Московский государственный технический университет имени Н.Э. Баумана

Факультет «Фундаментальные науки»

Кафедра «Математическое моделирование»

А.Н. Канатников

МАТЕМАТИЧЕСКАЯ ЛОГИКА И ТЕОРИЯ АЛГОРИТМОВ

Конспект лекций

Для студентов кафедры ИУ9

Москва
2010

12. НЕРАЗРЕШИМЫЕ АЛГОРИТМИЧЕСКИЕ ПРОБЛЕМЫ

Одна из причин разработки теории алгоритмов — проверка существования алгоритмов, решающих те или иные задачи. Если заявлен некоторый алгоритм, решающий ту или иную задачу, то проверить его качества — в основном техническая задача. Но если есть подозрение, что такого алгоритма не существует (например, алгоритма проверки формулы логики предикатов на истинность), обосновать это подозрение можно только тогда, когда алгоритм из интуитивного понятия трансформирован в строгое математическое понятие.

Разумеется, нельзя доказать, что та или иная формализация понятия алгоритма охватывает все мыслимые алгоритмы. В этом смысле тезис Черча (или Тьюринга, Маркова) — неразрешимая проблема.

Выделим так называемые ***массовые алгоритмические проблемы***. Массовой называют проблему существования алгоритма, решающего бесконечную серию однотипных задач. Если удается арифметизировать массовую проблему, т.е. перенумеровать все частные случаи серии однотипных задач и представить в символической форме возможные решения каждой задачи, то задача сводится к построению некоей функции h, которая каждой задаче с номером n ставит в соответствие ее решение $h(n)$. Символические записи всех возможных решений тоже можно перенумеровать. В этом смысле функция h будет функцией натурального аргумента. Существование алгоритма вычисления этой функции и есть математическая интерпретация разрешимости массовой проблемы. Дадим точное определение.

Пусть X — счетное множество, которое мы можем интерпретировать как множество слов в некотором алфавите A. Полагаем, что есть свойство, которым обладает или нет каждый элемент $x \in X$. Это свойство можно описать некоторым предикатом $P(x)$, определенным на множестве X. Массовая алгоритмическая проблема состоит в построении алгоритма $\mathfrak{A}$ в алфавите A, который применим к любому элементу $x \in X$, причем $\mathfrak{A}(x) = \Lambda$, если $P(x)$ истинно, и $\mathfrak{A}(x) \neq \Lambda$, если $P(x)$ ложно.

Массовая алгоритмическая проблема неразрешима, если такого алгоритма нет. Примером таких проблем является неразрешимость ряда формальных аксиоматических теорий (в частности, исчисления высказываний), т.е. отсутствие алгоритма, который по заданной формуле устанавливает, выводима эта формула в данной теории или нет.

Мы остановимся на одной внутренней проблематике теории алгоритмов.

12.1. Проблема распознавания самоприменимости

Если алфавит A содержит не менее двух букв, любой нормальный алгорифм можно записать в виде слова $\mathfrak{A}^3$ в том же алфавите A. Для этого достаточно в алфавите A выделить подалфавит A_0 из двух букв и перевести изображение алгоритма $\mathfrak{A}$ в алфавит A_0. То, что получится, назовем ***записью нормального алгорифма***. Эта запись может быть исходным словом для самого нормального алгорифма. Можно поставить вопрос, применим ли нормальный алгорифм к своей записи? Может быть да, а может быть и нет. Например, алгоритм, добавляющий к исходному слову первую букву алфавита A, применим к любому слову, в том числе и к своей записи. А если нормальный алгорифм неприменим к словам, имеющим некоторые сочетания букв (например, к словам, содержащим первую букву подалфавита A_0), то он может быть неприменимым к своей записи.

Будем говорить, что ***алгоритм $\mathfrak{A}$ самоприменим***, если $!\mathfrak{A}(\mathfrak{A}^3)$. Если же $\neg!\mathfrak{A}(\mathfrak{A}^3)$, то говорим, что ***алгоритм $\mathfrak{A}$ несамоприменим***. Можно ли построить алгоритм, проверяющий, является ли данный алгоритм $\mathfrak{A}$ самоприменимым?

Теорема 12.1. Пусть алфавит A содержит не менее двух букв. Не существует нормального алгорифма $\mathfrak{R}$ в алфавите A, который для любого нормального алгорифма $\mathfrak{A}$ в алфавите A применим к $\mathfrak{A}^3$ тогда и только тогда, когда нормальный алгорифм $\mathfrak{A}$ несамоприменим.

◄ Предположим, что алгоритм $\mathfrak{R}$ существует. Тогда в силу своего определения $!\mathfrak{R}(\mathfrak{R}^3)$, если $\mathfrak{R}$ несамоприменим. Но эти два утверждения взаимно исключающие. ►

Доказанная, в общем-то тривиальная, теорема ограничивается лишь частным случаем нормальных алгорифмов, не использующих специальных символов. Однако общий случай можно свести к этому частному. Пусть алфавит A содержит не менее 4-х букв. Зафиксируем подалфавит A_0 из двух букв, которые используем для записи нормальных алгорифмов в алфавите A, и выделим подалфавит $A_1 \subset A$ из четырех букв, включающий в себя A_0.

Теорема 12.2. Пусть A содержит не менее четырех букв и $A_0 \subset A$ — двухбуквенный подалфавит. Не существует нормального алгорифма $\mathfrak{R}$ над алфавитом A_0, который для любого нормального алгорифма $\mathfrak{A}$ в алфавите A применим к $\mathfrak{A}^3$ тогда и только тогда, когда $\mathfrak{A}$ несамоприменим.

◄ Пусть такой алгоритм существует. Согласно теореме приведения, можно построить нормальный алгоритм $\mathfrak{R}_1$ в четырехбуквенном алфавите A_1, вполне эквивалентный алгоритму $\mathfrak{R}$ относительно алфавита A_0. Это значит, что для любого слова $X \in A_0^*$ имеем $!\mathfrak{R}(X) \Leftrightarrow !\mathfrak{R}_1(X)$. Для алгоритма $\mathfrak{R}_1$ можно построить его естественное расширение $\mathfrak{R}_2$ на алфавит A, так что для любого слова $X \in A_0^*$ имеем $!\mathfrak{R}_1(X) \Leftrightarrow !\mathfrak{R}_2(X)$. Мы получили, что алгоритм $\mathfrak{R}_2$ применим к тем и только к тем записям в A_0 нормальных алгорифмов в алфавите A, которые являются несамоприменимыми. Но такого нормального алгорифма не существует. Значит, и нормального алгорифма типа алгоритма $\mathfrak{R}$ тоже не существует. ►

Существование нормального алгорифма со свойством, описанным выше, все равно не решило бы проблему распознавания самоприменимости (несамоприменимости), поскольку для несамоприменимых алгоритмов наш алгоритм $\mathfrak{R}$ давал бы ответ, а для самоприменимых — нет, так как для таких алгоритмов он будет работать неограниченно долго. Модифицируем постановку задачи. ***Массовой проблемой распознавания несамоприменимости*** назовем проблему существования алгоритма $\mathfrak{P}$, применимого к записи в A_0 любого нормального алгорифма в алфавите A и удовлетворяющего условию: $\mathfrak{P}$ преобразует запись нормального алгорифма в пустое слово, если этот нормальный алгорифм несамоприменим, и в непустое слово в простивном случае. Оказывается, что эта массовая алгоритмическая проблема неразрешима.

Теорема 12.3. Пусть A содержит не менее четырех букв и $A_0 \subset A$ — двухбуквенный подалфавит. Не существует нормального алгорифма $\mathfrak{P}$ над алфавитом A_0, который дает пустое значение на слове $X \in A_0^*$ тогда и только тогда, когда X есть запись несамоприменимого нормального алгорифма.

◄ На самом деле это модификация предыдущей теоремы. Действительно, пусть нормальный алгорифм $\mathfrak{P}$ существует. Тогда его легко модифицировать, зацикливая в тех случаях, когда результатом является непустое слово (достаточно взять композицию $\mathfrak{P}$ с нормальным алгорифмом, описываемым групповой подстановкой $\xi \to \xi$, где ξ пробегает весь алфавит алгоритма $\mathfrak{P}$). В результате получим нормальный алгорифм с теми же свойствами, что и алгоритм $\mathfrak{R}$ из теоремы 12.2. Но такого алгоритма не существует. Значит, не существует и алгоритма $\mathfrak{P}$. ►

Доказанная теорема, в частности, утверждает и отсутствие нормального алгорифма, применимого ко всем записям нормальных алгорифмов в алфавите* A, но дающего в результате пустое

*В соответствии с формулировкой теоремы нормальный алгорифм $\mathfrak{P}$ применим к записям всех несамоприменимых алгорфмов и дает при этом пустое слово, а к записям других нормальных алгорифмов он может быть и неприменим, но если применим, то дает в результате непустое слово.

слово только для несамоприменимых нормальных алгорифмов. Это и означает, что массовая проблема распознавания несамоприменимости неразрешима. Нетрудно также сделать вывод о неразрешимости ***массовой проблемы распознавания самоприменимости***, поскольку из соответствующего алгоритма легко получить алгоритм распознавания несамоприменимости.

12.2. Проблема распознавания применимости алгоритма к слову

Схожая проблема — существование алгоритма, распознающего применимость данного алгоритма к произвольному слову. Пусть дан нормальный алгорифм $\mathfrak{A}$ над алфавитом A. Поставим вопрос: существует ли нормальный алгорифм $\mathfrak{P}$, который применим к любому слову в алфавите A, и дающий в результате пустое слово тогда и только тогда, когда к исходному слову применим (неприменим) нормальный алгорифм $\mathfrak{A}$?

Суть проблемы в следующем. Конечно, можно применить сам алгоритм $\mathfrak{A}$ к слову X. Однако это даст ответ только в случае, когда алгоритм $\mathfrak{A}$ применим к этому слову X. Неприменимость означает, что $\mathfrak{A}$ будет работать вечно, а мы на каждом шаге не знаем, остановится он через пару шагов или нет.

Разумеется, есть алгоритмы, для которых ответ очевиден. Например, тождественный алгоритм применим к любому слову, построение для него распознающего алгоритма тривиально: распознающий алгоритм должен всегда возвращать Λ. Но для всех ли алгоритмов имеется такое распознавание?

Теорема 12.4. Существует нормальный алгорифм $\mathfrak{C}$ над пятибуквенным алфавитом A_2, обладающий свойством: не существует нормального алгорифма $\mathfrak{P}$ над A_2, который преобразует в Λ те и только те слова в алфавите A_2, к которым неприменим $\mathfrak{C}$.

◀ Доказательство основано на существовании универсального алгорифма. Пусть $A_2 = a_0a_1\alpha\beta\delta$, полагаем $A_0 = a_0a_1$, $A_1 = A_0\alpha\beta$. Рассмотрим универсальный алгорифм $\mathfrak{R}$ для алфавита A_2. Мы можем его трасформировать в нормальный алгорифм $\mathfrak{C}$, в котором используются не изображения нормальных алгорфмов, а их записи в алфавите A_0. Действительно, нетрудно построить нормальный алгорифм $\mathfrak{S}_3$, переводящий запись нормального алгорифма в его изображение. Тогда композиция $\mathfrak{C} = \mathfrak{R} \circ \mathfrak{S}_3$ будет обладать свойством $!\mathfrak{C}(\mathfrak{A}^3\delta X) \simeq \mathfrak{R}(X)$.

Пусть для нормального алгорифма $\mathfrak{C}$ существует распознающий нормальный алгорифм $\mathfrak{P}$, т.е. $\mathfrak{P}(X) = \Lambda \Leftrightarrow \neg!\mathfrak{C}(X)$. В качестве X рассмотрим слова вида $\mathfrak{A}^3\delta\mathfrak{A}^3$ по всем нормальным алгорифмам $\mathfrak{A}$ в алфавите A_2. Мы заключаем, что $\mathfrak{P}(\mathfrak{A}^3\delta\mathfrak{A}^3) = \Lambda \Leftrightarrow \neg!\mathfrak{C}(\mathfrak{A}^3\delta\mathfrak{A}^3)$. Но $!\mathfrak{C}(\mathfrak{A}^3\delta\mathfrak{A}^3) \simeq !\mathfrak{A}(\mathfrak{A}^3)$. Поэтому $\mathfrak{P}(\mathfrak{A}^3\delta\mathfrak{A}^3) = \Lambda \Leftrightarrow \neg!\mathfrak{A}(\mathfrak{A}^3)$. Трансформируем алгоритм $\mathfrak{P}$, взяв его композицию с алгоритмом $\mathfrak{S}_2$, который удваивает запись нормального алгорифма: $\mathfrak{S}_2(\mathfrak{A}^3) = \mathfrak{A}^3\delta\mathfrak{A}^3$. Получим нормальный алгорифм $\mathfrak{P}_1 = \mathfrak{P} \circ \mathfrak{S}_2$, обладающий свойством: $\mathfrak{P}_1(\mathfrak{A}^3) = \Lambda \Leftrightarrow \neg!\mathfrak{A}(\mathfrak{A}^3)$, т.е. нормальный алгорифм $\mathfrak{P}_1$ распознает свойство несамоприменимости. Согласно теореме 12.3 такого алгоритма не существует. Значит, предположение о существовании нормального алгорифма $\mathfrak{P}$ неверно, что и доказывает теорему. ▶

Замечание 12.1. Построенный нами нормальный алгорифм $\mathfrak{C}$ имеет большой алфавит: кроме основного алфавита A_2 он использует значительное количество спецсимволов. Однако его можно перевести в эквивалентный нормальный алгорифм в двухбуквенном алфавите, переводя также и исходные слова. В результате мы получаем нормальный алгорифм в двухбуквенном алфавите, применимость которого к словам в этом алфавите не распознается.

Из доказанной теоремы выекают следующие утверждения.

Следствие 12.1. Существует нормальный алгорифм $\mathfrak{C}$ над пятибуквенным алфавитом A_2, обладающий свойством: не существует нормального алгорифма $\mathfrak{P}$ над A_2, применимого ко

всякому слову в A_2 и преобразующего в Λ те и только те слова в алфавите A_2, к которым неприменим $\mathfrak{C}$.

Следствие 12.2. Существует нормальный алгорифм $\mathfrak{C}$ над пятибуквенным алфавитом A_2, обладающий свойством: не существует нормального алгорифма $\mathfrak{P}$ над A_2, применимого ко всякому слову в A_2 и преобразующего в Λ те и только те слова в алфавите A_2, к которым применим $\mathfrak{C}$.

Эти следствия устанавливают неразрешимость проблемы распознавания применимости (неприменимости) алгоритма к слову. Отметим еще один аспект, связанный с этими следствиями. Любой нормальный алгорифм естественно ассоциировать с частично рекурсивной функцией (такая ассоциация возникает в результате процесса арифметизации нормальных алгоритмов). При этом нормальный алгорифм, применимый к любому слову и дающий результат в виде пустого или непустого слова, ассоциируется с общерекурсивным предикатом. При такой трактовке нетрудно увидеть следующую интерпретацию сформулированных следствий: существует рекурсивно перечислимое множество (область применения нормального алгорифма $\mathfrak{C}$), дополнение к которому не является разрешимым, и существует рекурсивно перечислимое множество, не являющееся разрешимым.

Близка к рассмотренной и проблема распознавания аннулирования, состоящая в следующем: существует ли для данного алгоритма $\mathfrak{A}$ над алфавитом A такой нормальный алгорифм $\mathfrak{R}$, который применим к любому слову в A и аннулирующий (преобразующий в пустое слово) те и только те слова алфавита A, которые аннулирует $\mathfrak{A}$.

Простой механизм разветвления позволяет алгоритм $\mathfrak{C}$ над алфавитом A_2 превратить в алгоритм $\mathfrak{B}$, который аннулирует те и только те слова, для которых $\mathfrak{C}$ применим (т.е. либо дает пустое слово, либо неприменим). Достаточно взять композицию алгоритма $\mathfrak{C}$ с алгоритмом, аннулирующим любое слово. В результате проблема распознавания аннулирования для такого алгоритма будет равносильна проблеме распознавания применимости, а последняя не имеет решения.

Множество всех частично рекурсивных функций счетно. Это можно усмотреть разными способами. Например, можно вспомнить о существовании универсальных функций, которое вытекает из теоремы Клини о нормальной форме. Ограничимся функциями одного переменного. Существует такая частично рекурсивная функция U^2, что множество функций $U^2(n,\cdot)$ совпадает с множеством всех частично рекурсивных функций одного переменного. Универсальную функцию можно рассматривать как нумерующую: каждому номеру соответствует некоторая функция и таким способом охватываются все частично рекурсивные функции. Отметим, что этот способ нумерации не гарантирует единственность номера.

Однако важно не только по номеру вычислять функцию, но и наоборот, по функции находить один из ее номеров. А как идентифицировать функцию? Мы не можем этого сделать по ее действию, поскольку для этого надо выполнить бесконечное количество проверок. Два этих момента связаны.

Идентифицировать функции можно следующим образом. Каждую функцию можно связать с нормальным алгорифмом. Нормальный алгорифм в свою очередь можно записать в двухбуквенном алфавите. Действительно, можно ограничиться счетным набором букв и считать, что алфавиты всех схем черпаются из этого множества. Все эти буквы можно закодировать в двухбуквенном алфавите. В результате все нормальные алгорифмы окажутся представленными словами в двухбуквенном алфавите. С каждым нормальным алгорифмом связана частично рекурсивная функция. В результате перенумерации всех нормальных алгорифмов получим нумерацию частично рекурсивных функций. Можно показать, что существует алгоритм, который по номеру определяет нормальный алгорифм, затем соответствующую частично рекурсивную функцию и вычисляет ее значение, т.е. мы имеем алгоритм вычисления функции двух переменных, которая оказывается универсальной.

Чтобы перенумеровать все алгоритмы, достаточно устроить нумерацию всех слов по мере возрастания их длины, а путем проверки, что это запись нормального алгорифма (по формальным признакам), пропустить все лишние слова и тем самым получить нумерацию всех нормальных алгорифмов. В дальнейшем будем рассматривать именно такие нумерации.

Теорема 12.5. Для любой частично рекурсивной функции f^2 существует такая общерекурсивная функция $s(x)$, что $f(x,y) = U^2(s(x), y)$.

◄ Существует нормальный алгорифм $\mathfrak{A}$, вычисляющий функцию $f(x,y)$. Идея такая. Для каждого x мы имеем функцию одного переменного $f(x,\cdot)$. Модификацией алгоритма $\mathfrak{A}$ можно получить нормальный алгорифм, вычисляющий функцию $f(x,\cdot)$: достаточно к исходному слову 01^y0 слева добавить аргумент x и применить алгоритм[**] $\mathfrak{A}$. В результате мы получаем алгоритм, который по первому аргументу x определяет номер соответствующего алгоритма и тем самым номер $s(x)$. ►

Следствие 12.3. Существует общерекурсивная функция $s(n,x)$, для которой

$$U^3(n,x,y) = U^2(s(n,x), y),$$

где U^3 — нумерация всех частично рекурсивных функций двух переменных.

◄ Алгоритм, который по $f(x,y)$ вычисляет ее функцию номера $s(x)$, несложно модифицировать следующим образом: к числу y добавляем пару чисел n и x и применяем алгоритм вычисления U^3. Эту процедуру оформляем в виде записи и по ней определяем номер $s(n,x)$ значение y на самом деле здесь не нужно. В результате есть алгоритм, вычисляющий $s(n,x)$, а это значит, что эта функция общерекурсивна. ►

Установим следующий факт.

Теорема 12.6 (Клини). Для любой частично рекурсивной функции $g(x)$ существует число a такое, что $U^2(g(a),x) = U^2(a,x)$.

◄ Рассмотрим вспомогательную функцию $f(y,x) = U^2(g(s(y,y)),x)$, где s — функция, существование которой утверждается следствием 12.3. Поскольку $f(y,x)$ частично рекурсивна, существует номер n, при котором $f(y,x) = U^3(n,y,x)$. В результате $U^2(g(s(y,y)),x) = U^3(n,y,x)$. Согласно следствию 12.3 $U^3(n,y,x) = U^2(s(n,y),x)$. Поэтому $U^2(g(s(y,y)),x) = U^2(s(n,y),x)$. Положив $y = n$ и обозначив $a = s(n,n)$ с учетом общерекурсивности s получаем $U^2(g(a),x) = U^2(a,x)$. ►

Доказанная теорема позволяет установить удивительный факт. Предположим, что есть свойство P, которым может обладать или не обладать частично рекурсивная функция одного переменного. Опустим очевидные случаи, когда свойством обладают все функции или, наоборот, ни одна из функций одного переменного. Поставим вопрос: существует ли алгоритм, который по номеру l устанавливает, обладает ли свойством P функция $U^2(l,\cdot)$? Оказывается, что эта алгоритмическая проблема неразрешима.

Тезис о свойстве здесь можно опустить: свойство определяет некоторое подмножество в множестве всех частично рекурсивных функций одного переменного, причем это подмножество собственное (не пусто и не есть все множество). Существование распознающего алгоритма равносильно существованию общерекурсивного предиката, вычисляемого алгоритмом, который для данного x принимает значение 1, если функция с номером x обладает свойством P, и значение 0 в противном случае. Иначе говоря, разрешимость поставленной проблемы равносильна тому, что множество номеров всех частично рекурсивных функций, обладающих свойством

[**]Правда, здесь есть тонкость: есть часть алгоритмов, работающих с системами чисел и тем самым вычисляющих функцию, а есть процедура арифметизации, которая каждый алгоритм трансформирует в функцию. Надо, чтобы при этом характер первых алгоритмов не изменился.

P, разрешимо. Отметим, что множество самих частично рекурсивных функций может быть разрешимым, например, если оно конечно.

Теорема 12.7 (Райс). Пусть P — собственное подмножество множества всех частично рекурсивных функций одного переменного. При любой нумерации частично рекурсивных функций множество номеров функций, принадлежащих P, не разрешимо.

◀ Предположим, что множество номеров $N(P)$ функций, принадлежащих P, разрешимо. Тогда и дополнение $\overline{N(P)}$ разрешимо, т.е. обе характеристические функции χ_N и $\chi_{\overline{N}}$ множеств $N(P)$ и $\overline{N(P)}$ являются общерекурсивными. По условию оба множества не пусты, поэтому можно выбрать $\alpha \in N(P)$ и $\beta \notin N(P)$. Рассмотрим функцию

$$g(x) = \begin{cases} \alpha, & x \notin N(P); \\ \beta, & x \in N(P). \end{cases}$$

В силу представления $g(x) = \alpha\chi_{\overline{N}}(x) + \beta\chi_N(x)$ делаем заключение, что функция $g(x)$ общерекурсивна.

По функции $g(x)$ выберем число a в соответствии с теоремой Клини и рассмотрим функцию $f(x) = U(g(a), x)$, которая, очевидно, частично рекурсивна. Интересно, принадлежит ли число a множеству $N(P)$?

Предположим, что $a \in N(P)$. Тогда $g(a) = \beta \notin N(P)$. В результате, с одной стороны, $U(a, \cdot) \in P$, а с другой, $U(a, \cdot) = U(g(a), \cdot) = U(\beta, \cdot) \notin P$. Предположение $a \notin P$ также приводит к противоречию. Следовательно, исходное предположение о разрешимости $N(p)$ неверно. ▶

13. СЛОЖНОСТЬ АЛГОРИТМОВ

Богатая практика программирования в наше время естественным образом формирует представления о сложности программы. Мы бы сказали, что более сложной является программа, потребляющая большее количество ресурсов. Ключевыми ресурсами в современной вычислительной практике являются память и время. Это и ложится в основу понятий со сложности алгоритмов.

Заметим, что используемую программой память можно разделить на две части: память для размещения самой программы и память для размещения данных. Мы сосредоточимся только на 1-й части, т.е. на оценке емкости самой программы.

Разумеется, используемые ресурсы зависят от того, в какой среде реализуется программа. В контексте теории алгоритмов можно сказать так: сложность алгоритма зависит от используемой вычислительной модели. Мы коснемся двух вычислительных моделей: нормальных алгорифмов и однолентоных машин Тьюринга.

Рекурсивные функции стоят здесь особняком, поскольку механизм рекурсии можно уподобить неалгоритмическому описанию соответствующего преобразования. Подобные описания нашли свое применение в практике программирования. Эту вычислительную модель в контексте сложности мы обсуждать не будем.

13.1. Сложность нормальных алгорифмов

Сложностью нормального алгорифма $S(\mathfrak{A})$ называется количество знаков в изображении этого алгорифма, т.е. $S(\mathfrak{A}) = |\mathfrak{A}^{\text{ц}}|$. ***Сложность времени работы*** $t_{\mathfrak{A}}(X)$ алгоритма $\mathfrak{A}$ над словом X — это число шагов работы алгоритма для данного слова X. Если алгоритм неприменим к слову X, считаем, что $t_{\mathfrak{A}}(X) = \infty$.

Рассмотрим проблему распознавания свойства P, которым могут обладать или не обладать слова в алфавите A. Такая проблема в целом неразрешима. Однако разрешимой может быть ***ограниченная проблема*** — проблема существования алгоритма, распознающего свойство P для слов длины не более n. Отсутствие алгоритма для всего множества слов, в частности, вытекает из того, что сложность алгоритма, решающего ограниченную проблему, растет с ростом предельной длины слова.

Назовем верхней оценкой сложности распознавания свойства P такую функцию $f(n)$, что для каждого n существует нормальный алгорифм, решающий n-ограниченную проблему распознавания свойства P и имеющий сложность не выше $f(n)$. Нижней оценкой назовем такую функцию $g(n)$, что ни один алгоритм, решающий n-ограниченную проблему распознавания, не может иметь сложность менее $g(n)$.

Простенькая теорема о сложности распознавания самоприменимости показывает, почему эта массовая проблема неразрешима.

Теорема 13.1. Пусть N — натуральное число и $\mathfrak{R}$ — нормальный алгорифм в алфавите A, имеющем не менее четырех букв, применимый к записи нормального алгорифма $\mathfrak{A}$ в алфавите A сложности $S(\mathfrak{A}) \leqslant N$ тогда и только тогда, когда $\mathfrak{A}$ несамоприменим. Тогда $S(\mathfrak{R}) > N$.

◄ Если $S(\mathfrak{R}) \leqslant N$, то его можно применить к своей записи. При этом согласно условию, если он несамоприменим, то он применим к своей записи, т.е. самоприменим и, наоборот, если он самоприменим, то он неприменим к своей записи, т.е. несамоприменим. Противоречие показывает, что на самом деле $S(\mathfrak{R}) > N$. ►

Переходим к оценке сложности проблемы распознавания несамоприменимости. Зафиксируем четырехбуквенный алфавит $A_1 = abcd$. Для произвольного нормального алгорифма $\mathfrak{A}$ в алфавите A_1 построим алгоритм $\mathfrak{B}$ в алфавите $A_1 e$ со схемой

$$\begin{cases} e\xi \to\cdot\ \Lambda, \\ \xi e \to\cdot\ \Lambda, \\ e \to e, \\ \mathfrak{A}_e, \end{cases}$$

где $\mathfrak{A}_e$ — нормальный алгорифм, полученный из замыкания $\mathfrak{A}$ заменой каждой терминальной подстановки $P \to\cdot\ Q$ подстановкой $P \to eQ$.

Нетрудно увидеть, что явно построена композиция двух алгоритмов, причем второй есть „зацикливатель“ $\xi \to\cdot\ \Lambda$, $\Lambda \to \Lambda$. Алгоритм $\mathfrak{B}$ применим к слову X в A_1 тогда и только тогда, когда $\mathfrak{A}(X) \neq \Lambda$. Устроим перевод $\mathfrak{B}^\tau$ алгоритма $\mathfrak{B}$ в алфавит A_1, используя в качестве базы $A_0 = ab$ и переводя остальные буквы по правилам $c \to cdc$, $d \to cd^2c$, $e \to cd^3c$. Оценим сложность $\mathfrak{B}^\tau$ через сложность $\mathfrak{A}$, полагая, что схема алгоритма $\mathfrak{A}$ имеет k подстановок.

Число букв алфавита A_1 в схеме $\mathfrak{A}$ можно подсчитать, вычтя все служебные символы: k символов α или β и $k+1$ символ γ. Получается $S(\mathfrak{A}) - (2k+1)$. Схема алгоритма $\mathfrak{A}_e$ содержит дополнительную подстановку, но число символов алфавита A_1 будет то же самое. При переходе к переводу этого алгорифма каждая буква алфавита A_1 заменяется словом длины не более 4. Кроме того, в этом переводе будет до $k+1$ букв e, каждая из которых заменяется словом длины 5. В результате получим оценку

$$4(S(\mathfrak{A}) - (2k+1)) + 5(k+1) + 2k + 3 = 4S(\mathfrak{A}) - k + 4.$$

Первые три строки (фактически это 9 подстановок) добавляют еще $(5 \cdot 4 + 4 + 3 + 1 + 1 + 4) \cdot 2 + 11 + 9 = 86$ букв. В результате

$$S(\mathfrak{B}^\tau) \leqslant 4S(\mathfrak{A}) - k + 90 \leqslant 4S(\mathfrak{A}) + 89. \tag{13.1}$$

Полученная оценка позволяет доказать следующую теорему.

Теорема 13.2. Пусть натуральное число N и нормальный алгорифм $\mathfrak{C}$ в A_1 таковы, что $\mathfrak{C}$ применим к записи всякого алгоритма $\mathfrak{A}$ в A_1 с $S(\mathfrak{A}) \leqslant N$, причем $\mathfrak{C}(\mathfrak{A}^3) = \Lambda$ тогда и только тогда, когда $\mathfrak{A}$ несамоприменим. Тогда $S(\mathfrak{C}) > \dfrac{N-89}{4}$.

◀ По алгоритму $\mathfrak{C}$ построим алгоритм $\mathfrak{B}^\tau$, как описано выше. Тогда алгоритм $\mathfrak{B}^\tau$ будет неприменим к записи алгоритма $\mathfrak{A}$ с $S(\mathfrak{A}) \leqslant N$ тогда и только тогда, когда $\mathfrak{C} = \Lambda$, т.е. когда $\mathfrak{A}$ несамоприменим. По доказанному в теореме 13.1 $S(\mathfrak{B}^\tau) > N$, а согласно неравенству (13.1) получаем

$$S(\mathfrak{C}) \geqslant \frac{S(\mathfrak{B}^\tau) - 89}{4} > \frac{N-89}{4}.$$

что и требовалось доказать. ▶

Замечание 13.1. Из доказанной теоремы немедленно вытекает, что общая проблема несамоприменимости неразрешима: для разрешающего алгоритма $\mathfrak{C}$, если он существует, получаем, что $S(\mathfrak{C}) > \dfrac{N-89}{4}$ для любого натурального N, а это невозможно.

Замечание 13.2. Аналогичным образом можно получить оценку для алгоритмов, решающих ограниченную проблему самоприменимости. Для этого взамен алгоритма $\mathfrak{B}$ строим алгоритм $\mathfrak{B}'$ со схемой

$$\begin{cases} e\xi \to e\xi, \\ \xi e \to \xi e, \\ e \to\cdot\ \Lambda, \\ \mathfrak{A}_e, \end{cases}$$

Первые четыре строки в схеме дают $[2\cdot(5\cdot 4+4+3+1+1)+4]\cdot 2+6+9=139$ символов (было 86). Поэтому, повторяя рассуждения, заключаем, что нормальный алгорифм $\mathfrak{C}'$, решающий N-ограниченную проблему самоприменимости, имеет оценку сложности

$$S(\mathfrak{C}')>\frac{N-142}{4}.$$

Скажем, что нормальный алгорифм $\mathfrak{C}$ над алфавитом A решает N-ограниченную проблему применимости нормального алгорифма $\mathfrak{A}$ к словам в алфавите A, если для всякого слова $X\in A^*$ длины не более N, во-первых, $!\mathfrak{C}(X)$, а во-вторых, $\mathfrak{C}(X)=\Lambda\Leftrightarrow !\mathfrak{A}(X)$.

Напомним, что проблема применимости была сведена к проблеме несамоприменимости. Уточним это сведение, подсчитав соответствующее количество символов. Напомним, что, видоизменив универсальный алгорифм $\mathfrak{R}$, мы построили нормальный алгорифм $\mathfrak{R}_1$ над алфавитом A_1e, который удовлетворяет условию $\mathfrak{R}_1(\mathfrak{A}^3eX)\simeq\mathfrak{A}(X)$. Очевидная модификация с соединением приводит к нормальному алгорифму $\mathfrak{R}_2$, для которого $\mathfrak{R}_2(X)\simeq\mathfrak{R}_1(XeX)$. Для этого алгоритма, в частности, $\mathfrak{R}_2(\mathfrak{A}^3)\simeq\mathfrak{A}(\mathfrak{A}^3)$, т.е. алгоритм $\mathfrak{R}_2$ распознает самоприменимость алгоритма. Осталось перевести его в алгоритм $\mathfrak{R}_3$ в алфавите A_1, имея в виду, что записи нормальных алгорифмов используют только буквы a и b.

При отсутствии ограничений на длину алгоритма эта конструкция приводила к противоречию, поскольку допускалось рассматривать применимость построенного алгоритма к самому себе. Однако введение на ограничение длины записи эту проблему снимает и мы лишь можем утверждать, что запись построенного алгоритма не проходит по длине.

Теорема 13.3. Пусть N — произвольное натуральное число. Каков бы ни был алгоритм $\mathfrak{C}$ в алфавите A_1, решающий N-ограниченную проблему неприменимости алгоритма $\mathfrak{R}_3$ к словам в алфавите A_0, верна оценка

$$S(\mathfrak{C})>\frac{N}{36}-\frac{45}{2}.$$

◀ Если нормальный алгорифм $\mathfrak{C}$ решает проблему неприменимости алгоритма $\mathfrak{R}_3$ к словам в алфавите A_0, то, в частности, он применим к записи любого алгоритма с длиной записи не более N, причем $\mathfrak{C}(\mathfrak{A}^3)=\Lambda\Leftrightarrow\neg!\mathfrak{A}(\mathfrak{A}^3)$.

Отметим, что для любого нормального алгорифма $\mathfrak{A}$ в алфавите A_1 имеем $|\mathfrak{A}^3|\leqslant 9|\mathfrak{A}^{\text{и}}|$, поскольку изображение строится в семибуквенном алфавите $abcd\alpha\beta\gamma$ и при переводе семи букв в двухбуквенный алфавит каждая будет записана максимум 9 буквами.

Пусть $\mathfrak{A}$ — нормальный алгорифм в a_1 с длиной изображения (сложностью) не более $\left[\frac{N}{9}\right]$ (целой части дроби). Тогда $|\mathfrak{A}^3|\leqslant N$. Следовательно, $!\mathfrak{C}(\mathfrak{A}^3)$ и $\mathfrak{C}(\mathfrak{A}^3)=\Lambda\Leftrightarrow\neg!\mathfrak{A}(\mathfrak{A}^3)$. Мы видим, что $\mathfrak{C}$ решает K-ограниченную проблему несамоприменимости с $K=\left[\frac{N}{9}\right]$. Поэтому

$$S(\mathfrak{C})>\frac{K-89}{4}\geqslant\frac{1}{4}\Big(\frac{N}{9}-1-89\Big)=\frac{N}{36}-\frac{45}{2},$$

что и требовалось доказать. ▶

Замечание 13.3. Учитывая замечание 13.2, нетрудно получить оценку сложности для алгоритма, решающего N-ограниченную проблему применимости к слову:

$$S(\mathfrak{C}')>\frac{N}{36}-\frac{143}{4}.$$

Теорема 13.3 и замечание 13.3 устанавливают нижнюю оценку для ограниченной проблемы неприменимости или применимости к слову. Существуют и верхние оценки, имеющие тот порядок роста, что и нижние оценки.

Теорема 13.4. Каков бы ни был нормальный алгорифм $\mathfrak{R}$ над алфавитом A_0, существует такая постоянная k, что для любого N существует нормальный алгорифм $\mathfrak{C}$ сложности не более $N+k$, решающий N-ограниченную проблему применимости $\mathfrak{R}$ к словам алфавита A_0.

13.2. Сложность машин Тьюринга

Пусть задана машина Тьюринга T с внешним алфавитом $A = a_0a_1 \dots a_n$, и внутренним алфавитом $Q = q_0q_1 \dots q_m$. ***Сложностью*** $S(T)$ ***машины Тьюринга*** T называется максимальная емкость программы, равная $m(n+1)$. ***Временно́й сложностью*** (сложностью времени работы) $t_T(K)$ на конфигурации K называется количество шагов, сделанных машиной Тьюринга начиная с конфигурации K до завершения работы. Если машина Тьюринга неприменима к данной конфигурации, то считаем, что $t_T(K) = \infty$.

Для данной вычислимой словарной функции f можно указать много машин Тьюринга, ее вычисляющих. В каких пределах меняется сложность этих машин? Назовем число l ***верхней оценкой сложности*** для функции f, если существует машина Тьюринга, вычисляющая f, сложности не выше l. Число k называется ***нижней оценкой сложности***, если любая машина Тьюринга, вычисляющая f, имеет сложность не менее k.

При отсутствии алгоритма, решающего ту или иную массовую проблему, можно рассматривать серию алгоритмов, решающих ограниченную массовую проблему. Допустим, что решение массовой проблемы выражается как вычисление некоторой словарной функции $f(X)$. Отсутствие алгоритма означает, что эта функция невычислима. Рассмотрим N-ограниченную проблему, состоящую в вычислении функции $f(X)$ для слов длины $|X| \leqslant N$. Таких слов в данном алфавите конечное число и, следовательно, ограниченная проблема разрешима. Разрешающих алгоритмов может быть много. Скажем, что функция $h(N)$ есть верхняя оценка сложности вычисления функции $f(X)$, если для любого N существует машина Тьюринга, вычисляющая $f(X)$ при $|X| \leqslant N$ и имеющая сложность не выше $h(N)$. Нижней оценкой сложности вычисления функции $f(X)$ называется любая функция $g(N)$, такая, что для любая машина Тьюринга, решающая N-ограниченную проблему вычисления $f(X)$, имеет сложность не ниже $g(N)$. Аналогичные определения вводятся и для временной сложности.

По поводу временно́й сложности отметим следующий факт.

Теорема 13.5 (теорема Блюма об ускорении). Пусть r — общерекурсивная функция. Существует общерекурсивная функция f, такая, что для любой машины Тьюринга T, вычисляющей f, существует машина Тьюринга T_1, также вычисляющая f, для которой $r(t_{T_1}(n)) < t_T(n)$ начиная с некоторого номера n.

Пусть $r(n) = 2^n$ (это, как легко показать, общерекурсивная функция). Существует функция f, обладающая следующим свойством. Если T — машина Тьюринга, вычисляющая f, то существует машина Тьюринга T_1, также вычисляющая f, для которой $t_{T_1}(n) < \log_2 t_T(n)$ при больших n. Далее, существует машина Тьюринга T_2, также вычисляющая f, для которой $t_{T_2}(n) < \log_2 t_{T_1}(n) < \log_2 \log_2 t_T(n)$ при больших n. Этот процесс можно продолжить.

13.3. Классы сложности P и NP

Теорема Блюма ставит крест на попытке ввести понятие оптимального, т.е. наименьшего по сложности, алгоритма. Развитие теории сложности пошло по пути выделения классов сложности алгоритмов. Во всех этих задачах рассматриваются так называемые ***распознающие алгоритмы***, которые применимы к любому слову и дают два возможных ответа 1 (да) и 0 нет. Этим алгоритмам соответствует понятие общерекурсивный предикат. Им можно придать различную трактовку: распознавание свойства, которым может обладать слово в данном алфавите, распознавание принадлежности слова множеству, проверка тех или иных утверждений о словах в данном алфавите и т.п.

Можно в самом общем виде так поставить массовую проблему. Есть бесконечное (счетное) множество однотипных задач, зависящих от некоторого набора параметров. Мы кодируем эти параметры с помощью некоторого набора символов (алфавита) и тем самым ставим каждой конкретной задаче слово в зафиксированном алфавите. Решение задачи — ответ „да“ или

„нет“. Мы приходим в результате к задаче вычислимости некоторого словарного предиката в заданном алфавите.

В значительной части литературы по этой тематике используют такую трактовку. Любое множество слов в данном алфавите называют ***языком***. Иначе говоря, язык в алфавите A — это любое подмножество $L \subset A^*$. Алгоритм T (машина Тьюринга или другая вычислительная модель) распознает язык L, если он применим к любому слову в данном алфавите и $T(X) = 1$ тогда и только тогда, когда $X \in L$. Напомним, что в этом контексте можно ограничиться двухбуквенным алфавитом.

Пусть задан алфавит A и словарная функция f: $A^* \to A^*$ вычислима по Тьюрингу. Скажем, что машина Тьюринга T, вычисляющая f, полиномиальная (вычисляет f за полиномиальное время), если существует такой полином $Q(t)$, что для любого слова $X \in A^*$ имеем $t_T(X) \leqslant Q(|X|)$, где $|X|$ — длина слова X. ***Словарная функция f полиномиальная*** (вычислима за полиномиальное время), если существует машина Тьюринга, вычисляющая f за полиномиальное время. Множество таких функций обозначим через П.

Среди словарных функций выделим ***словарные предикаты*** — всюду определенные функции, принимающие лишь два значения 0 и 1. Словарный предикат представляет собой характеристическую функцию некоторого множества слов в рассматриваемом алфавите.

Скажем, что машина Тьюринга T распознает язык L за полиномиальное время, если она вычисляет характеристическую функцию этого языка за полиномиальное время. ***Язык L распознаваем за полиномиальное время***, tсли его характеристическая функция полиномиальная. Класс таких языков обозначим через P.

Разумеется, вычислимость словарного предиката не зависит от выбора вычислительной модели, но сложность для каждой модели своя. Например, можно поставить вопрос о временной сложности вычисления данного предиката в рамках нормальных алгорифмов и в рамках машин Тьюринга какой-либо модификации. Однако можно показать, что перенос алгоритма из одной вычислительной модели в другую (среди известных) приводит к полиномиальному увеличению сложности. Это значит, что класс P, формально вводимый для каждой вычислительной модели, на самом деле одинаков для широкого круга таких вычислительных моделей.

В качестве эталонных при оценке сложности выбирают одноленточные машины Тьюринга как более элементарные, например, по сравнению с нормальными алгорифмами. Поэтому те или иные оценки для машин Тьюринга получить легче.

Говорят о задачах полиномиальной сложности, а также о задачах экспоненциальной сложности — задачах, сложность которых не может быть оценена сверху никаким полиномом. Отметим, что к экспоненциальным относят, например, и задачи, временная сложность которых растет как $e^{\ln^2 n}$ (т.е. чуть выше полиномиальной). С практической точки зрения проблемы полиномиальной сложности отличает то, что при увеличения быстродействия в несколько раз, сложность задачи, решаемой за приемлемое время также повышается в несколько раз, т.е. здесь зависимость мультипликативная. При экспоненциальной сложности зависимость аддитивная.

Рассмотрим, к примеру, задачу проверки выполнимости КНФ. Можно выполнить 2^n операций вычисления значения КНФ на всевозможных наборах булевых переменных. И длина КНФ, записанной в том или ином алфавите, и алгоритм вычисления ее значения на конкретном наборе значений переменных полиномиально зависят от количества переменных. А в целом рассматриваемый алгоритм перебора значений имеет экспоненциальную сложность. Алгоритм полиномиальной сложности для этой задачи не известен. Однако ситуация меняется, если количество переменных в каждой элементарной дизъюнкции ограничено. Например, если в каждой элементарной дизъюнкции всего две переменных, то метод резолюций приводит к перебору не более $\frac{n(n+1)}{2}$ пар и даст ответ за полиномиальное время.

Еще одна задача — проверка, является ли отношение, заданное на конечном множестве, отношением эквивалентности. Здесь надо проверить три условия: симметричность, рефлексивность и транзитивность. Проверка каждого из этих условий имеет полиномиальную временную сложность. Таким образом, эта задача — из класса P.

Класс сложности NP аналогичен классу сложности P, но за основу взяты так называемые ***недетерминированные машины Тьюринга***. Такая машина устроена в целом так же, как и обычная, но для каждого состояния она позволяет несколько вариантов перехода, т.е. паре „состояние — обозреваемый символ“ соответствует несколько команд, и она выбирает любую.

Для недетерминированной машины Тьюринга для каждого слова (или для всех) есть как бы несколько вариантов ответа. Считаем, что результат ее работы 1, если хотя бы один ответ 1, иначе 0. Временем работы недетерминированной машины на данном слове называется либо шаг первого появления результата 1, либо шаг заершения последнего варианта, если все ответы 0.

Детерминированную машину Тьюринга можно рассматривать как частный случай недетерминированной. Из того, что представляет собой результат работы недетерминированной машины Тьюринга, заключаем, что речь идет не о выборе случайного варианта работы (для этого есть вероятностные машины), а об анализе все вариантов работы машины. Можно показать, что работу любой недетерминированной машины можно реализовать с помощью соответствующей детерминированной машины, которая при появлении нескольких вариантов на ленте дублирует информацию и параллельно ведет каждый из этих вариантов. Другими словами, недетерминированная машина — это теоретическое описание переборного алгоритма.

Хотя класс разрешимых проблем для детерминированной и недетерминированной машин совпадают, эти машины различаются по сложности вычислений. Класс NP — это класс проблем, для которых существует разрешающая недетерминированная машина Тьюринга полиномиальной временно́й сложности. Очевидно, что $P \subset NP$. Но какое это включение строгое или нет? Ответ не известен. Анализу этой проблемы посвящено довольно много усилий.

Недетерминированная машина — удобный способ описания переборных алгоритмов, таких как поиск в глубину в графе. При линейном переборе мы имеем дело с полиноимальной задачей. Но если перебор идет по дереву (например, выводимость в исчислении высказываний), то сложность возрастает до экспоненциальной.

Отметим альтернативное определение класса NP. Язык $L \in A^*$ принадлежит классу NP, если существует полином $p(x)$ и полиномиальная распознающая детерминированная машина Тьюринга T, такие, что $X \in L$ тогда и только тогда, когда существует слово Y, $|Y| \leqslant p(|X|)$, для которого $T(Y + X) = 1$.

Пусть $L \in NP$. Если $X \in L$, существует вариант работы недетерминированной машины, приводящий к значению 1. Достаточно записать в качестве слова Y информацию о правильный выборе при разветвлении в недетерминированной машине, а затем повторить ее работу по единственной ветке. Наоборот, пусть детерминированный алгоритм работает, если исходное слово „подправлено“. Тогда можно составить недетерминированный алгоритм полиномиальной сложности, формирующий „подсказку“.

Например, задача проверки выполнимости КНФ относится к классу NP, так как достаточно добавить конкретный набор значений переменных, при которых КНФ истинна, а затем запустить полиномиальный алгоритм вычисления значения КНФ на известном наборе значений переменных.

Принадлежность той или иной задачи данному классу можно попытаться установить непосредственно. Но чаще выгоднее свести такую задачу к принадлежности классу другого алгоритма. Будем говорить о языках. Скажем, что язык $L_1 \in A^*$ сводим к языку $L_2 \in A^*$, если существует вычислимая словарная функция f полиномиальной сложности, удовлетворяющая условию: $X \in L_1 \Leftrightarrow f(X) \in L_2$ (f — сводящая функция). Обозначаем $L_1 \preccurlyeq L_2$.

Если для L_2 существует распознающий алгоритм полиномиальной сложности, то для любого слова X мы сперва вычисляем $f(X)$, затрачивая полиномиальное время, а затем применяем распознающий алгоритм. Получаем распознающий L_1 алгоритм полиномиальной сложности. Другими словами, если $L_2 \in P$ и $L_1 \preccurlyeq L_2$, то и $L_1 \in P$. То же, очевидно, относится и к классу NP.

Отношение сводимости, очевидно, рефлексивное и транзитивное, но не антисимметричное (существуют разные языки, сводимые друг к другу). Скажем, что языки L_1 и L_2 эквивалентны, если $L_1 \preccurlyeq L_2$ и $L_2 \preccurlyeq L_1$. Получим отношение эквивалентности.

Язык $L \in NP$ называется NP-полным, если к нему сводится любой язык класса NP. Вопрос: существуют ли NP-полные проблемы, принадлежащие классу P? Этот вопрос эквивалентен равенству $P = NP$. Действительно, если существует NP-полный язык из P, то автоматически любой язык из NP попадает в P как сводимый к языку из P. Обратное утверждение очевидно. Пока таких языков не нашли. Существуют ли вообще NP-полные языки? Ответ да. Один из таких возникает в задаче проверки выполнимости КНФ. Множество выполнимых КНФ (а точнее их записей) обозначим SAT. Полнота этого языка объясняется тем, что любую переборную задачу можно оформить как проверку выполнимости (истинности хотя бы в одном варианте) некоторой КНФ.

ОГЛАВЛЕНИЕ