Читателю, мозг которого – не машина Тьюринга[1]

[1] G. Longo [279, стр. 21].

Оглавление

Предисловие от издательства 9

Вступительное слово 10

Предисловие 11

Глава 1. От Entscheidungsproblem к проблеме остановки 14

1.1. Почему проблема остановки получила такое название? 14
1.2. Доказательство в стихах 15
1.3. Программы на псевдокоде 18
1.4. Остановка программы 19
1.5. Неостанавливающиеся программы 20
1.6. Циклы и неостанавливающиеся программы 21
1.7. Проблема остановки 22
1.8. Бесконечно много разрешимых случаев 23
1.9. Бесконечный анализатор 25
1.10. Теорема об остановке 25
1.11. Может ли человек решить проблему остановки? 27
1.12. Тезис Чёрча–Тьюринга 28
1.13. Существует ли квантовая проблема остановки? 33
1.14. Языки программирования с разрешимой проблемой остановки 40
1.15. Перечислители 40
1.16. Приближенные решения проблемы остановки 41

Глава 2. Неполнота и остановка 46

2.1. Евклид, Гильберт и аксиоматический метод 46
2.2. От теоремы об остановке к теореме о семантической неполноте 54
2.3. От теоремы о семантической неполноте к теореме об остановке 57
2.4. Минимальные программы 58
2.5. Алгоритмическая сложность и несжимаемость 60
2.6. Разрешима ли несжимаемость? 64
2.7. Вычислима ли алгоритмическая сложность? 65
2.8. Кодирование проблемы остановки 65
2.9. Алгоритмическая и временная сложность 70

Глава 3. Знаменитые математические утверждения и проблема остановки 72

3.1. Бесконечность множества простых чисел 72
3.2. Принцип Дирихле 73

3.3. Гипотеза Гольдбаха и другие проблемы теории чисел 73
3.4. Десятая проблема Гильберта 75
3.5. Теорема о четырех красках 77
3.6. Гипотеза Римана 78
3.7. Теорема Эйлера о разбиении целого числа 79
3.8. Гипотеза Коллатца 80
3.9. Проблема равенства P и NP 81
3.10. Конечно опровержимые утверждения 84
3.11. Вычисление сложности класса математических утверждений 86
3.12. Универсальная беспрефиксная двоичная машина Тьюринга 87
3.13. Двоичное кодирование программ для регистровой машины 88
3.14. Методы программирования 90
3.15. Мера сложности для Π_1-утверждений 93
3.16. Конкретные сложности 93

Глава 4. Интересные применения теоремы об остановке 101

4.1. Можно ли для тестирования и доказательства правильности программы ограничиться одним лишь компьютером? 101
4.2. Можно ли написать антивирусную программу, которая не будет нуждаться в обновлениях? 103
4.3. Можно ли написать идеально безопасную операционную систему? 105
4.4. Существуют ли неразрешимые проблемы в программной инженерии? 106
4.5. Поведение программы 108
4.6. Небезопасность сети 110
4.7. Бесконечная делимость 111
4.8. Есть ли у физики иммунитет к невычислимости? 112
4.9. Сила и слабость случайности 114
4.10. Реляционные базы, теория Рамсея и невозможность полного беспорядка 117
4.11. Спонтанные корреляции в больших данных: «Избыток информации внешне походит на недостаток информации» 122
4.12. Квантовая случайность 129
4.13. Проблема остановки в шахматах и в игре на четность 141
4.14. Искусственный интеллект: могущество и пределы 144
4.15. Свобода воли и случайность 164
4.16. Существует ли универсальный генератор изображений? 171

Глава 5. Проблема остановки, и не только 181

5.1. И снова о теории вычислимости 181
5.2. Машины Тьюринга 182
5.3. Еще о циклах и неостанавливающихся программах 184
5.4. Теорема о разрешимости: формальное доказательство 184
5.5. Конечная версия теоремы об остановке 185
5.6. Теорема о принятии 185
5.7. Теорема Райса – формальное доказательство 186
5.8. Теорема Спекера 187

5.9. Можно ли «экспериментально проверить» невычислимость? 190
5.10. Насколько «неразрешима» проблема остановки? 195
5.11. Большинство программ останавливаются быстро или не останавливаются никогда 196
5.12. Насколько велико множество спонтанных корреляций? 202

Глава 6. Формальный подход к неполноте и остановке 204

6.1. Аксиоматические системы 204
6.2. Автоматизированное доказательство теоремы об остановке 206
6.3. Унифицированное представление теорем о неполноте, основанное на теореме об остановке 209
6.4. Необычно длинные доказательства 214
6.5. Является ли гёделева неполнота исключением? 219
6.6. Любая функция вычислима 221

Глава 7. Исторические и философские замечания 224

7.1. Предтечи теории вычислимости 224
7.2. Entscheidungsproblem и Тьюринг 227
7.3. Проблема остановки: неправильная версия 228
7.4. ChatGPT-4 о проблеме остановки 229
7.5. Еще об истоках проблемы остановки 231
7.6. Программные средства доказательства 233
7.7. Почему и что мы вычисляем? 234
7.8. Краткая история неразрешимости 236

Послесловие 241

Библиография 243

Предисловие от издательства

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте *www.dmkpress.com*, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу *dmkpress@gmail.com*; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу *http://dmkpress.com/authors/publish_book/* или напишите в издательство по адресу *dmkpress@gmail.com*.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг – возможно, ошибку в основном тексте или программном коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу *dmkpress@gmail.com*, и мы исправим это в следующих тиражах.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательство «ДМК Пресс» очень серьезно относится к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу *dmkpress@gmail.com*.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Вступительное слово

Двадцатый век стал свидетелем нескольких важных концептуальных революций. Сначала, конечно, теория относительности – специальная и общая. Но еще более революционными стали квантовая механика, со всех сторон обсуждавшаяся философами, с ее суммами Фейнмана по всем путям и запутыванием кубитов, а в математике – целый пласт работ, посвященных такой, на первый взгляд, безобидной «проблеме остановки», которая тоже скрывает глубокий философский подтекст, но известна далеко не так широко. И вот теперь Кристиан Калюд в этой весьма интересной и разнообразной по тематике книге рассказал, почему проблема остановки является такой революционной и важной. Рискну даже сказать, излюбленной.

Он разворачивает перед нами различные варианты проблемы остановки, многочисленные области ее применения, демонстрируя плодородие и богатство почвы, на которой на протяжении большей части столетия взращивало новый концептуальный аппарат небольшое сообщество, и ставя себе целью досконально разобраться в приложениях такой, казалось бы, непритязательной идеи.

В результате получился на удивление захватывающий путеводитель по проблеме остановки, красочная мозаика новых идей и концепций, описывающая новую веху в развитии оснований математики.

Отправьтесь в путешествие с Калюдом в эту страну блистательных озарений. В век, когда над нами нависла угроза искусственного суперинтеллекта, находишь особое удовольствие и гордость от созерцания того, на что способен раскрепощенный человеческий дух.

Грегори Хайтин,
автор книги «Algorithmic Information Theory» (CUB Press, 1987)

Предисловие

Компьютеры кажутся такими мощными и вездесущими, что возникает искушение поверить в то, что рано или поздно они решат все проблемы. Но верно в точности противоположное. Компьютеры, что классические, что нетрадиционные – типа квантовых, – существенно ограничены: проблемы, неразрешимые для компьютеров, не только существуют, но и значительно (в смысле, который будет точно определен в этой книге), превосходят в числе те, что компьютеры могут решить. Более того, если доказано, что проблема алгоритмически неразрешима, то таковой она пребудет *во веки вечные*.

Какого рода проблемы компьютеры не могут решить, т. е. какие проблемы (алгоритмически) неразрешимы? Наверное, это что-то экзотическое, существующее только в воображении математиков? Или что-то второстепенное, не имеющее значения для практической информатики? Ни то, ни другое: даже некоторые задачи, ежедневно возникающие перед людьми, алгоритмически неразрешимы.

В этой книге рассказывается о «проблеме остановки», пожалуй, самой знаменитой (хотя малоизвестной широкой публике) неразрешимой проблеме: может ли алгоритм за конечное время решить, остановится ли когда-нибудь произвольная компьютерная программа?

Поначалу этот вопрос кажется мелким и праздным: ну кому это может быть интересно? Ну запустите вы программу и дождитесь, когда она остановится. Но что, если все разумные сроки – неделя, год, десять лет – истекли, а программа всё не останавливается? Можно ли сделать вывод, что она не остановится никогда? Не нужно долго думать, чтобы понять, что утверждение «программа никогда не остановится» не станет верным, как бы долго ни работало вычисление. На самом деле *отрицательный* ответ на поставленный вопрос был дан больше 70 лет назад[1].

Вам еще не стало интересно? Нет? Тогда, может быть, стоит добавить, что по той же причине отрицательны ответы на следующие вопросы:

а) Может ли математика развиваться усилиями одних лишь компьютеров, нуждающихся не более чем в дюжине математиков?
б) Можно ли проводить тестирование и доказательство правильности программ только с помощью компьютеров, с привлечением лишь немногих программистов?
в) Можно ли написать антивирус, который не будет требовать обновлений?
г) Можно ли написать стопроцентно безопасную операционную систему?
д) Решит ли ИИ все практические проблемы человечества?

Все эти вопросы, конечно, интересны и вызывают беспокойство, но они ограничены лишь миром компьютеров… Затрагивает ли проблема остановки, пря-

[1] Для читателей, интересующихся историей, сообщим, что не Тьюринг поставил эту задачу и не он доказал ее неразрешимость.

мо или косвенно, другие области знания? В нашем рассказе встретится немало и таких проблем. Хотите примеров? Пожалуйста. Наделен ли человек свободой воли или наш мозг автоматически генерирует мысли и решения? Возможен ли полный беспорядок? Влияют ли случайные корреляции на науку, экономику и биологию? Допускают ли шахматные правила[1] бесконечную партию?

Некоторые вопросы, касающиеся остановки, трудны, даже если технически не являются неразрешимыми для компьютеров. Например, прекращать ли работу системы искусственного жизнеобеспечения, брак, утративший смысл для супругов, старую дружбу и т. д. Мучительные размышления над такими вопросами могут вызывать сильное расстройство, беспокойство и делать людей несчастными. Универсальных решений нет, но есть стратегии [214], которые подсказывают, делать следующий ход или не стоит – останавливаться или нет. Некоторые процедуры, относящиеся к неразрешимости, могут предложить новые способы распутывания этих загадок.

Мы интерпретируем решение проблемы остановки не только в стандартном отрицательном ключе (невозможность), но и в положительном. Примером может служить продуктивность. В книге «Getting Things Done: The Art of Stress-Free Productivity» (Penguin Books, 2015) Д. Аллен подчеркивает, что определение «следующего действия» существенно в любой эффективно функционирующей системе повышения продуктивности. Многие «действия» можно довести до конца алгоритмически. Но не все. Вот несколько примеров: а) решение трудной математической задачи, б) сочинение художественного произведения, завоевавшего премию, в) разработка новой бизнес-идеи. Может ли ИИ успешно закончить неразрешимые задачи, в частности перечисленные выше? До некоторой степени он может помочь, но довести дело до конца не способен. К. Ньюпорт[2] говорил, что «способность единообразно завершать неразрешимые задачи приобретает все большую ценность в нашей информационной экономике». Почему? Потому что навыки, необходимые для решения нетривиальных примеров неразрешимых проблем, требуют глубоких знаний и опыта, творческой интуиции, настойчивости, физической проворности и много еще чего. И что еще важнее, эти навыки невозможно автоматизировать или без труда препоручить кому-то другому.

Иногда мои студенты задают необычные вопросы о проблеме остановки. Вот несколько таких вопросов и ответов на них: i) ваше любимое применение неразрешимости проблемы остановки? Связь между проблемой остановки и теоремой о неполноте Гёделя; ii) самое неожиданное применение неразрешимости проблемы остановки? После того как вы поняли проблему остановки, таковых нет; iii) самое дорогостоящее следствие неразрешимости проблемы остановки? Невозможность автоматизировать тестирование семантической правильности программы.

В этой книге предлагается путешествие, которое просветит и заставит думать читателя[3], сочетает неформальный подход с формальными доказательствами

---

[1] Включая правило ничьей, которое диктует окончание после конечного числа ходов.

[2] *www.calnewport.com/blog/2014/11/30/on-undecidable-tasks-or-how-alan-turing-can-help-you-earn-a-promotion/*.

[3] Не пропускайте сноски «Упражнения для читателя».

и познакомит со многими классическими и новыми результатами, примерами и приложениями. Она призвана возбудить любопытство у широкого круга читателей, включая математиков, специалистов по информатике, физиков, биологов, обществоведов, философов, а также студентов и аспирантов – всех, кто интересуется пределами вычислимости и различными способами их преодоления.

Писать эту книгу было труднее, чем я думал, но и награда оказалась более щедрой, чем я мог рассчитывать. Ничего не получилось бы без поддержки Елены, кратковременных разочарований и сотрудничества со многими коллегами и друзьями.

Я в большом долгу перед всеми соавторами статей, использованных в этой книге: A. A. Abbott, J. M. Agüero Trejo, E. H. Allen, E. Calude, K. F. Celine, J. Conder, D. Desfontaines, M. J. Dinneen, M. Dumitrescu, A. Fedorov, Z. Gao, N. J. Hay, S. Heidari, R. Hua, N. Huang, S. Jain, H. Jürgensen, B. Khoussainov, F. Kroon, A. Kulikov, S. Legg, J. P. Lewis, W. Li, G. Longo, S. Marcus, W. Merkle, F. W. Meyerstein C. Müller, G. Păun, R. Navarathna, N. Poznanović, M. S. Queen, A. Salomaa, C. Shu, J. Sifakis, L. Staiger, M. Stay, F. Stephan, D. Thompson, K. Svozil, I. Ţevy, M. Zimand и D. Vaida.

За (долгое) время работы над книгой я получил много пользы от обсуждений и замечаний по различным рассмотренным вопросам со стороны своих соавторов: J.-P. Delahaye, S. George, T. Murphy, S. Yang, N. Yanofsky, L. Velez. Огромную помощь также оказал коллектив издательства World Scientific, особенно редактор книги L. F. Kwong (мы снова встретились!), специалист по LaTeX D. R. Babu и выпускающий редактор G. Lee – всем им выражаю горячую благодарность.

Огромное спасибо Ю. Колежицкому, который внимательно вычитывал различные черновые варианты книги и внес немало предложений, улучшивших ее.

Я также признаюсь в использовании программы Grammarly в некоторых частях книги.

Я благодарю В. Хайтин за разрешение использовать принадлежащее ей изображение Дж. Хайтина, Ж.-П. Делахэя за разрешение использовать таблицу «Краткая история неразрешимости» из его статьи «Presque tout est indécidable», *Pour La Science* 375 (2009), 88–93, а также Дж. Лонго и Ю. Матиясевича за разрешение использовать их фотографии.

Все рисунки, взятые из Википедии, отвечают требованиям, изложенным по адресу *commons.wikimedia.org/wiki/Commons:Reusing_content_outside_Wikimedia*.

Кристиан С. Калюд,
Оклендский университет,
Новая Зеландия

Ноябрь 2023

Глава 1

От Entscheidungsproblem к проблеме остановки

В этой главе мы обсудим проблему остановки. Начнем с алгоритмов, псевдокода и примеров программ, которые останавливаются и не останавливаются, а затем сформулируем проблему, взглянем на разрешимые случаи, докажем теорему об остановке (неразрешимость) и исследуем смежные проблемы и решения.

1.1. Почему проблема остановки получила такое название?

Проблема остановки исторически важна, потому что для нее первой была доказана неразрешимость «классическими» компьютерами, т. е. устройствами того же типа, что стоящее на столе у читателя. Классические компьютеры – это физические реализации математического понятия детерминированной машины Тьюринга [151][1].

Проблема остановки – математический вопрос, относящийся к детерминированным машинам Тьюринга[2]. Согласно Коуплэнду [138, стр. 40]:

> Проблема остановки названа так (и, по-видимому, впервые сформулирована) Мартином Дэвисом в работе [147, стр. 70–71]. Утверждение о том, что проблему остановки нельзя решить с помощью вычислительной машины, называется «теоремой об остановке». (Часто приходится слышать, что Тьюринг сформулировал и доказал теорему об остановке в статье «О вычислимых числах»[3], но, строго говоря, это не так.)

---

[1] При классическом подходе к вычислениям информация хранится в виде битов, логически представленных цифрами 0 (выключено) или 1 (включено). Для классических вычислений используются детерминированные процессоры типа x86 или ARM; многократная подача им на вход одних и тех же значений приводит к одному и тому же результату. С другой стороны, в нетрадиционных компьютерах, например квантовых, биологических и других [70], используются другие средства хранения и обработки информации.

[2] Тьюринг изобрел машину Тьюринга и описал ее в работе [397]. Согласно работе [270], Гёдель развивал идею вычислительной машины, очень похожей на машину Тьюринга.

[3] [397].

Результат, сформулированный в работе [147, стр. 70], звучит так:

Теорема 2.2. *Существует машина Тьюринга, для которой проблема остановки рекурсивно неразрешима*[1].

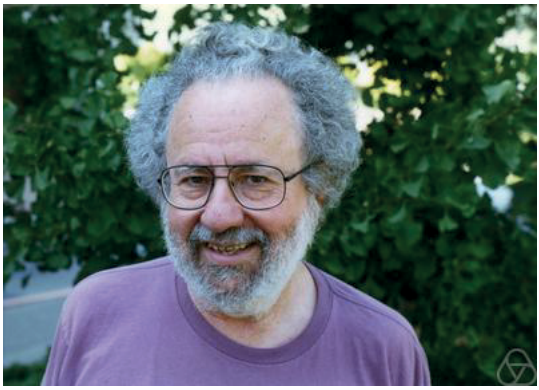

Рис. 1.1. М. Дэвис[2]

В своей книге 1958 года Дэвис упоминает неформальное доказательство в монографии Клини [248, p. 382]. Как отмечал Коуплэнд, работа [397] некорректно указана как место, где впервые была сформулирована и доказана теорема об остановке. В чем же причина того, что это ошибочное утверждение так настойчиво повторяется вновь и вновь? Мы подробно ответим на этот вопрос в разделе 7.5.

1.2. Доказательство в стихах

Дж. Паллум из Эдинбургского университета опубликовал следующее забавное поэтическое доказательство теоремы об остановке [329][3]:

No general procedure for bug checks will do.
Now, I won't just assert that, I'll prove it to you.
I will prove that although you might work till you drop,
you cannot tell if computation will stop.

For imagine we have a procedure called P
that for specified input permits you to see
whether specified source code, with all of its faults,
defines a routine that eventually halts.

---

[1] Рекурсивная неразрешимость – старое название эффективной (алгоритмической) неразрешимости или невычислимости.

[2] Источник: *en.wikipedia.org/wiki/Martin_Davis_(mathematician)#/media/File:Martin_Davis.jpg*.

[3] Первоначально опубликовано в *Mathematics Magazine* 73, 4, 319–320, в октябре 2000 года.

You feed in your program, with suitable data,
and P gets to work, and a little while later
(in finite compute time) correctly infers
whether infinite looping behavior occurs.

If there will be no looping, then P prints out 'Good'.
That means work on this input will halt, as it should.
But if it detects an unstoppable loop,
then P reports 'Bad!' which means you're in the soup.

Well, the truth is that P cannot possibly be,
because if you wrote it and gave it to me,
I could use it to set up a logical bind
that would shatter your reason and scramble your mind.

Here's the trick that I'll use and it's simple to do.
I'll define a procedure, which I will call Q,
that will use P's predictions of halting success
to stir up a terrible logical mess.

For a specified program, say A, one supplies,
the first step of this program called Q I devise
is to find out from P what's the right thing to say
of the looping behavior of A run on A.

If P's answer is 'Bad!', Q will suddenly stop.
But otherwise, Q will go back to the top,
and start off again, looping endlessly back,
till the universe dies and turns frozen and black.

And this program called Q wouldn't stay on the shelf;
I would ask it to forecast its run on itself.
When it reads its own source code, just what will it do?
What's the looping behavior of Q run on Q?

If P warns of infinite loops, Q will quit;
yet P is supposed to speak truly of it!
And if Q's going to quit, then P should say 'Good'.
Which makes Q start to loop! (P denied that it would.)

No matter how P might perform, Q will scoop it:
Q uses P's output to make P look stupid.
Whatever P says, it cannot predict Q:
P is right when it's wrong, and is false when it's true!

I've created a paradox, neat as can be
and simply by using your putative P.
When you posited P you stepped into a snare;
Your assumption has led you right into my lair.

So where can this argument possibly go?
I don't have to tell you; I'm sure you must know.
We've proved that there just cannot possibly be
a procedure that acts like the mythical P.

You can never find general mechanical means
for predicting the acts of computing machines;
it's something that cannot be done. So we users
must find our own bugs. Our computers are losers!

Не будучи в состоянии дать стихотворный перевод, мы приведем лишь прозаический подстрочник.

Не существует общей процедуры нахождения ошибок.
И это не голословное утверждение, я его докажу.
Я докажу, что даже если вы будете работать до упаду,
Все равно не сможете сказать, остановится вычисление или нет.

Представьте, что имеется процедура, назовем ее P,
Которая для заданного входа позволяет узнать,
Определяет ли заданный исходный код, со всеми его ошибками,
Процедуру, которая в конечном итоге остановится.

Вы подаете на вход свою программу и данные для нее,
P начинает работать и спустя некоторое время
(Конечное время вычисления) приходит к правильному выводу
О том, имеет место бесконечный цикл или нет.

Если зацикливания нет, то P печатает «Хорошо».
Это означает, что обработка этого входа остановится, как и должно быть.
Но если она обнаруживает бесконечный цикл,
То печатает «Плохо!», это значит, что вы попали в беду.

Но все дело в том, что такой P не может быть,
Потому что если бы вы ее написали и вручили мне,
То я смог бы построить логическое рассуждение,
Которое разрушило бы ваши доводы и взорвало ваш мозг.

И вот что я бы сделал.
Я определил бы процедуру, назову ее Q,
Которая, используя предсказания P об успешной остановке,
Создает полный логический хаос.

Для заданной программы, скажем A,
Первый шаг предлагаемой мной процедуры Q –
Узнать, что P говорит о
Зацикливании при запуске A со входом A.

Если P отвечает «Плохо!», то Q немедленно останавливается.
В противном случае Q возвращается в начало
И будет крутиться в цикле до тех пор, пока
Вселенная не умрет, став черной и холодной.

Но эта программа Q не останется стоять на полке.
Я попрошу ее предсказать результат ее выполнения для нее самой:
Что будет, если подать ей на вход ее собственный исходный код?
Что можно сказать о зацикливании Q, примененной к Q?

Если P предупреждает о бесконечных циклах, то Q остановится.
Но ведь предполагается, что P должна ясно заявить об этом!
И если Q останавливается, то P должна сказать «Хорошо».
А раз так, то Q входит в бесконечный цикл! (P отрицала такую возможность.)

Как бы ни работала P, Q ее обставит:
Q использует выход P, оставляя P в дураках.
Что бы P ни говорила, она не сможет предсказать поведение Q:
P права, когда ошибается, и лжет, говоря правду!

Я создал самый настоящий парадокс,
Просто воспользовавшись вашей предполагаемой P.
Постулировав P, вы угодили в ловушку;
Ваше предположение завело вас аккурат в подстроенную мной западню.

Ну и куда же ведет это рассуждение?
Тут и говорить ничего не надо, я уверен, что вы сами все поняли.
Мы доказали, что не может существовать процедуры,
Работающей, как эта мифическая P.

Вам никогда не найти общего механического средства,
Способного предсказывать поведение вычислительных машин, –
Это попросту невозможно. Так что нам, пользователям,
Придется самим искать свои ошибки. Компьютер нам не помощник.

Читатель, которого это доказательство убедило, может сразу переходить к следующей главе.

1.3. Программы на псевдокоде

Алгоритм – это множество подлежащих выполнению операций, или шагов. Алгоритм можно выполнить в уме, на бумаге или с помощью компьютера.

Псевдокод – это неформальный язык описания высокого уровня с похожим на английский язык синтаксисом. В нем используются структурные соглашения стандартных языков программирования, но предназначен он для чтения человеком, а не для исполнения машиной.

Псевдокод нужен не только инженерам; опытные программисты часто пишут псевдокод для обмена мыслями с коллегами или даже для себя, его можно встретить в статьях и книгах. Например, в науке о данных и в процессе веб-разработки псевдокод применяется для описания шагов алгоритма способом, понятным даже читателю с минимальными знаниями в области программирования. Его интутитивная понятность помогает при разработке и отладке.

Для псевдокода не существует стандартного синтаксиса; уровень детализации может быть различным и в некоторых случаях приближается к уровню формального языка программирования общего назначения, т. е. языка, эквивалентного по вычислительным возможностям универсальной машине Тьюринга [50].

В программах на псевдокоде используются типичные конструкции стандартных языков программирования, как то:

(1) присваивание (=),
(2) сравнения (=, ≠,_≤,_≥, <, >),
(3) арифметические операции (+, –, ×, /),
(4) логические операции (∨, ∧),
(5) ключевые слова (**read**, **print**, **do**, **for**, **stop**),
(6) условные предложения (**if-then-else**, **case**), итерация (**while**),
(7) операторные скобки (**begin**, **end**) и
(8) функции (**P**(*N*)).

Например, $i = 1$ обозначает присваивание значения 1 переменной i; команда $s = s + 1$ увеличивает текущее значение переменной s на 1; конструкция **while** $i \leqslant 100$ **do** – команда управления потоком, которая повторяет блок команд, пока выполняется булево условие $i \leqslant 100$.

Мы будем требовать, чтобы *каждая программа завершалась специальной командой* **stop**.

1.4. Остановка программы

Программа (на псевдокоде)

Program(1)
1 $i = 1$
2 $s = 0$
3 **while** $i \leqslant 100$ **do**
4 $\quad s = s + i$
5 $\quad i = i + 1$
6 **end while**
7 **print** s
8 **stop**

вычисляет сумму 1 + 2 + … + 100: она выводит число 5050 и останавливается[1]. Решить, останавливается эта программа или нет, просто: условие «$i \leqslant 100$» цикла в строках 3, 4, 5, 6 остается истинным только 100 раз, а на 101-й итерации оказывается ложным. Управление переходит к командам 7 и 8, которые печатают 5050 и завершают вычисление соответственно. Чтобы было проще рассуждать об этой программе, ее строки пронумерованы. Эти числа не являются частью программы, они несут метаинформацию.

1.5. Неостанавливающиеся программы

Как может программа работать вечно? На рис. 1.2 приведен пример такого поведения.

Рис. 1.2. Сначала Вы!

Пример такого рода можно получить, модифицировав программу PROGRAM(1):

Program(2)
1 $i = 1$
2 **while** $i \geqslant 1$ **do**
3 $\quad i = i + 1$
4 $\quad$ **print** i
5 **end while**
6 **stop**

[1] Восьмилетний Гаусс, которому суждено было стать королем математиков, поразил своего учителя быстрым вычислением этой суммы. Гаусс не пользовался компьютером, но заметил, что эту сумму можно представить как сумму пятидесяти слагаемых вида (1 + 100), (2 + 99), (3 + 98), …, (50, 51), каждое из которых равно 101, а значит, ее значение равно 50 × 51 = 5050. Чем больше вы думаете, тем меньше вычисляете, и наоборот!

Решение о том, останавливается ли PROGRAM(2), принять также легко. Условие «$i \geqslant 1$» в условии цикла **while** (строки 2, 3, 4, 5) удовлетворяется всегда; поэтому команды 2, 3 и 4 повторяются снова и снова: PROGRAM(2) никогда не останавливается, и команда **stop** не достигается. Эта программа печатает все целые положительные числа: 1, 2, 3, ...

1.6. Циклы и неостанавливающиеся программы

Чтобы лучше во всем разобраться, рассмотрим PROGRAM(2) более пристально: эта программа никогда не останавливается из-за бесконечного повторения команд в строках 2, 3 и 4. Выполнение программы можно описать следующей *бесконечной* последовательностью команд:

$$1, 2, 3, 4, 2, 3, 4, 2, 3, 4, \ldots 2, 3, 4, \ldots$$

Иными словами, PROGRAM(2) никогда не останавливается, потому что *цикл*, состоящий из строк 2, 3, 4, повторяется вечно, так что команда в строке 5 никогда не достигается. В математике последовательность $u_1, u_2, \ldots, u_n \ldots$ называется *периодической*, если она регулярно повторяется. Формально последовательность является периодической с периодом $T > 0$, если $u_{n+T} = u_n$ для всех $n \geqslant 1$. Наименьшее такое T называется наименьшим периодом (или просто периодом) последовательности. Например, последовательность 0, 1, 0, 1, 0, 1, ... периодическая с периодом 2, а последовательность 0, 1, 1, 1, 0, 1, 1, 1, ... – периодическая с периодом 4.

В информатике *бесконечным циклом* называется бесконечно повторяющееся множество команд. Например, программа, которая исполняет команды 1,2,3,4,5,6 в следующем порядке

$$1, 2, 3, 4, 5, 6, 3, 4, 3, 4, 3, 4, \ldots,$$

имеет цикл (3, 4) и, следовательно, никогда не останавливается. Эта последовательность периодическая. Обычно программа содержит бесконечный цикл, если исполняет периодическую последовательность команд при обработке входа. Однако можно написать программу, содержащую бесконечный цикл, которая для некоторых входов останавливается – и этим отличается от PROGRAM(2)[1].

Программа, последовательность команд которой содержит бесконечный цикл, никогда не останавливается, а решить, содержит ли программа цикл, просто[2]. А как насчет обратного утверждения? Верно ли, что любое вычисление, которое никогда не останавливается, обязано содержать бесконечный цикл?

Ответ отрицательный, примером служит PROGRAM(3).

```
Program(3)
1    i = 1
2    while i ⩾ 1 do
3        j = 1
4        while j ⩽ i do
```

[1] Упражнение для читателя: напишите такую программу.

[2] Упражнение для читателя: почему?

```
5          print 0
6          j = j + 1
7       end while
8       i = i + 1
9    end while
10   stop
```

Действительно, последовательность команд, исполняемых PROGRAM(3), начинается так:

$$1, 2, 3, 4, 5, 6, 7, 8, 9, 2, 3, 4, 5, 6, 7, 4, 5, 6, 7, 8, 9, \ldots$$

– и никогда не заканчивается. Однако в этой последовательности нет бесконечного цикла, потому что количество повторений последовательности 4, 5, 6, 7 растет бесконечно.

Командой цикла называется команда управления потоком, которая позволяет повторять блок команд. Например, **while** является такой командой для программ, рассматриваемых в этой книге. Некоторые циклы конечны, например цикл в PROGRAM(1). Бесконечный цикл, образованный группой команд 2, 3, 4, 5, 6, 7, 8, 9, показывает, что PROGRAM(3) никогда не останавливается. Другой пример – программа, содержащая всего две команды, *a* и *b*, исполняемые в следующем порядке:

$$\text{a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb, aaaa}, \ldots \tag{1.1}$$

Эта бесконечная двоичная последовательность содержит каждую двоичную строку[1]. Можно написать программу с командами, помеченными *a* и *b*, выполнение которой описывается точно последовательностью (1.1): эта программа никогда не останавливается, но бесконечного цикла в ней нет[2]. Программы такого типа – не единственные неостанавливающиеся программы, но составляют их «большинство»[3].

1.7. Проблема остановки

Как мы видели в разделах 1.4 и 1.5, принять решение о том, что PROGRAM(1) всегда останавливается, а PROGRAM(2) не останавливается никогда, было просто[4]. Так, может быть, немного подумав, мы сможем найти решение, работающее для любой программы? Но как тогда насчет обсуждения в разделе 1.6?

> *Проблема остановки* (для программ) заключается в том, чтобы решить, останавливается ли произвольная программа.

Для решения проблемы остановки (для программ) нам нужна программа, называемая **анализатором остановки**, которая за конечное время правильно

---

[1] Упражнение для читателя: почему?

[2] Упражнение для читателя: напишите такую программу.

[3] Упражнение для читателя: формализуйте понятия «большинства» и обоснуйте высказанное утверждение.

[4] Заметим, что обе программы не получают входных данных.

Конец ознакомительного фрагмента.
Приобрести книгу можно
в интернет-магазине
«Электронный универс»
e-Univers.ru