц. Математическим предшественником такого результата взаимодействия атомов была классическая «задача многих тел», но даже задачи трех тел достаточно для большой алгоритмической головной боли.

Если вселенная – действительно машина Тьюринга, и даже если бы мы могли знать ее текущую «полную конфигурацию» и имели бы таблицу всех ее конфигураций, которые управляют этой машиной, мы не смогли бы предсказать, куда эта вселенная движется, без фактического «выполнения» этой «программы».

Неразрешимость – это, в сущности, свобода выбора. Сет Ллойд отмечает, что проблема останова

---

[1] Pierre Simon, Marquis de Laplace, *A Philosophical Essay on Probabilities*, translated by Frederick Wilson Truscott and Frederick Lincoln Emory (John Wiley & Sons, 1902; Dover, 1995).

> ставит в тупик не только традиционные компьютеры, но и любую систему, способную выполнять цифровую логику. Поскольку сталкивающимся атомам внутренне присуща цифровая логика, их отдаленное будущее невычислимо... Непостижимая природа нашей свободы выбора – точный аналог проблемы останова: определив направление движения своих мыслей, мы не знаем, приведут ли они к чему-то вообще. И даже если они ведут к чему-то, мы не знаем, где это «что-то», пока не достигнем его[1].

Дэвид Дейч рассуждает на тему, чем может быть мозг – «классическим» неквантовым или квантовым компьютером:

> Нередко считается, что мозг может быть квантовым компьютером и что его интуиция, сознание и способность решать проблемы могут основываться на квантовых вычислениях. Это *могло бы быть* так, но я не знаю ни доказательств, ни убедительных аргументов в пользу этого. Мое мнение: если мозг – это компьютер, то классический[2].

Затем он признает, что «объяснение вычисления Тьюрингом, кажется, не оставляет места, даже в принципе, для любого будущего объяснения *в физических терминах* таких неотъемлемых свойств разума, как сознание и свобода выбора». Имейте в виду, однако, что с точки зрения многих миров квантовой физики миры постоянно делятся так, что в одном мире вы можете сделать выбор в пользу одного, тогда как в другом мире вы выберете совсем другое. Так что же это, как не свобода выбора? Дейч заключает: «Понятие вычисления у Тьюринга кажется менее оторванным от человеческих ценностей и не препятствует пониманию таких человеческих признаков, как свобода выбора, с учетом того, что она понимается в контексте многих миров»[3].

Когда Стивен Вольфрам начал изучать сложные конструкции, возникающие в клеточных автоматах, он попытался найти способы предсказания результата и, по возможности, короткие пути к генерациям. Он не смог этого, поскольку «нет иного способа предсказать поведение системы, кроме как пройти вместе с ней через все многочисленные этапы ее эволюции... Для многих систем невозможно систематически предсказать их поведение, таким образом, не существует общего способа ускорения процесса их эволюции...». Невозможность предсказания на самом деле предоставляет системе свободу выбора, и Вольфрам даже приводит диаграмму «клеточного автомата, чье поведение должно изображать подобие свободного выбора»[4].

---

[1] Lloyd, Programming the Universe, 98, 36.
[2] Deutsch, *The Fabric of Reality*, 238.
[3] Там же, 336, 339.
[4] Wolfram, *A New Kind of Science*, 739, 741, 750.

Это – утешение. Даже если вселенная и мозг человека имеют в основе своей простые правила и сложные структуры клеточных автоматов или машину Тьюринга, мы не можем предсказать будущее, опираясь только на эти правила. Будущего нет, пока программа не выполнила код.

Или, как в конце трилогии «Назад в будущее»[1] доктор Эмметт Браун говорит Марти Макфлаю и Дженниферу Паркеру: «Это означает, что Ваше будущее еще не написано. Никем. Ваше будущее – это все, что вы делаете. Так делайте его добрым, вместе».

[1] Кинороль Боба Гейла (Bob Gale) по сценарию Гейла и Роберта Земекиса (Robert Zemeckis).

Глава 18

Долгий сон Диофанта

Даже после того как Алан Тьюринг и Алонзо Чёрч доказали, что не существует общей разрешающей процедуры для логики первого порядка, самая первая проблема все еще ждала своего решения. Ею была знаменитая Десятая проблема Давида Гильберта, приведенная в его обращении к Международному конгрессу математиков в 1900 году, как одна из проблем, стоящих перед математиками XX века:

> 10. Определение разрешимости диофантова уравнения
>
> Дано диофантово уравнение с произвольным числом переменных и рациональными числовыми коэффициентами: *разработать процесс, согласно которому за конечное число шагов можно определить, разрешимо ли это уравнение в рациональных числах*[1].

Алгебраические задачи, придуманные александрийским математиком III века Диофантом в его *Арифметике*, всегда могут быть представлены многочленами от многих переменных с целыми коэффициентами. Десятая проблема Гильберта ставит вопрос об общем процессе, который показывает, имеет ли частное диофантово уравнение решение в целых числах.

Конечно, после теоремы Гёделя о неполноте и выводов Чёрча и Тьюринга о неразрешимости немногие математики ожидали, что кто-либо выполнит пожелание Гильберта и действительно «разработает процесс» определения разрешимости диофантовых уравнений. Многие ожидали отрицательного результата – доказательства того, что такого общего процесса не может быть.

Десятой проблемой интересовались многие математики, но, наверное, только один посвятил почти всю свою профессиональную карьеру охотой за этим неуловимым отрицательным доказательством. Им была Джулия Робинсон.

Одно из самых ранних воспоминаний Джулии Робинсон – выкладывание гальки вокруг огромного кактуса сагуаро (цереус гигантский) около своего дома в Аризоне.

---

[1] Ben H. Yandell, The Honors Class: Hilbert's Problems and Their Solvers (A. K. Peters, 2002), 406.

> Думаю, у меня всегда была глубокая симпатия к натуральным числам. Они для меня – единственная реальная вещь. Мы можем представить себе химию, которая отличается от нашей, или биологию, но мы не можем представить себе другую математику чисел. То, что доказано о числах, будет истинно всюду[1].

Джулия Робинсон (Боумен) родилась в Сент-Луисе в 1919 году. Ее сестра Констанс была на два года старше. Когда Джулии было около двух лет, их мать умерла, и их отправили жить к бабушке недалеко от Финикса. Позже к ним присоединился отец со своей новой женой, и они переехали в Пойнт-Лома у залива Сан-Диего.

Когда Джулии было 9 лет, она слегла со скарлатиной, а затем с ревматизмом и в итоге пропустила больше двух лет школы. Чтобы помочь ей нагнать отставание, ее родители наняли репетиторшу, с которой Джулия всего за год прошла классы с пятого по восьмой.

> Однажды она сказала, что мне никогда не удастся извлечь квадратный корень из 2 до того места, где десятичные цифры числа начинают повторяться. Она знала, что этот факт уже доказан, но не знала как. Я тоже не видела, чтобы кто-то доказывал такую вещь, и, придя домой, применила мои недавно приобретенные навыки по извлечению квадратных корней для проверки этого факта, но, в конце концов, к вечеру я сдалась[2].

Джулия поступила в Среднюю школу Сан-Диего в 1933 году, в тот самый год, когда математики стали покидать Гёттингенский и другие университеты Германии. Ее детские болезни сделали ее застенчивой, тихой и неуклюжей, но при этом способной работать в одиночку с большой настойчивостью и терпением.

Поднимаясь вверх по ступенькам знаний, она стала в итоге единственной девушкой в классе, которая не только овладела курсами математики и физики, но и заявила о себе лучшими оценками по этим предметам. В средней школе подарком для нее был скользящий график, и осенью 1936 года, еще не достигнув 17 лет, она начала посещать Государственный колледж Сан-Диего (теперь Государственный университет Сан-Диего) по математической специальности. Обучение стоило 12 долларов за семестр, а Джулия надеялась стать учительницей. «В то время я понятия не имела, что кроме учителей математики бывают еще и математики»[3].

---

[1] Constance Reid, *Julia: A Life in Mathematics* (Mathematical Association of America, 1996), 3. Эта цитата из части книги, названной «Автобиография Джулии Робинсон», которая была фактически написана Констанс Рейд на основе бесед с сестрой. Автобиография была ранее издана в книге Constance Reid, *More Mathematical People* (Academic Press, 1990), 262–280.

[2] Constance Reid, Julia: A Life in Mathematics, 9.

[3] Там же, 21.

В 1937 году в издательстве Simon&Schuster вышла знаменитая теперь книга *Men of Mathematics* (*Математик*) математика Эдварда Темпла Белла (1883–1960). Читая ее, Джулия получила первые представления о том, кто такие математики и что они делают. Несмотря на довольно пристрастное толкование истории и ее персонажей, *Men of Mathematics* действительно вдохновила Джулию, как и десятилетия спустя многих подающих надежды математиков.

В 1939 году Джулия ушла из Государственного колледжа Сан-Диего в Калифорнийский университет в Беркли, где в то время создавался внушительный факультет математики. В первый год обучения ее преподавателем по теории чисел был Рафаэль М. Робинсон, который был примерно на восемь лет старше Джулии. «Во втором семестре у нас было только четыре студента – и снова я была единственной девушкой, – и Рафаэль стал просить меня прогуливаться с ним... На одной из первых прогулок он познакомил меня с результатами Гёделя»[1].

Наверное, они обсуждали и другие теоремы тоже, так как в 1941 году Рафаэль Робинсон и Джулия Боумен поженились. По закону о родстве Джулия не могла преподавать на отделении математики Беркли, хотя она уже работала ассистентом на отделении статистики. (Когда ее принимали на эту работу, в отделе кадров спросили, что она делает каждый день. Она ответила: «в понедельник пыталась доказать теорему, во вторник пыталась доказать теорему, в среду пыталась доказать теорему, в четверг пыталась доказать теорему, в пятницу теорема оказалась ложной»[2].)

Рафаэль и Джулия Робинсоны хотели иметь детей, но детские болезни Джулии серьезно ослабили ее сердце. У нее был один выкидыш, и врач настоятельно отговаривал ее от продолжения попыток иметь детей[3].

В 1946–1947 учебном году Рафаэль был приходящим профессором в Принстонском университете. Они с Джулией посещали курсы Алонзо Чёрча и слушали лекцию Курта Гёделя по основаниям математики во время двухсотлетнего юбилея Принстона.

Вернувшись в Беркли, Джулия Робинсон училась у знаменитого польского логика Альфреда Тарского (1902–1983) и в 1948 году получила степень доктора философии. Еще тогда ее диссертация по-

---

[1] Constance Reid, Julia: A Life in Mathematics, 31, 35.
[2] Там же, 33.
[3] Там же, 43.

казала, «что ее основная область интересов лежит на границе между логикой и теорией чисел»[1].

В 1948 году Тарский получил ее работу по вопросу, косвенно связанному с проблемой, которая будет господствовать в ее жизни как профессионального математика, – Десятой проблемой Гильберта.

Как большинство математиков, Джулия Робинсон не питала иллюзий по поводу разрешимости Десятой проблемы, что осчастливило бы Гильберта. Как она объяснила в своей первой статье о диофантовых уравнениях[2]:

> Поскольку существует множество классических диофантовых уравнений с одним параметром, для которых, как известно, нет эффективного метода определения разрешимости при произвольном значении параметра, то маловероятно, что такая разрешающая процедура может быть найдена. Например, как известно, нет способа определения значения, для которого система диофантовых уравнений
>
> $x^2 + ay^2 = s^2, \quad x^2 - ay^2 = t^2$
>
> разрешима. (Эта задача впервые изучалась арабами в Средневековье.)

Некоторые математики, пытавшиеся разобраться с диофантовыми уравнениями, шли к проблеме не прямым, а окольным путем. Они определяли *диофантово множество* как множество всех решений частного диофантова уравнения. Например, множество всех четных чисел – это на самом деле множество целочисленных значений x, удовлетворяющих следующему диофантову уравнению:

$$x - 2y = 0.$$

В этом уравнении – две переменные, но диофантово множество образуется только из значений x; если x и y – целочисленные решения, то x всегда четно.

Можно также определить *диофантово отношение* между многими переменными. Например, допустим, что нам нужно выразить отношение «x меньше y». Значения x и y, удовлетворяющие этому условию, – это решения следующего диофантова уравнения:

$$x - y + z + 1 = 0.$$

---

[1] Constance Reid with Raphael M. Robinson, «Julia Bowman Robinson (1919–1985)»; в кн. *A Century of Mathematics in America*, Part III (American Mathematical Society, 1989), 410.

[2] Julia Robinson, «Existential Definability in Arithmetic», *Transactions of the American Mathematical Society*, Vol. 72 (1952), 437–449. См. также Julia Robinson, *The Collected Works of Julia Robinson*, ed. Solomon Feferman (American Mathematical Society, 1996), 47–59.

В своей статье Джулия Робинсон не проверяла диофантовость конкретных множеств и отношений, а утверждала, что определенные множества и отношения могут быть выражены в экспоненциальной форме, то есть в виде x^y, где и x, и y – переменные.

Введение степени при обсуждении диофантовых уравнений поначалу кажется нелепым. В диофантовых уравнениях степень недопустима. В диофантовом уравнении могут быть переменные, у которых показатель степени – целое число, но никак не переменная.

Тем не менее в статье показано, что степень – важное отношение, через которое могут быть определены биномиальные коэффициенты, факториал и множество простых чисел. Возможно ли, чтобы степень была действительно диофантовой, потому что может быть определена как диофантово отношение? Это было неясно, но статья Джулии Робинсон показала также, что степень может быть определена через любое диофантово отношение, которое показывает примерно экспоненциальный рост. Ни одного такого диофантова отношения не было известно, но ей казалось «очень вероятным»[1], что они есть.

Строго говоря, последняя (или Великая) теорема Ферма *не* диофантово уравнение:

$$x^n + y^n = z^n.$$

Теорема Ферма утверждает, что уравнение неразрешимо в целых числах для любого n, большего 2, что означает, что n – переменная. Если же заменить n произвольным целым числом, большим 2, уравнение становится диофантовым. Если бы степень оказалась диофантовым отношением, то уравнение Ферма фактически могло бы стать обычным диофантовым уравнением – хотя оно *много* сложнее, чем его стандартная форма.

В 1950 году статья Джулии Робинсон была представлена Международному конгрессу математиков, проходившему тогда в Гарварде. Именно там Джулия Робинсон впервые встретилась с Мартином Дэвисом.

Мартин Дэвис «заболел» Десятой проблемой, будучи еще студентом Городского колледжа Нью-Йорка. Среди профессорско-преподавательского состава отделения математики колледжа был Эмиль Пост, который писал, что Десятая проблема просто «напрашивается на доказательство неразрешимости»[2].

---

[1] Robinson, «Existential Definability in Arithmetic», 438.

[2] Martin Davis, Предисловие к Yuri V. Matiyasevich, *Hilbert's Tenth Problem* (MIT Press, 1993), xiii.

Как оказалось, будучи аспирантом Принстона, Дэвис «не мог заставить себя не думать над Десятой проблемой Гильберта. Вряд ли я думал, что возьмусь когда-нибудь за эту трудную проблему и буду безуспешно пытаться заставить себя избавиться от нее», хотя его докторская диссертация касалась именно этой темы.

Получив степень доктора философии, Мартин Дэвис сразу же отправился на Международный конгресс. Джулия Робинсон вспоминает его искреннее недоумение по поводу ее статьи: «помню, он сказал, что не понимает, как моя работа может помочь решению проблемы Гильберта, если она представляет собой лишь ряд примеров. Я сказала, ну что ж, я сделала, что могла»[1]. Позже Дэвис признавался: «Высказав сомнение в перспективности ее подхода, я, конечно же, сделал одно из самых глупых заявлений в своей жизни»[2].

Джулия и ее старшая сестра Констанс росли порознь, но в 1950 году Констанс вышла замуж за Нейла Дэна Рейда, студента юридического факультета университета Сан-Франциско. Теперь, живя рядом, две сестры быстро сдружились. При поддержке Робинсонов Констанс Рейд написала свою первую книгу *От нуля до бесконечности* (Crowell, 1955). После паломничества сестер в Гёттинген Констанс Рейд написала биографию Давида Гильберта (1970), которую я часто использовал в главе 3 этой книги. Позже Констанс Рейд написала биографии Рихарда Куранта (1976), Иржи Неймана (1982), Э. Т. Белла (1993) и, как дань уважения сестре, *Джулия: Жизнь в Математике* (1996).

Летом 1958, 1959 и 1960 годов у Мартина Дэвиса была возможность работать с Хилари Патнэмом (р. 1926), скорее философом, чем математиком, но имеющим опыт в обеих областях. Летом 1959 года они начали применять в своей работе методы Джулии Робинсон. В итоге Дэвис и Путнэм отправили Джулии Робинсон набросок своей статьи. Та доработала некоторые ее части, и все трое стали авторами новой статьи «Проблема разрешимости экспоненциальных диофантовых уравнений», опубликованной в 1961 году[3].

Как следует из заголовка, статья была посвящена *экспоненциальным* диофантовым уравнениям – разновидности диофантовых

---

[1] Reid, Julia: A Life in Mathematics, 61.

[2] Davis, Предисловие к *Hilbert's Tenth Problem*, xiv.

[3] Martin Davis, Hilary Putnam, and Julia Robinson, «The Decision Problem for Exponential Diophantine Equations», *Annals of Mathematics*, Vol. 74, No. 3 (November 1961), 425–436. Также в *The Collected Works* of Julia Robinson, 77–88.

уравнений, где допускаются степени нескольких видов: переменная в степени другой переменной, постоянная в степени переменной или переменная в постоянной степени (как в обычных диофантовых уравнениях). Во второй половине статьи Дэвиса–Путнэма–Робинсон доказывается, что «не существует общего алгоритма, определяющего, есть или нет у экспоненциального диофантова уравнения решение в положительных целых числах».

До отрицательного решения Десятой проблемы Гильберта теперь не хватало только одной решающей детали, которую Дэвис назвал «гипотезой Джулии Робинсон»[1]. Это диофантово отношение почти экспоненциального роста, наличие которого тогда подразумевало бы, что само возведение в степень – это диофантово отношение, что означало бы, что экспоненциальные диофантовы уравнения могут быть выражены через обычные диофантовы уравнения.

В течение 1960-х годов Джулия Робинсон преподавала математику (и один семестр – философию) в Беркли, изредка продолжая работать и публиковаться по Десятой проблеме Гильберта. Для книги 1969 года *Исследования по теории чисел* она написала 40-страничную главу, в которой подвела итог исследованиям на тот момент, оставив открытым один главный вопрос:

> Является ли отношение $r = s^t$ диофантовым? Если б это было так, то каждое экспоненциальное диофантово уравнение можно было бы заменить равносильным диофантовым уравнением с бóльшим количеством переменных. Кроме того, каждое рекурсивно перечислимое отношение стало бы диофантовым, и, следовательно, проблема Гильберта стала бы неразрешимой. В настоящее время у нас нет ответа на этот вопрос[2].

В течение 1960-х годов Мартин Дэвис преподавал в Политехническом институте Ренселера и в университете Нью-Йорка и часто имел возможность читать лекции о Десятой проблеме Гильберта. Если кто-то просил его дать прогноз о ее разрешимости или неразрешимости, у него был готовый ответ: как пророк из еврейской Библии, Дэвис произносил «я думаю, что гипотеза Джулии Робинсон истинна, и это будет доказано молодым умным русским»[3].

Юрий Владимирович Матиясевич родился в 1947 году в Ленинграде и учился в средней школе с математическим уклоном. В 17 лет по-

---

[1] Davis, Foreword to *Hilbert's Tenth Problem*, xiii.

[2] Julia Robinson, «Diophantine Decision Problems», в кн. LeVeque W. J., ed., *Studies in Number Theory* (Mathematical Association of America, 1969), 107. Также в *Collected Works of Julia Robinson*, 176.

[3] Davis, Предисловие к *Hilbert's Tenth Problem*, xiii.

ступил в Ленинградский государственный университет, и в 1966 году выступил на проходившем тогда в Москве Международном математическом конгрессе. В 1970 году получил степень доктора наук в Ленинградском отделении Математического института имени Стеклова (ЛОМИ).

Матиясевич впервые узнал о Десятой проблеме Гильберта в 1965 году, когда был второкурсником Ленинградского государственного университета. Его научный руководитель Сергей Маслов (1939–1981) предложил ему «попробовать доказать алгоритмическую неразрешимость диофантовых уравнений. Эта задача известна как Десятая проблема Гильберта, но это для вас не должно иметь значения». Он также предложил план действий: «В настоящее время неразрешимость обычно доказывается сведением задачи, о которой уже известно, что она неразрешима, к задаче, неразрешимость которой нужно установить». Маслов не смог только порекомендовать хоть какую-то литературу по этой теме, кроме того, что «есть немного статей американских математиков по десятой проблеме Гильберта, но вам нет нужды изучать их... Американцы пока не добились успеха, так что их подход, вполне вероятно, неприемлем»[1].

После нескольких лет исследования некоторых ничего не обещающих подходов к Десятой проблеме навязчивая идея Матиясевича в кругах Ленинградского университета стала притчей во языцех. Один профессор подтрунивал над ним: «Вы уже доказали неразрешимость десятой проблемы Гильберта? Еще нет? Но в таком случае Вы не сможете окончить университет!»[2] В конце концов, Матиясевич решил прочесть статьи американцев, включая ключевую статью Дэвиса–Патнэма–Робинсон 1961 года.

Вскоре после новогодних праздников 1970 года Матиясевич нашел диофантово отношение для чисел Фибоначчи, что отвечало гипотезе Джулии Робинсон. Ему было 22 года. В конце января он прочитал свою первую публичную лекцию о неразрешимости Десятой проблемы Гильберта, и его слова облетели весь мир.

Джулия Робинсон писала ему: «Если Вам действительно 22 года, то мне особенно приятно осознавать, что когда я впервые пришла к этой догадке, Вы были еще ребенком, и мне оставалось лишь ждать, когда Вы вырастите!»[3]

---

[1] Yuri Matiyasevich, «My Collaboration with Julia Robinson», *The Mathematical Intelligencer*, Vol. 14, No.4 (1992), 38–45. Перепечатано в Reid, *Julia: A Life in Mathematics*, 99–116.

[2] Там же.

[3] Reid, Julia: A Life in Mathematics, 73.

В 1971 году Джулия и Рафаэль Робинсоны приехали в Ленинград, чтобы встретиться с Юрием Матиясевичем и его женой. Последующие несколько лет они, пользуясь почтой, совместно работали над несколькими статьями по диофантовым уравнениям.

Мартин Дэвис написал популярную статью на эту тему для журнала *Scientific American*[1] и более специальную статью[2], которая вошла как Приложение 2 в его классическую книгу *Вычислимость и неразрешимость*, переизданную в 1982 году издательством Dover Publications. В мае 1974 года Американское математическое общество провело симпозиум по чистой математике в университете Северного Иллинойса, посвященный проблемам Гильберта. Дэвис, Матиясевич и Робинсон сделали доклад «Диофантовы уравнения: положительные аспекты отрицательного решения»[3], в котором они исследовали некоторые ожидаемые результаты доказательства.

Благодаря заметной роли в разрешении Десятой проблемы Гильберта Джулия Робинсон стала теперь знаменитой. В 1976 году она стала, наконец, полным профессором в Беркли и первой женщиной-математиком, избранной в Национальную академию наук. В 1982 году она стала первой женщиной-президентом Американского математического общества, а журнал *Ladies Home Journal* включил ее в список ста самых выдающихся женщин Америки[4].

В 1983 году Джулия Робинсон получила награду Общества МакАртура (известную как «грант гения» (genius grant)) и анонимно пожертвовала часть денег, чтобы напечатать в издательстве Оксфордского университета *Собрание сочинений* Курта Гёделя – книги, которая необходима каждому исследователю в области математической логики и вычислимости.

В 1984 году у Джулии Робинсон была обнаружена лейкемия, и в следующем году в возрасте 65 лет она умерла. Ее муж Рафаэль Робинсон умер в 1995 году в возрасте 83 лет.

---

[1] Martin Davis and Reuben Hersh, «Hilbert's 10th Problem», *Scientific American*, Vol. 229, No. 5 (Nov.1973), 84–91.

[2] Martin Davis, «Hilbert's Tenth Problem is Unsolvable», *The American Mathematical Monthly*, Vol. 80, No. 3 (Mar. 1973), 233–269.

[3] Martin Davis, Yuri Matijasevic, and Julia Robinson, «Hilbert's Tenth Problem. Diophantine Equations: Positive Aspects of a Negative Solution», в кн. Felix E. Browder, ed., *Mathematical Developments Arising from Hilbert Problems* (American Mathematical Society, 1976), Vol. 2, 323–378. Также в *Collected Works of Julia Robinson*, 269–324.

[4] Reid, *Julia: A Life in Mathematics*, 81.

Матиясевичу недавно перевалило за 60, а Мартину Дэвису перевалит за 80 в год выхода этой книги (*2008 – прим. перев.*). Оба все еще очень активны в математике.

В 1993 году Юрий Матиясевич написал книгу *Десятая проблема Гильберта*, которая была быстро переведена на английский и вышла в издательстве Массачусетского технологического института. Хотя доказательство неразрешимости Десятой проблемы Гильберта занимает первую сотню страниц книги, Матиясевич переработал его так, чтобы оно было самодостаточным и не требовало почти никаких предварительных знаний.

Машины Тьюринга вводятся в доказательство в главе 5 *Десятой проблемы Гильберта*. Как дитя компьютерного века Матиясевич дает машинам свои имена, похожие на ключевые слова современных языков программирования, а затем показывает их связь с инструкциями, подобными программным. Он доказывает, что «машины Тьюринга не способны решить, действительно ли уравнения, относящиеся к *одному частному семейству диофантовых уравнений*, имеют решения, не говоря уже о *произвольных диофантовых уравнениях*»[1].

Древняя легенда говорит, что Диофант шестую часть своей жизни был мальчиком, спустя двенадцатую часть отрастил бороду, спустя седьмую – женился, спустя пять лет у него появился сын, а затем увидел смерть сына, достигшего половины жизни отца. А потом оставшиеся четыре года жизни прожил, утешая свое горе написанием книги задач по алгебре.

Семнадцать веков спустя Алан Тьюринг умер почти в том же возрасте, что и сын Диофанта. Он создал воображаемый инструмент, который до сих пор позволяет нам исследовать возможности и ограничения человеческого разума и его логические и математические способности.

И Диофант, и Тьюринг оставили после себя тайны. Как и человеческие намерения, некоторые диофантовы уравнения имеют решение, некоторые – нет, а для многих других мы этого так и не узнаем.

---

[1] Matiyasevich, *Hilbert's Tenth Problem*, 93. См. Матиясевич Ю. В. Десятая проблема Гильберта. – М.: Наука, 1993.

Избранная библиография

Веб-ресурсы

Веб-сайт Алана Тьюринга, поддерживаемый Эндрю Ходжесом: www.turing.org.uk

Архив Тьюринга по истории вычислительной техники: www.AlanTuring.net

Цифровой архив Тьюринга: www.turingarchive.org

Книги

Ackermann, Wilhelm. Solvable *Cases of the Decision Problem*. North-Holland Publishing Company, 1962.

Agar, John. *Turing and the Universal Machine: The Making of the Modern Computer*. Icon Books, 2001.

Arbib, Michael A. *Brains, Machines, and Mathematics*. McGraw-Hill, 1964.

Aspray, William. *John von Neumann and the Origins of Modern Computing*. MIT Press, 1990.

Beth, Evert W. *The Foundations of Mathematics: A Study in the Philosophy of Science*, second edition. North-Holland, 1964; Harper & Row, 1966.

Boolos, George S., John P. Burgess, and Richard C. Jeffrey. *Computability and Logic*, fourth edition. Cambridge University Press, 2002.

Börger, Egon, Erich Grädel, and Yuri Gurevich. *The Classical Decision Problem*. Springer, 1997, 2001.

Boyer, Carl B. *A History of Mathematics*, second edition revised by Uta C. Merzbach. John Wiley & Sons, 1989.

Cantor, Georg. *Contributions to the Founding of the Theory of Transfinite Numbers*, перевод и введение Philip E. B. Jourdain. Open Court, 1915; Dover, 1955.

Carpenter, B. E. and R. W. Doran. *A. M. Turing's ACE Report of 1946 and Other Papers*. MIT Press, 1986.

Ceruzzi, Paul E. *A History of Modern Computing*, second edition. MIT Press, 2003.

Ceruzzi, Paul E. *Reckoners: The Prehistory of the Digital Computer, from Relays to the Stored Program Concept*, 1935–1945. Greenwood Press, 1983.362 Selected Bibliography.

Chaitin, Gregory. *Meta Math! The Quest for Omega*. Vintage Books, 2005.

Chaitin, Gregory. *Thinking about G.odel and Turing: Essays on Complexity*, 1970-2007. World Scientific, 2007.

Church, Alonzo. *The Calculi of Lambda-Conversion*. Princeton University Press, 1941.

Church, Alonzo. *Introduction to Mathematical Logic*. Princeton University Press, 1956.

Clark, Andy and Peter Millican, eds. *Connectionism, Concepts, and Folk Psychology: The Legacy of Alan Turing*, Vol. 2. Clarendon Press, 1996.

Conway, Flo and Jim Siegelman. *Dark Hero of the Information Age: In Search of Norbert Wiener, the Father of Cybernetics*. Basic Books, 2005.

Copeland, B. Jack. *Alan Turing's Automatic Computing Engine*. Oxford University Press, 2005.

Copeland, B. Jack. *The Essential Turing*. Oxford University Press, 2004.

Dauben, Joseph Warren. *Georg Cantor: His Mathematics and Philosophy of the Infinite*. Princeton University Press, 1979.

Davis, Martin. *Computability and Unsolvability*. McGraw-Hill, 1958. Enlarged edition, Dover, 1982.

Davis, Martin. *The Universal Computer: The Road from Leibniz to Turing*. W. W.

Norton, 2000. Издано в книге в мягкой обложке под заголовком *Engines of Logic: Mathematicians and the Origin of the Computer*.

Davis, Martin, ed. The Undecidable: *Basic Paper on Undecidable Propositions, Unsolvable Problems and Computable Functions*. Raven Press, 1965.

Dawson, John W. Jr. *Logical Dilemmas: The Life and Work of Kurt Gödel*. A. K. Peters, 1997.

DeLong, Howard. *A Profile of Mathematical Logic*. Addison-Wesley, 1970; Dover, 2004.

Dennett, Daniel. *Consciousness Explained*. Back Bay Books, 1991.

Devlin, Keith. *Mathematics: The New Golden Age*. Penguin, 1988.

Enderton, Herbert B. *A Mathematical Introduction to Logic*, second edition. Harcourt, 2001.

Epstein, Richard L. and Walter A. Carnielli. *Computability: Computable Functions, Logic, and the Foundations of Mathematics*, second edition. Wadsworth, 2000.

Eves, Howard. *Foundations and Fundamental Concepts of Mathematics*, third edition. PWS-Kent, 1990; Dover, 1997.

Ewald, William. *From Kant to Hilbert: A Source Book in the Foundations of Mathematics*, в 2 т. Clarendon Press, 1996.

Franz.en, Torkel. *Gödel's Theorem: An Incomplete Guide to Its Use and Abuse*. A. K. Peters, 2005.

Gödel, Kurt. *Collected Works, Volume I: Publications 1929–1936*, edited by Solomon Feferman et. al. Oxford University Press, 1986.

Gödel, Kurt. *Collected Works, Volume II: Publications 1938–1974*, edited by Solomon Feferman et. al. Oxford University Press, 1990.

Gödel, Kurt. *Collected Works, Volume III: Unpublished Essays and Lectures*, edited by Solomon Feferman et. al. Oxford University Press, 1995.

Goldstein, Rebecca. *Incompleteness: The Proof and Paradox of Kurt Gödel*. W.W. Norton, 2005.

Goldstern, Martin and Haim Judah. T*he Incompleteness Phenomenon: A New Course in Mathematical Logic*. A. K. Peters, 1995.

Grattan-Guiness, I. *The Search for Mathematical Roots, 1870–1940: Logics, Set Theories and the Foundations of Mathematics from Cantor through Russell to Gödel*. Princeton University Press, 2000.

Gray, Jack and Keith Thrower. *How the Turing Bombe Smashed the Enigma Code*. Speedwell, 2001.

Heath, Sir Thomas L. *Diophantus of Alexandria: A Study in the History of Greek Algebra*, second edition. Cambridge University Press, 1910; Dover, 1964.

Heims, Steve J. *John von Neumann and Norbert Wiener: From Mathematics to the Technologies of Life and Death*. MIT Press, 1980.

Herkin, Rolf, ed. *The Universal Turing Machine: A Half-Century Introduction*. Oxford University Press, 1988. Second edition, Springer-Verlag, 1995.

Hilbert, David and Wilhelm Ackermann. *Grundzüge der Theoretischen Logik*. Springer, 1928. Second edition, Springer, 1938; Dover, 1946.

Hilbert, David and Wilhelm Ackermann. *Principles of Mathematical Logic* (перевод второго немецкого издания *Grundzüge der Theoretischen Logik*). Chelsea, 1950.

Hodges, Andrew. *Alan Turing: The Enigma*. Simon & Schuster, 1983.

Hodges, Andrew. *Turing: A Natural Philosopher*. Phoenix, 1997.

Hofstadter, Douglas R. *Gödel, Escher, Bach: an Eternal Golden Braid*. Basic Books, 1979. (Хофштадтер Д. *ГЁДЕЛЬ, ЭШЕР, БАХ: эта бесконечная гирлянда*. – Самара: Издательский дом «Бахрах-М», 2001.)

Hunter, Geoffrey. *Metalogic: An Introduction to the Metatheory of Standard First Order Logic*. University of California Press, 1971.

Kleene, Stephen Cole. *Introduction to Metamathematics*. Van Nostrand, 1952. (Клини С. К. *Введение в метаматематику*. – Либроком, 2008.)

Kleene, Stephen Cole. *Mathematical Logic*. John Wiley & Sons, 1967; Dover, 2002.

Kline, Morris. *Mathematics: The Loss of Certainty*. Oxford University Press, 1980.

Kneale, William and Martha Kneale. *The Development of Logic*. Clarendon Press, 1962.

Leavitt, David. *The Man Who Knew Too Much: Alan Turing and the Invention of the Computer*. W. W. Norton, 2006.

Levin, Janna. *A Madman Dreams of Turing Machines*. Alfred A. Knopf, 2006.

Lewis, C. I. *A Survey of Symbolic Logic*. Univesity of California Press, 1918; Dover, 1960.

Mancosu, Paolo. *From Brouwer to Hilbert: The Debate on the Foundations of Mathematics in the 1920s*. Oxford University Press, 1998.

Matiyasevich, Yuri M. *Hilbert's Tenth Problem*. MIT Press, 1993.

Mendelson, Elliott. *Introduction to Mathematical Logic*, fourth edition. Chapman & Hall, 1997.

Millican, Peter and Andy Clark, eds. *Machines and Thought: The Legacy of Alan Turing*, Vol. 1. Clarendon Press, 1996.

Niven, Ivan. *Numbers: Rational and Irrational*. Mathematical Association of America, 1961.

Ore, Oystein. *Number Theory and Its History*. McGraw-Hill, 1948; Dover, 1988.

Peano, Giuseppe. *Selected Works of Giuseppe Peano*, translated and edited by Hubert C. Kennedy. George Allen & Unwin, 1973.

Penrose, Roger. *The Emperor's New Mind: Concerning Computers, Minds, and the Laws of Physics*. Oxford University Press, 1989.

Petzold, Charles. *Code: The Hidden Language of Computer Hardware and Software*. Microsoft Press, 1999.

Phillips, Esther R., ed. *Studies in the History of Mathematics* (*MAA Studies in Mathematics, Volume 26*). Mathematical Association of America, 1987.

Prager, John. *On Turing*. Wadsworth, 2001.

Quine, Willard Van Orman. *Mathematical Logic*, revised edition. Harvard University Press, 1951, 1981.

Reid, Constance. *Hilbert*. Springer-Verlag, 1970, 1996.

Reid, Constance. Julia: A Life in Mathematics. Mathematical Association of America, 1996.

Robinson, Julia. *The Collected Works of Julia Robinson*, edited by Solomon Feferman. American Mathematical Society, 1996.

Russell, Bertrand. *Introduction to Mathematical Philosophy*, second edition. George Allen & Unwin, 1920; Dover, 1993.

Russell, Bertrand. *The Principles of Mathematics*. Cambridge University Press, 1903. W. W. Norton, 1938.

Shanker, S. G., ed. *Gödel's Theorem in Focus*. Routledge, 1988.

Sipser, Michael. *Introduction to the Theory of Computation*. PWS, 1997.

Teuscher, Christof, ed. *Alan Turing: Life and Legacy of a Great Thinker*. Springer, 2004.

Tiles, Mary. *The Philosophy of Set Theory: An Historical Introduction to Cantor's Paradise*. Basil Blackwell, 1989; Dover, 2004.

Turing, A. M. *Collected Works of A. M. Turing: Mathematical Logic*, edited by R. O. Gandy and C. E. M. Yates. Elsevier, 2001.

Turing, A. M. *Collected Works of A. M. Turing: Mechanical Intelligence*, edited by D. C. Ince. North-Holland, 1992.

Turing, A. M. *Collected Works of A. M. Turing: Morphogenesis*, edited by P. T. Saunders. North-Holland, 1992.

Turing, A. M. *Collected Works of A. M. Turing: Pure Mathematics*, edited by J. L. Britton. North-Holland, 1992.

van Atten, Mark. *On Brouwer*. Wadsworth, 2004.

van Heijenoort, Jean. *From Frege to Gödel: A Source Book in Mathematical Logic*, 1879–1931. Harvard University Press, 1967.

von Neumann, John. *The Computer and the Brain*. Yale University Press, 1958.

Wang, Hao. *Popular Lectures on Mathematical Logic*. Van Nostrand Reinhold, 1981; Dover, 1993.

Wang, Hao. *Reflections on Kurt Gödel*. MIT Press, 1987.

Whitehead, Alfred North and Bertrand Russell. *Principia Mathematics to *56*. Cambridge University Press, 1962.

Whitemore, Hugh. *Breaking the Code*. Fireside Theatre, 1987.

Wiener, Norbert. *Cybernetics, or Control and Communication in the Animal and the Machine*. John Wiley & Sons, 1948. Second edition, MIT Press, 1961. (Винер Н. *Кибернетика, или Управление и связь в животном и машине*. – 2-е изд. – М.: Наука, 1983.)

Wilder, Raymond L. *Introduction to the Foundations of Mathematics*. John Wiley & Sons, 1952.

Wolfram, Stephen. A New Kind of Science. Wolfram Media, 2002.

Yates, David M. *Turing's Legacy: A History of Computing at the National Physical Laboratory 1945–1995*. Science Museum (London), 1997.

Машины Тьюринга, их разновидности и моделирование

Л. Н. Чернышов

В этом дополнении рассматриваются вопросы, которые могут быть интересны тем, кто заинтересовался машинами Тьюринга. Это прежде всего студенты, изучающие информатику. Тема о машинах Тьюринга, которая включается в различные дисциплины, служит теоретической базой основных понятий информатики, в первую очередь понятия алгоритма. Одной из трудностей в освоении темы является различие в определениях машины Тьюринга, которые даются в книгах и учебниках. Важно понять причину этих различий и уметь приводить одни определения в другие. Моделирование машин Тьюринга и их разновидностей – различного рода автоматов – хорошая задача на программирование, которая часто ставится перед студентами. Для начинающих программистов полезно показать простые решения этой задачи, а для продвинутых – инструментальные средства, реализация которых могла бы составить предмет курсовой и даже дипломной работы.

Понятие алгоритма в математике известно давно, но началом систематической разработки теории алгоритмов стал именно 1936 год, когда были опубликованы работы Чёрча, Поста и Тьюринга. В дальнейшем теория получила развитие в трудах и других математиков, что привело к появлению разных формализмов для описания алгоритмов. Вот неполный список наиболее важных формализмов:

- Машины Тьюринга [Тьюринг, 1936–1937];
- Лямбда-исчисление [Чёрч, 1941];
- Системы Поста [Пост, 1943];
- Алгоритмы Маркова [Марков, 1951];
- Грамматики Хомского [Хомский, 1959];
- ТАГ-системы [Пост, 1965];
- Большинство языков программирования [Саммет, 1969].

В каждом из этих формализмов дается свой способ описания алгоритма. Но алгоритм, записанный одним из этих способов, можно

промоделировать алгоритмом, записанным любым другим способом. Доступное введение в эту область можно найти в книге Мальцева [5].

Очень близкими являются абстракции, задаваемыми машинами Тьюринга и машинами Поста (в книге Мальцева дается определение машины Тьюринга–Поста). Статья Поста «Финитные комбинаторные процессы, формулировка 1» была опубликована в «Журнале символической логики» в том же 1936 году, что и статья Тьюринга (перевод статьи Поста см. в [8]). В машинах Поста ячейки ленты либо пустые, либо помечены, то есть алфавит состоит из двух символов, но зато присутствует конечное состояние. Первоначальное тьюринговское определение в ходе развития теории алгоритмов неоднократно видоизменялось. В его современном виде машина Тьюринга (МТ) также снабжается конечными состояниями.

Различия в определениях МТ можно объяснить разными целями авторов в их использовании (удобство выкладок в доказательствах, связи с другими теориями и т. п.). Однако все определения эквивалентны, и легко можно преобразовать МТ из одного представления в другое. В фундаментальной монографии Ахо и Ульмана [7] определения МТ и других конструкций даются с использованием языка теории множеств, функций, бинарных отношений. Оно удобнее для описания свойств, особенностей функционирования...

Система обозначений и формализованное описание помогает (облегчает) и при моделировании (выбор способа хранения, разбиение на подпрограммы и даже именование переменных).

Рассмотрим несколько определений МТ из работ разных авторов.

(1) Определение МТ [3, стр. 61 или 6, стр. 69] из монографии одного из начинателей теоретической информатики в СССР Б. А. Трахтенберга носит неформализованный характер, что вполне соответствует серии «Популярные лекции по математике». Основные понятия: *внешний алфавит* (символы на ленте), *пустой знак, обозреваемая ячейка, внутренний алфавит* (*П, Л, Н,* $q_1 \ldots q_n$), где q_i . . – состояния. Машина Тьюринга описывается подобно реальной ЭВМ, имеющей *логический блок* с *входной парой* (символ внутреннего алфавита + символ внешнего алфавита) *и выходной парой* (новое состояние и новый символ на ленте). Ключевое понятие – *функциональная схема* машины, представляющая собой таблицу, столбцы которой занумерованы знаками состояний, а строки – знаками внешнего алфавита. (Фактически эта таблица задает функцию перехода). В понятие начальной конфигурации входят начальная информация на ленте и начальная ячейка. Машина останавливается после конечного числа тактов, по-

давая сигнал об остановке. Если нет – машина не применима к начальной конфигурации.

(2) Определение машины Тьюринга–Поста [3, стр. 219, 2-е изд.]. В этом определении рассматривается конечная лента, но в процессе работы к существующим ячейкам машина может пристраивать новые ячейки. Каждая ячейка ленты может находиться в одном из конечного множества состояний, обозначаемых символами a_0, $a_1 ... a_m$ (*внешний алфавит*). Состояние внутренней памяти (*внутренние состояния*) обозначается символами q_0, $q_1 ... q_n$, образующими *внутренний алфавит*. Одно из внутренних состояний называется заключительным (q_0 – *стоп-символ*). Состояние машины определяется *машинным словом* $a_{j1}a_{j2} ... q_ia_{jk} ... a_{jr}$. Совокупность команд одного из трех видов: $q_ia_j \rightarrow q_sL$, *(сдвиг влево)* $q_ia_j \rightarrow q_s R$*(сдвиг вправо)*, $q_ia_j \rightarrow q_sa_r$ *(замена символа)*, – образует *программу*.

Такое определение иногда называют *МТ в четверках*, так как за один такт машина либо передвигает головку без изменения символов на ленте, либо без движения изменяет символ. Для описания команды достаточно четырех элементов. В других определениях, где одновременно может быть и движение, и замена, машины называют *МТ в пятерках*.

Последовательные состояния машины записываются цепочками $m \models m^{(1)} \models m^{(2)} ...$, а запись $m \vdash b$ означает, что b получено из m за несколько тактов работы (в современных определениях $\models$ – бинарное отношение на множестве цепочек, а $\vdash$ – его транзитивное замыкание). Команды МТ могут рассматриваться как правила подстановки в машинное слово, что используется автором при доказательстве основных теорем.

Приведенные определения МТ применяют разный набор понятий и различную терминологию, потому что первое рассчитано на неподготовленного читателя, а второе содержится в серьезной математической работе.

Следующие три определения более формализованы, но тоже отличаются. Для удобства сравнения определения приводятся в обозначениях, принятых в [7].

Три элемента у них общие, а именно:

- Q – конечное множество *состояний,*
- Σ – конечное множество *входных символов,*
- q_0 – *начальное состояние*, $q_0 \in Q$.

В остальных элементах имеются отличия:

(3) Определение МТ [7, стр. 49]

МТ описывается шестеркой $M = (Q, \Sigma, \Gamma, \delta, q_0, b)$, где

- Γ – конечное множество *ленточных символов,* подмножество Σ,
- δ – *функция переходов* или программа, $\delta: Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$,
- b – пустой символ, $b \in \Gamma \setminus \Sigma$.

(4) Определение МТ [9, стр. 331]

МТ описывается семеркой $M = (Q, \Sigma, \Gamma, \delta, q_0, b, F)$, где

- Γ – конечное множество *ленточных символов*, подмножество Σ,
- δ – *функция переходов,* $\delta: Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$,
- b – пустой символ, $b \in \Gamma \setminus \Sigma$,
- F – множество *заключительных (допускающих) состояний,* подмножество Q.

(5) Определение МТ [10, стр. 258]

МТ описывается шестеркой $M = (Q, \Sigma, \delta, q_0, q_{yes}, q_{no})$, где

- δ – *функция переходов* или программа, $\delta: Q \times \Gamma \to Q \times \Gamma \times \{L, R, H\}$,
- q_{yes} – *допускающее состояние,* $q_{yes} \in Q$,
- q_{no} – *отвергающее состояние,* $q_{no} \in Q$,

Входная лента ограничена слева.

Разделение алфавита на входные и ленточные символы не является существенным. Ленточные символы вводятся только для удобства. У Тьюринга ленточными символами были маркеры, играющие служебную роль. Результатом работы были только входные символы. Если машина реализует алгоритм преобразования входа в выход, то всегда можно считать, что они состоят из символов одного алфавита.

Действия в результате такта могут включать лишь движение головки только влево или вправо. Замена символа без движения иногда называют пустым тактом. От них всегда можно избавиться. Пусть есть команда $q_i a_j \to q_s a_r$H. После нее может выполниться только команда $q_s a_r \to q_{s1} a_{r1}$X, где X – некоторое действие. Если это сдвиг Л (П), то вместо этих двух команд можно задать команду $q_i a_j \to q_{s1} a_{r1}$X. Если X – опять команда Н (стоять на месте), то рассмотрим команду $q_{s1} a_{r1} \to q_{s2} a_{r2}$X, и если X – сдвиг, то заменяем три команды одной. Если X = Н, то смотрим следующую команду и т. д. До бесконечности (головка стоит на месте) это продолжаться не может. Либо мы попадем в конечное состояние, либо опять вернемся к первой конфигурации $q_i a_j \to q_s a_r$H, что будет означать зацикливание машины.

В определении (5) не указан пустой символ. Опять же его введение – только удобство при записи. Можно считать его входящим в алфавит, а можно считать, что ячейка ленты просто пустая, и для обозначения использовать специальный символ, например *e* (от *empty*).

Следующее отличие – количество и виды заключительных состояний. В определении (3) заключительные состояния вообще не заданы. Считается, что машина останавливается, если она не может выполнить ни одной команды. Это значит, что для пары $q_i a_j$ не определена функция перехода. В соответствующей таблице переходов (функциональной схеме) клетка для (q_i, a_j) пустая. Если добавить еще одно состояние q_k, объявив его конечным, и заполнить пустые клетки значениями (q_k, a_j, R), то мы получим эквивалентную МТ с одним конечным состоянием. Она будет давать тот же результат, что и первоначальная МТ, только останавливаться на одну ячейку правее. Машину, в которой таблица переходов не имеет пустых клеток (функция переходов полностью определена), обычно называют полной. Если в ней не определять конечного состояния, то при любом входе она никогда не останавливается.

Заключительных состояний может быть несколько. Смысл в этом заключается в том, что кроме переработки входа его можно как-то квалифицировать. Например, разделить их на «хорошие» и «нехорошие». Так сделано в определении (5), где есть два заключительных состояния: допускающее (входная цепочка «хорошая») и отвергающее (входная цепочка «нехорошая»). Для преобразования МТ с несколькими заключительными состояниями в МТ с одним заключительным состоянием достаточно все заключительные состояния заменить (переименовать в таблице переходов) в одно.

Более существенным отличием в определениях является односторонняя (ограниченная слева) лента (5). Собственно, так был определена машина Поста. Покажем, что для любой МТ можно построить эквивалентную МТ', которая делает то же самое, но работает с лентой, ограниченной слева. Добавим в алфавит входных символов специальный маркер #, который МТ' в начале работы поставит слева от входной цепочки символов, затем вернется в исходную позицию и начнет выполнять команды МТ. Как только МТ' сдвинется влево на символ # по команде $q_i a_j \to q_s a_r$Л, выполним сдвиг всего содержимого ленты вправо на одну позицию. Инициализирующую часть МТ' построить несложно, особенно если считать, что начальная позиция головки на первом символе входа. Построим вспомогательную МТ'' для сдвига содержимого ленты вправо. Ограничимся пока алфавитом из двух символов: 0 и 1. Например, для входа #101_ должен быть получен выход #_101_ (знак подчеркивания означает пробел). В начальном состоянии q_0 головка установлена на левый #. Этот сдвиг будет осуществляться следующими командами (q_6 – конечное состояние):

$q_0\,\#\to q_1\#\Pi$, $q_1 0\to q_1 0\Pi$, $q_1 1\to q_1 1\Pi$, $q_{1\_}\to q_{2\_}\Lambda$, $q_{3\_}\to q_{2\_}\Lambda$,
$(q_2 0\to q_{4\_}\Pi$, $q_{4\_}\to q_3 0\Lambda)$, $(q_2 1\to q_{5\_}\Pi$, $q_{5\_}\to q_3 1\Lambda)$, $q_2\#\to q_6\#\Pi$.

Если на входе будут другие символы, кроме 0 и 1, то для каждого нужно будет добавить по две команды, как команды для 0 и 1, взятые в скобки, и команду $q_1 X\to q_1 X\Pi$. Можно было бы обойтись фиксированным числом команд, если использовать технику Тьюринга с маркерами.

Если в исходной МТ имеются команды $q_i a_j\to q_s a_r\Lambda$, $q_{s\_}\to qa\mathrm{D}$, то это означает, что мы оказались в состоянии q_s на пустой клетке после движения влево, то есть дошли до крайнего левого символа. Заменим теперь команду $q_{s\_}\to qa\mathrm{D}$ на команду $q_s\#\to q_1\#\Pi$, в результате которой начнут выполняться команды МТ" сдвига ленты. Последнюю команду $q_2\#\to q_6\#\Pi$ заменит на $q_2\#\to q_s\#\Pi$, чтобы МТ' продолжила работу исходной МТ. Поскольку сдвиг за самый левый символ цепочки может произойти из разных состояний, придется для каждого такого q_s сделать свою копию МТ". Здесь возникает та же проблема, как вызов подпрограммы и возврат из нее в обычной программе в машинных кодах. В реальных машинах для этого имеются специальные команды, сохраняющие адрес возврата, но в МТ такой возможности нет.

Последнее различие – в определениях конфигурации и такта. Следуя обозначениям, принятым в (3), прописными греческими буквами будем обозначать цепочки, отдельные символы алфавита Σ – строчными латинскими. В (2) состояние машины определяется машинным словом, а понятие конфигурации не используется. В (3) конфигурация определяется как $(q, \alpha\uparrow\beta)$, где q – текущее состояние, $\uparrow$ – специальный для обозначения позиции головки, которая обозревает самый левый символ цепочки β. В (4) для описания конфигураций используется запись, подобная машинному слову из (1), но применяется термин *мгновенное описание*. Также похоже обозначаются *переходы* (а не такты) с помощью знаков $\vdash$ (один переход) и $\vdash^*$ (несколько переходов).

Более строгое определение конфигурации и такта можно дать по аналогии с определениями для МП-автомата [7, стр. 193]. *Конфигурацией* МТ называется четверка $(\alpha, q, c, \beta) \in \Gamma^* \times Q \times \Gamma \times \Gamma^*$, где q – текущее состояние, α – цепочка слева от позиции головки, c – обозреваемый символ, β – цепочка справа от позиции головки. (Множество Γ^* – множество всех цепочек в алфавите Γ.) Необходимое уточнение: если α непустая, то она не начинается с пустого символа, если β непустая, то она не заканчивается пустым символов. *Такт* работы МТ

представляется в виде бинарного отношения ├ на множество конфигураций:

$(\alpha a, q, c, \beta) \vdash (\alpha, q', a, c'\beta)$, если $\delta(q,c) = (q',c',L)$
$(\alpha, q, c, a\beta) \vdash (\alpha c', q', a, \beta)$, если $\delta(q,c) = (q',c',R)$

Особые случаи (головка находится на крайнем левом или крайнем правом символе):

$(e, q, c, \beta) \vdash (e, q', b, c'\beta)$, если $\delta(q,c) = (q',c',L)$
$(\alpha, q, c, e) \vdash (\alpha c', q', b, e)$, если $\delta(q,c) = (q',c',R)$

Начальная конфигурация (e, q_0, a, β), где e – пустая цепочка (головка находится на крайнем левом символе входной цепочки). Из такого определения видно, что части ленты справа и слева от текущего символа ведут себя как стеки. Этим мы воспользуемся при написании программы моделирования МТ.

Разновидности МТ

Существуют разновидности машин Тьюринга, часть из которых описана в [10]. Эти разновидности связаны со структурой МТ, а именно: добавлением дополнительных лент (многоленточные МТ), добавлением головок (МТ с несколькими головками), записью в одну клетку нескольких символов (МТ с многомерной лентой) и т. п. Такие МТ могут оказаться полезными для решения каких-либо задач, но они могут быть сведены к простой МТ, то есть они не увеличивают вычислительной мощи машины.

Другой класс разновидностей МТ связан с наложением ограничений на функцию перехода и совершаемые действия. В результате получаются абстрактные машины меньшей мощности, но практически полезные. Наиболее известными являются конечные автоматы и МП-автоматы (автоматы с магазинной памятью). Более полное изложение теорий этих автоматов в приложении для задач синтаксического анализа и построения компиляторов дается в [7]. Обычно определение подобных автоматов дается независимо без ссылки на определения МТ.

Существенным расширением МТ являются недетерминированные МТ, отличающиеся тем, что функция перехода неоднозначна, то есть ее область значений – множество всех подмножеств множества $Q \times \Gamma \times \{L, R\}$. Вопрос о преобразовании такой МТ в обычную МТ в общем случае остается открытым. С недерминированными

МТ связана одна из семи задач тысячелетия (см., например, Википедию) – эквивалентности классов *P* и *NP* проблем, где *P* – класс проблем, решаемых за полиноминальное время обычными МТ, а *NP* – недерминированными МТ.

Рассмотрим, как из определения МТ получаются определения конечных автоматов и МП-автоматов, которые играют существенную роль в теории формальных языков и методах трансляции.

Возьмем за основу определение (4), в котором функция перехода определена как $\delta: Q \times \Gamma \to Q \times \Gamma \times \{L,R\}$. Пусть МТ не изменяет символы на ленте и двигается только направо. Тогда мы получим функцию перехода конечного автомата: $\delta: Q \times \Sigma \to Q$. В ленточных символах при этом нет необходимости. Но что является результатом работы, если на ленте ничего не меняется? В данном случае мы хотим определить, прочитает ли машина входную цепочку до конца и в каком состоянии это произойдет. Если да и мы окажемся в конечном состоянии, то цепочка считается допустимой, если нет или состояние не будет конечным, то цепочка недопустима. Говорят, что автомат работает как распознаватель. Заметим, что здесь конечное состояние «работает» по-другому. Автомат не останавливается, попадая в конечное состояние, а может продолжать читать цепочку дальше. Как следует из теории, конечные автоматы распознают важный класс языков – регулярных.

Автомат с магазинной памятью можно рассматривать как МТ с двумя лентами и соответственно с двумя головками. Ленты играют разную роль. На основной ленте размещается входная цепочка, и МТ работает с ней как конечный автомат, то есть читает слева направо. Вторая, рабочая лента, которую называют магазинной памятью (МП), ограничена с одной стороны (с левой). На ней записываются ленточные (магазинные) символы, головка всегда находится на самом правом символе цепочки (вершине МП). На каждом такте автомат может прочитать один символ на входной ленте (или остаться на месте), а на второй ленте сначала стереть символ в вершине, сдвинуться влево и записать подряд символы цепочки, значение которой определяется функцией перехода. Таким образом, функция перехода задается так: $\delta: Q \times (\Sigma \cup \{e\}) \times \Gamma \to Q \times \Gamma^*$.

МП-автоматы, как и обычные автоматы, могут быть как детерминированными, так и недетерминированными. Существуют также различные их модификации. Класс распознаваемых языков – контекстно-свободные языки.

Моделирование машин Тьюринга

Написание программы, моделирующей МТ, не представляет труда. Можно моделировать как конкретную МТ, так и универсальную (УМТ). Разработка таких программ-эмуляторов обычно предлагается студентам младших курсов, изучающим программирование, в качестве упражнения. При этом важна не столько логика программы, которая достаточно проста, сколько структура программы, разбиение на процедуры, интерфейс программы, требования к ограничениям и т. п.

В Интернете можно найти много различных реализаций МТ, например на сайте http://ru.downv.com/software-download/turing-machine-emulator.

Приведем несколько примеров с использованием различных инструментов. Первый пример написан на Компонентном Паскале и реализован в среде Блэкбокс (см. http://www.inr.ac.ru/~info21/software.htm). Те, кто не сталкивался с этим языком, легко разберется в программе, так как язык Паскаль знают многие. Компонентный Паскаль был создан на основе языка Оберон-2, который рассматривался авторами, Н. Виртом и Х. Мёссенбоком, как язык для написания алгоритмов. Язык имеет простой и понятный синтаксис, а среда программирования Блэкбокс – удобные и эффективные средства разработки.

```
MODULE UMT;   (* моделирование универсальной МТ *)
  IMPORT Log := StdLog, In := i21sysIn;
  CONST N=256; L=100;
  VAR MT : ARRAY N,N OF RECORD q,c,d:CHAR END; (* таблица переходов *)
    Лента: ARRAY L OF CHAR;
    состояние,начальное,конечное: CHAR;
    позиция,начало: INTEGER;

  PROCEDURE ЗадатьМТ*;
    VAR c:CHAR; i,j,k:INTEGER;
  BEGIN
    In.OpenSelection; k:=0;
    In.Char(начальное); In.Char(конечное); In.Char(c);
    состояние := начальное;
    In.Char(c);
    WHILE In.done DO
      i:= ORD(c); In.Char(c); j:= ORD(c);
      In.Char(c); MT[i,j].q :=c;
      In.Char(c); MT[i,j].c :=c;
      In.Char(c); MT[i,j].d :=c;
      In.Char(c); INC(k); ASSERT(c=';');
```

```
    In.Char(c);
  END;
  Log.Ln;
  Log.String('начальное '); Log.Char(состояние);
  Log.String('; конечное '); Log.Char(конечное);
  Log.String(' команд '); Log.Int(k);
END ЗадатьМТ;

PROCEDURE Конфигурация*; (* вывод конфигурации в рабочий журнал *)
  VAR i:INTEGER;
BEGIN
  Log.Ln; Log.Char('(');Log.Char(состояние); Log.Char(',');
  i := начало;
  WHILE Лента[i] # 0X DO
    IF i = позиция THEN Log.Char('!'); END;
    Log.Char(Лента[i]); INC(i)
  END;
  Log.Char(')');
END Конфигурация;

PROCEDURE ЗадатьЛенту*;
  VAR c:CHAR; i:INTEGER;
BEGIN
  FOR i:=0 TO L-1 DO Лента[i] := 0X; END; (* чистка ленты     *)
  In.OpenSelection;     (* открыть область исходных данных    *)
  In.Int(i); In.Char(c);
  позиция := i;         (*начальная позиция на первом символе *)
  начало := i;          (*самый левый непустой символ         *)
  WHILE In.done   DO
    In.Char(c); Лента[i] := c; INC(i);
  END;
  Лента[i-1] := 0X;
  Log.Ln;Log.String('Начальная конфигурация');
  состояние := начальное;
  Конфигурация
END ЗадатьЛенту;

PROCEDURE Переход*(q,c:CHAR; OUT q1,c1,d:CHAR);
BEGIN
  q1 := MT[ORD(q),ORD(c)].q;
  c1 := MT[ORD(q),ORD(c)].c;
  d := MT[ORD(q),ORD(c)].d;
END Переход;

PROCEDURE Такт*;
  VAR c,q1,c1,d:CHAR;
BEGIN
  IF состояние = конечное THEN  Log.String('Конец!');RETURN
  END;
  Переход(состояние, Лента[позиция],q1,c1,d);
```

```
    Лента[позиция] := c1; состояние := q1;
    IF d = 'L' THEN   (* сдвиг влево *)
      DEC(позиция);
      IF позиция<начало THEN начало := позиция; ASSERT(начало>=0)  END;
    ELSIF d = 'R' THEN (* сдвиг вправо *)INC(позиция);
END;
    IF Лента[позиция] = 0X THEN  Лента[позиция] := '_';  END;
    Конфигурация
  END Такт;

  PROCEDURE Пуск*;
  BEGIN
    состояние := начальное;
    WHILE состояние # конечное DO Такт END
  END Пуск;

BEGIN
END UMT.

☻UMT.ЗадатьМТ начальное конечное q с q1 c1 d ; …
15 1_1_R;1.1.R;1_1_R;1=2_L;2-5_L;2,3=L;3.3.L;3-4-L;4_4_L;4,1_R;
bt b0b0R;b1b1R;b_i_L;i0r1L;i1i0L;i_r1L;r0r0L;r1r1L;r_t_R;
06 0#1#R;1010R;1111R;1_2_L;3_2_L;204_R;4_30L;215_R;5_31L;2#6#R;

☻UMT.ЗадатьЛенту   начальная позиция входная цепочка
0 .....-..=     50 0 0 #101_

☻UMT.Такт     пошаговое выполнение МТ
☻UMT.Пуск     запуск МТ до конца
```

Программа легко читается (теми, кто знает Паскаль) и практически не нуждается в дополнительных комментариях. Несколько пояснений для тех, кто не знаком с системой программирования Блэкбокс. Данные и команды запуска находятся в одном файле с исходным текстом (от начала модуля «MODULE UMT ...» до строки «☻UMT. Пуск ...»). Знак «☻» является кнопкой запуска указанной после него процедуры. При копировании текста в рабочее окно Блэкбокса (что лучше делать через обычный Notepad) этот знак надо ввести, нажав **Ctrl**+**Q**. Он будет выглядеть как черный кружок с восклицательным знаком внутри.

После компиляции выполнение программы осуществляется следующим образом. Сначала задается МТ. Здесь приводятся примеры трех МТ в строках за «☻UMT.ЗадатьМТ.»: вычитание чисел (пример из [10]), прибавление 1 к двоичному числу (пример из книги) и вставка пробела (пример, приведенный выше). МТ задается пятерками, разделенными точками с запятой. Перед пятерками задаются

начальное и конечное состояния. Для задания МТ необходимо выделить мышкой соответствующую строку и нажать на командную кнопку «☻UMT.ЗадатьМТ.». Как результат в рабочий журнал будет выведено (для второго примера):

```
начальное b; конечное t команд 9
```

Далее необходимо задать начальную конфигурацию. Для каждого примера приведено по одной конфигурации (в строке после «☻UMT.ЗадатьЛенту»). Перед входной цепочкой указывается позиция в массиве «Лента», с которой цепочка будет записана. Предполагается, что головка МТ устанавливается на первый символ цепочки. Пустой символ задается знаком «_». Выделив мышкой символы «50 0», в рабочем журнале должны получить:

```
Начальная конфигурация
(b,!0)
```

В конфигурации указываются текущее состояние и цепочка, в которой знаком «!» отмечается позиция головки. Далее можем выполнить МТ в пошаговом режиме рабочем, нажимая ☻UMT.Такт, или сразу выполнив МТ до конца по команде ☻UMT.Пуск. В рабочем журнале получим последовательные конфигурации (к нулю прибавилась 1):

```
(b,0!_)
(i,!0_)
(r,!_1_)
(t,_!1_)
```

Программу моделирования конкретной МТ можно получить, изменив процедуру ЗадатьМТ. В ней достаточно заполнить таблицу переходов непосредственными значениями.

Второй пример реализации программы-эмулятора МТ написан на функциональном языке Haskell [11]. Программу можно опробовать, используя простую среду программирования (http://www.haskell.org/haskellwiki/WinHugs). Обратите внимание на размер кода программ, их простоту, прозрачность и отсутствие ограничений, которые были в предыдущем примере. Заметим также, что зачатки функционального похода встречаются в статье Тьюринга.

Программа на Haskell'е, моделирущая ту же МТ прибавления единицы, может состоять из одной функции m1, имеющей три аргумента: цепочка слева от головки, состояние и цепочка справа от головки. Просматриваемый символ – первый символ цепочки справа. Запуск этой программы с входной цепочкой «1011» – вызов функции m1 [] 'b' "1011". Вот программа:

```
m1 xp     'b' ('0':xn) = m1 ('0':xp)  'b' xn
m1 xp     'b' ('1':xn) = m1 ('1':xp)  'b' xn
m1 (c:xp) 'b' []       = m1 xp        'i' (c:[])
m1 (c:xp) 'i' ('0':xn) = m1 xp        'r' (c:'1':xn)
m1 (c:xp) 'i' ('1':xn) = m1 xp        'i' (c:'0':xn)
m1 []     'i' ('1':xn) = m1 []        'r' ('1':'0':xn)
m1 []     'i' ('0':xn) = ('1':xn)
m1 (c:xp) 'r' ('0':xn) = m1 xp        'r' (c:'0':xn)
m1 (c:xp) 'r' ('1':xn) = m1 xp        'r' (c:'1':xn)
m1 []     'r' xn       =    xn
```

Программа строится достаточно просто. Вид строки зависит от вида действия МТ. Если команда МТ задается пятеркой (q, c, q', c', R), то надо записать строку (состояния и символы задаются символьными константами – литерой в апострофах)

```
m1 xp q (c:xn) = m1 (c':xp) q' xn
```

К левой строке дописывается символ `c'` (двоеточие – присоединение элемента к списку), а правая сокращается на один символ (строка разделяется по шаблону (c:xn)). Для пятерки (q, c, q', c', L) строка должна быть такой:

```
m1 (x:xp) q (c:xn) = m1 xp q' (x: c': xn)
```

Теперь в правой строке `c` заменяется на `c'` и дописывается символ, которой был слева.

Особые случаи, когда слева или справа оказываются пустые строки (обозначается «[]»), то есть просматривается пустая клетка, надо учитывать отдельно (b – символ пробела):

```
m1 xp     q [] = m1 (c':xp) q' (b:[]) -- вправо
m1 (x:xp) q [] = m1 xp q' (x: c': []) -- влево
m1 []   (c:xn) = m1 [] q' (b: c': xn) -- влево
```

Если состояние конечное, результатом функции перехода является строка

```
m1 xp q xn = (reverse xp) ++ xn
```

Здесь операция «++» соединяет строки, а функция `reverse` обращает строку слева от головки, что необходимо, так как в целях удобства в конфигурации часть справа обрабатывается в обратном порядке. В приведенном примере завершающая строка проще, поскольку машина заканчивает работу, когда слева – пустая строка.

Эти правила построения настолько просты, что легко можно написать программу, которая генерировала бы Haskell-программы по списку «пятерок» МТ (например, такому, как это было задано в программе на Компонентном Паскале).

В заключение приведем пример реализации УМТ на Haskell'e, в которой интерпретируемые МТ за один раз могут выполнять несколько действий, как в статье Тьюринга. Функция uMT, реализующая УМТ, будет иметь 2 параметра: собственно МТ (ее идентификатор или описание) и входную цепочку. МТ будем задавать кортежем с первым элементом (тоже кортежем), задающим начальное и конечное состояния, и вторым – списком команд МТ. Команда МТ задается парой: строка с состоянием, текущим символом и новым состоянием и строкой, задающей действия (сдвиги L, R и действия печать символа Px). Например, вызов `uMT (('S','T'),[("S0T","RP1RP0RP1")]) "0"` должен выдать `"0101"`.

Вот реализации функции uMT с вспомогательными функциями uMT', tr1, com1:

```
uMT mt (a:xs) = uMT' mt q0 (" ", a, xs++" ") -- обрамление пробелами
     where q0 = fst (fst mt)                 -- начальное состояние
uMT' mt q (ap,a,an) = if q==qt then (reverse ap)++[a]++an
                      else uMT' mt q' (ap',z,an')
   where (ap',z,an') = tr1 ww (ap,a,an)      -- преобразование ленты
         (q',ww) = com1 q a mt               -- действия для q и a
             qt = snd (fst mt)               -- конечное состояние
tr1 [] xx                = xx
tr1 ('P':z:t) (xp,c,xn)   = tr1 t (xp,z,xn)  -- печать символа
tr1 ('L':t) ((p:xp),c,xn) = tr1 t ((s xp),p,(c:xn))  -- сдвиг влево
tr1 ('R':t) (xp,c,(n:xn)) = tr1 t ((c:xp),n,(s xn))  -- сдвиг вправо
com1 q a mt = ((last p1),p2)
     where (p1,p2) = head (filter (fc1 q a) (snd mt))
           fc1 q a ((q1:a1:t),ww) = q==q1 && a==a1
```

МТ прибавления 1 задается следующим образом:

```
minc = (('b','t'),[("b0b","P0R"),("b1b","P1R"),("b i","P L"),
        ("i0r","P1L"),  ("i1i","P0L"),("i r","P1L"),("r0r","P0L"),
        ("r1r","P1L"),("r t","P R")])
```

При вызове `uMT minc "1011"` в результате получится `" 1100 "`.

Разбор программы не проводится, так как это потребует от читателя более углубленных знаний по языку программирования Haskell.

Из готовых инструментов следует выделить программу JFLAP, разработанную в Дюкском университете, США (http://www.jflap.org и [10]). Программа написана на языке Java. Программа замечательна тем, что, кроме МТ, она может эмулировать еще 10 моделей, в том числе конечные автоматы, МП-автоматы и др. Модели задаются в интерактивном графическом редакторе. МТ изображается в виде

диаграммы – ориентированного графа, в котором вершины помечены символами состояний, а на дугах указаны символ входа, символ замены и действие сдвига. Вершины, соответствующие начальному и конечному состояниям, особым образом выделяются (см. рис. 1). Вот так выглядит пример МТ прибавления единицы:

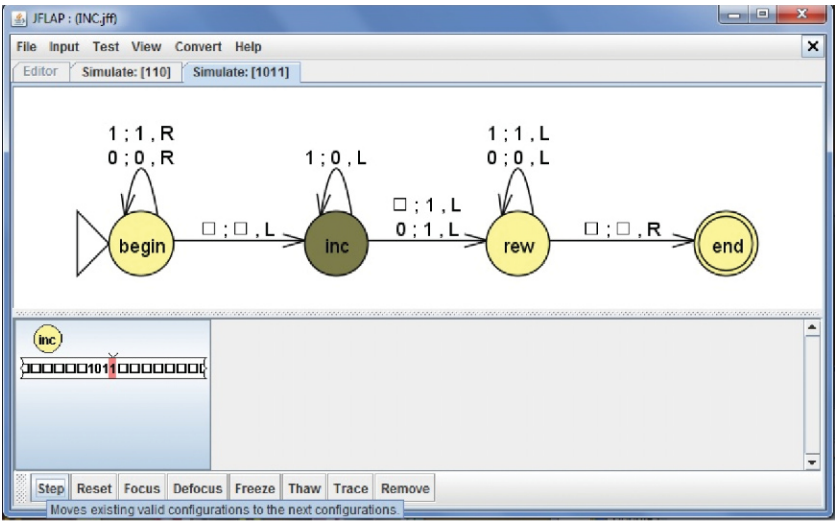


Рис. 1. Интерфейс программы JFLAP

В нижней части экрана отображается лента с отмеченной позицией головки. Также цветом выделяется текущая позиция (названия позиций соответствуют названиям позиций примера из книги). На рисунке в текущей позиции *inc* машина готова заменить 1 на 0. Потактовая работа осуществляется нажатием на «Step».

Файл, в котором сохраняется диаграмма МТ, имеет формат XML. Это позволяет непосредственно задавать МТ, минуя графический редактор (например, с помощью Notepad или какого-либо XML-редактора). Подобное представление можно считать языком описания МТ-диаграмм. В нем задаются не только пятерки переходов, но и координаты вершин для отображения.

Этот язык описания можно сделать более наглядным. На рисунке ниже приведен интерфейс программы, разработанный автором. В панели слева загружено описание МТ из файла B_MT.txt, выполняющего ту же работу, что и в предыдущем примере (прибавление 1). По

кнопке «Рисовать» строится изображение на панели справа, которая включает изображение ленты, головки и диаграммы МТ. Задаются входная цепочка и ее положение на ленте. По кнопке «Такт» машина начинает работать. В описании МТ можно задать параметры рисунка (цвет, толщина линий, размеры вершин и т. п.). Также задаются лента и форма головки. Строки, начинающиеся с «Q», задают вершины с координатами и меткой внутри. Строки «f:» – правила перехода. В квадратных скобках указываются три элемента «пятерки»: символ входной цепочки, символ замена и действие. Для петель (знак «<>» после Q[...]) задается угол направления петли в градусах. Для дуг (знак «->») – сторона выпуклости дуги (r или l) и величина отклонения дуги.

Для генерации лексического и синтаксического анализаторов использовалась система COCO/R. Атрибутная грамматика для анализа загружается по кнопке «Грамматика» и может оперативно редактироваться. Язык реализации – C#.

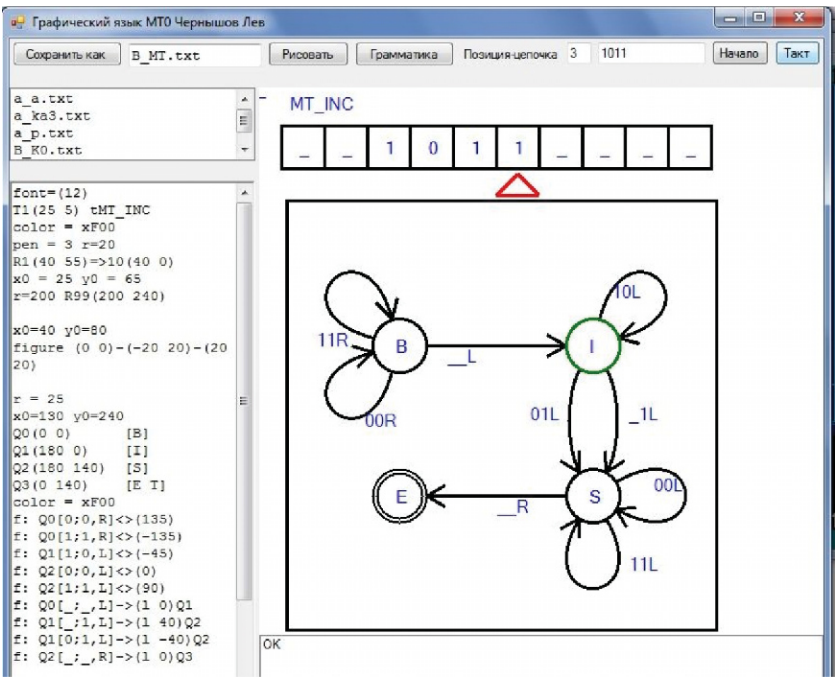


Рис. 2. Интерфейс программы моделирования автоматов

Можно надеяться, что приведенные примеры стимулируют интерес к рассмотренным инструментальным средствам и будут полезны при изучении основ теоретического программирования.

Библиография

1. Post E. L. J. Symbol. Logic. – 1936. – P. 544–546.
2. Клини С. К. Введение в метаматематику: пер. с англ. – М.,1957. (Переиздана изд. «Либроком» в 2008 г.)
3. Трахтенброт Б. . Алгоритмы и машинное решение задач: В серии «Популярные лекции по математике». – М.: Гостехиздат, 1957. (2-е изд. – 1960 г.)
4. Тьюринг А. Может ли машина мыслить? (С приложением статьи Дж. фон Неймана «Общая и логическая теория автоматов»: Пер. и примечания Ю. В. Данилова). – М.: ГИФМЛ, 1960. (Оригинал опубликован в 1950 г.) // http://www.gumer.info/bibliotek_Buks/Science/tyuring/mog_mash.php
5. Мальцев А. И. Алгоритмы и рекурсивные функции. – М.: Наука, 1965. (2-е изд. – 1986 г.)
6. Трахтенброт Б. А. Алгоритмы и вычислительные автоматы. – М.: «Сов. радио», 1974.
7. Ахо А., Ульман Дж. Теория синтаксического анализа, перевода и компиляции: в 2 т., пер. с англ. – М.: Мир, 1978.
8. Успенский В. А. Машины Поста. В серии «Популярные лекции по математике». – М.: Наука, 1979.
9. Хопкрофт Дж., Мотвани Р., Ульман Дж. Введение в теорию автоматов, языков и вычислений. – 2-е изд., пер. с англ. – М.: Издательский дом «Вильямс», 2002.
10. Мозговой М. В. Классика программирования: алгоритмы, языки, автоматы, компиляторы. Практический подход. – СПб.: Наука и техника, 2006.
11. Душкин Р. В. Функциональное программирование на языке Haskell. – М.: ДМК Пресс, 2007.

Предметный указатель

Б

В

Г

Д

Е

З

И

К

Л

M

Н

О

П

P

C

Т

У

Ф

Х

Ц

Ч

Книги издательства «ДМК Пресс» можно заказать в торгово-издательском холдинге «АЛЬЯНС-КНИГА» наложенным платежом, выслав открытку или письмо по почтовому адресу: 123242, Москва, а/я 20 или по электронному адресу: **orders@alians-kniga.ru**.

При оформлении заказа следует указать адрес (полностью), по которому должны быть высланы книги; фамилию, имя и отчество получателя. Желательно также указать свой телефон и электронный адрес.

Эти книги вы можете заказать и в интернет-магазине: **www.alians-kniga.ru**.

Оптовые закупки: тел. (499) 782-38-89; электронный адрес **books@alians-kniga.ru**.

Чарльз Петцольд

Читаем Тьюринга

Путешествие по исторической статье Тьюринга
о вычислимости и машинах Тьюринга

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com
Перевод *Борисов Е. В., Чернышов Л. Н.*
Корректор *Синяева Г. И.*
Верстка *Чаннова А. А.*
Дизайн обложки *Мовчан А. Г.*

Формат 60×90 1/16 .
Гарнитура «Петербург». Печать офсетная.
Усл. печ. л. 32. Тираж 200 экз.
№

Веб-сайт издательства: www.дмк.рф

Книга, которую вы держите в руках, принадлежит перу известного американского популяризатора Чарлза Петцольда.
В ней автор исследует главную работу Алана Тьюринга, посвященную проблеме разрешимости. Именно в этой работе впервые появились знаменитые машины Тьюринга, ставшие на многие годы универсальной теоретической концепцией computer science.
Автор тонко и деликатно проведет вас по самым потаенным уголкам, из которых родились на свет современные компьютеры и современное программное обеспечение.

Читателя ждет захватывающее путешествие в прошлое, из которого получилось наше настоящее и развивается будущее.

Internet-магазин
www.dmkpress.com

Книга-почтой:
orders@alians-kniga.ru

Оптовая продажа:
"Альянс-книга"
(499)725-5409
books@alians-kniga.ru

ДМК ИЗДАТЕЛЬСТВО
www.дмк.рф

ISBN 978-5-97060-010-8